lecture and hands-on

presentation

IoT & Embedded OS Security

Internet of Things & Embedded OS Security

Cristian TOMA
E-mail: cristian.toma@ie.ase.ro
Web: ism.ase.ro | acs.ase.ro | dice.ase.ro

Business Card

Agenda for IoT & eMbedded OS

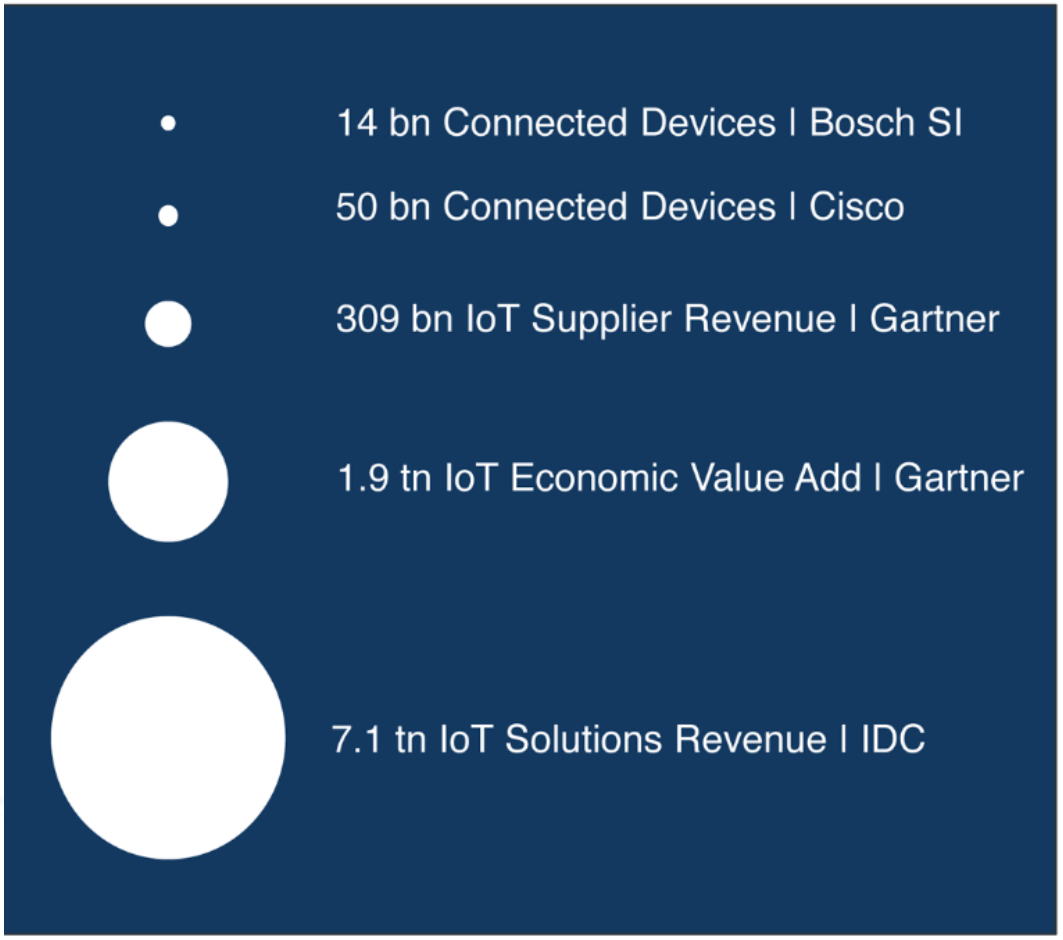
IoT Clouds, Device Communications Protocols – REST/MQTT/CoAP, IoT Java DIO
Hands-on: Java DIO and MQTT/HTTP-Rest/CoAP on Raspberry Pi, Node-RED/Node.js

Internet of Things

1. IoT Overview

"Mission: 50 Billion Connected Devices by 2020"

Some Big Numbers:



Some Small Numbers:

Peter Middleton, Gartner:
By 2020, component costs will have come down to the point that connectivity will become a standard feature, even for processors costing less than

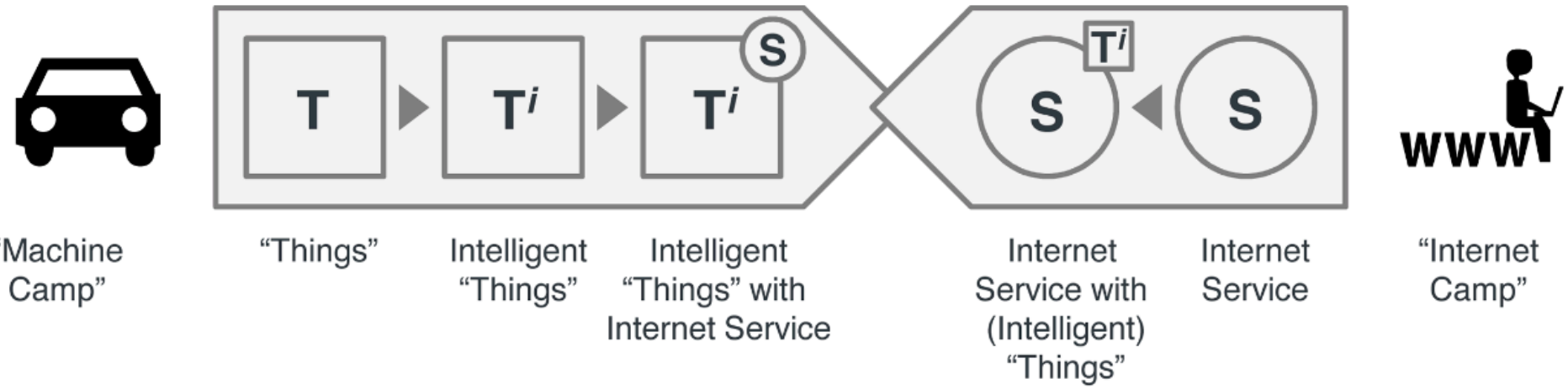
$1

Data from http://postscapes.com/internet-of-things-market-size

Copyright: Excerpt From: Dirk Slama, Frank Puhlmann, Jim Morrish, and Rishi M. Bhatnagar - "Enterprise IoT", O'Reilly Print House

1. IoT Overview

IoTS – Internet of Things Services Formula: "Difficulty of Finding the Right Service"



"First, we equipped our light bulbs with sensors, so they would only be on if somebody was in the room. Next, we added an iPhone app so you can manage all of the electric lights in your home."

"Our online security service is now using connected light bulbs to simulate a realistic lighting pattern when you are away from home at night."

Copyright: Excerpt From: Dirk Slama, Frank Puhlmann, Jim Morrish, and Rishi M. Bhatnagar - "Enterprise IoT", O'Reilly Print House

1. IoT Overview

IoTS – Internet of Things Definition

- ***Wiki:*** **Internet of Things (IoT) is the network of physical objects—devices, vehicles, buildings and other items embedded with electronics, software, sensors, and network connectivity—that enables these objects to collect and exchange data.[1] The Internet of Things allows objects to be sensed and controlled remotely across existing network infrastructure,[2] creating opportunities for more direct integration of the physical world into computer-based systems, and resulting in improved efficiency, accuracy and economic benefit;[3][4][5][6][7][8] when IoT is augmented with sensors and actuators, the technology becomes an instance of the more general class of cyber-physical systems, which also encompasses technologies such as smart grids, smart homes, intelligent transportation and smart cities. Each thing is uniquely identifiable through its embedded computing system but is able to interoperate within the existing Internet infrastructure. Experts estimate that the IoT will consist of almost 50 billion objects by 2020.[9]**

Copyright: Excerpt From: Dirk Slama, Frank Puhlmann, Jim Morrish, and Rishi M. Bhatnagar - "Enterprise IoT", O'Reilly Print House

1. IoT Overview

IoTS – Internet of Things Context

"**Moore's law:** Ever-increasing hardware performance enables new levels of abstraction in the embedded space, which provides the basis for semantically rich embedded applications and the decoupling of on-asset hardware and software lifecycles. The app revolution for smartphones will soon be replicated in the embedded space.

Wireless technology: From ZigBee to Bluetooth LE, and from LTE/4G to specialized low-power, wide-area (LPWA) IoT communication networks—the foundation for "always-on" assets and devices is either already available or in the process of being put in place.

Metcalfe's law: Information and its value grow exponentially as the number of nodes connected to the IoT increases. With more and more remote assets being connected, it looks like we are reaching a tipping point.

Battery technology: Ever-improving battery quality enables new business models, from electric vehicles to battery-powered beacons.

Sensor technology: Ever-smaller and more energy-efficient sensors integrated into multi-axis sensors and sensor clusters, an increasing number of which are preinstalled in devices and assets.

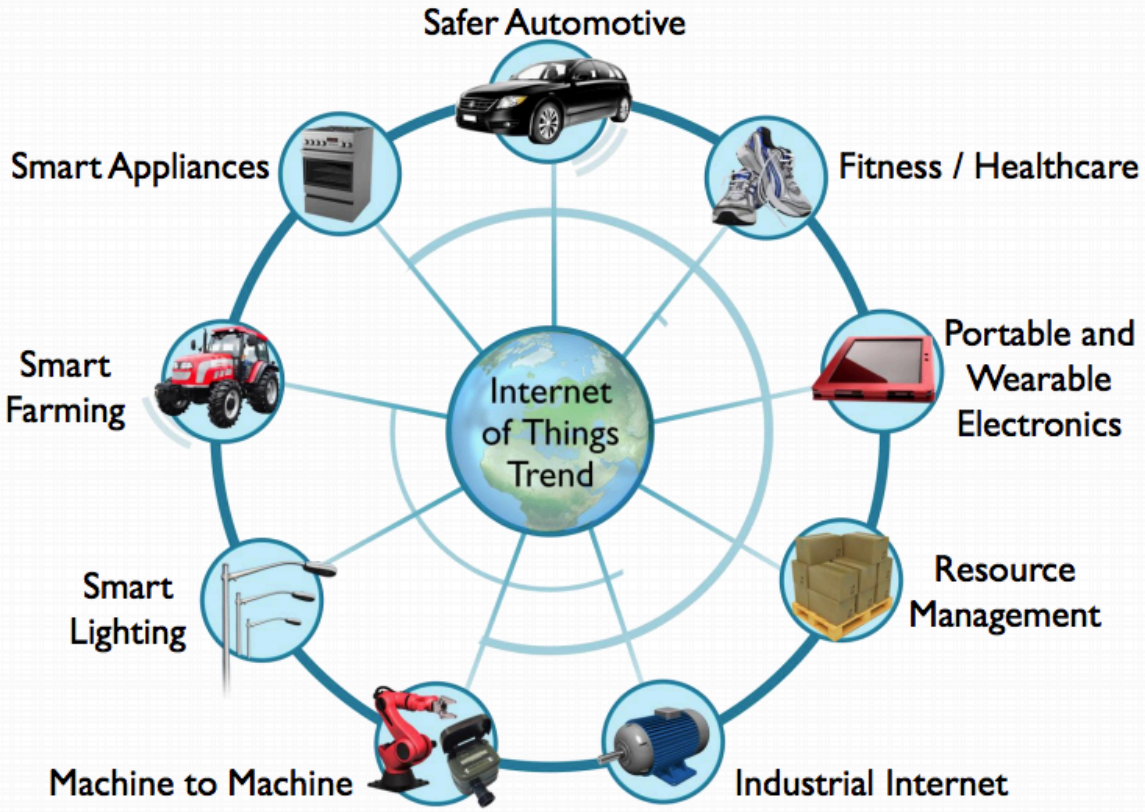
Big Data: Technology that is able to ingest, process, and analyze the massive amounts of sensor-generated data at affordable cost.

The cloud: The scalable, global platform that delivers data-centric services to enable new IoT business models."

Copyright: Excerpt From: Dirk Slama, Frank Puhlmann, Jim Morrish, and Rishi M. Bhatnagar - "Enterprise IoT", O'Reilly Print House

IoTS – Internet of Things Intro

The "Internet of Things" Drives Opportunities

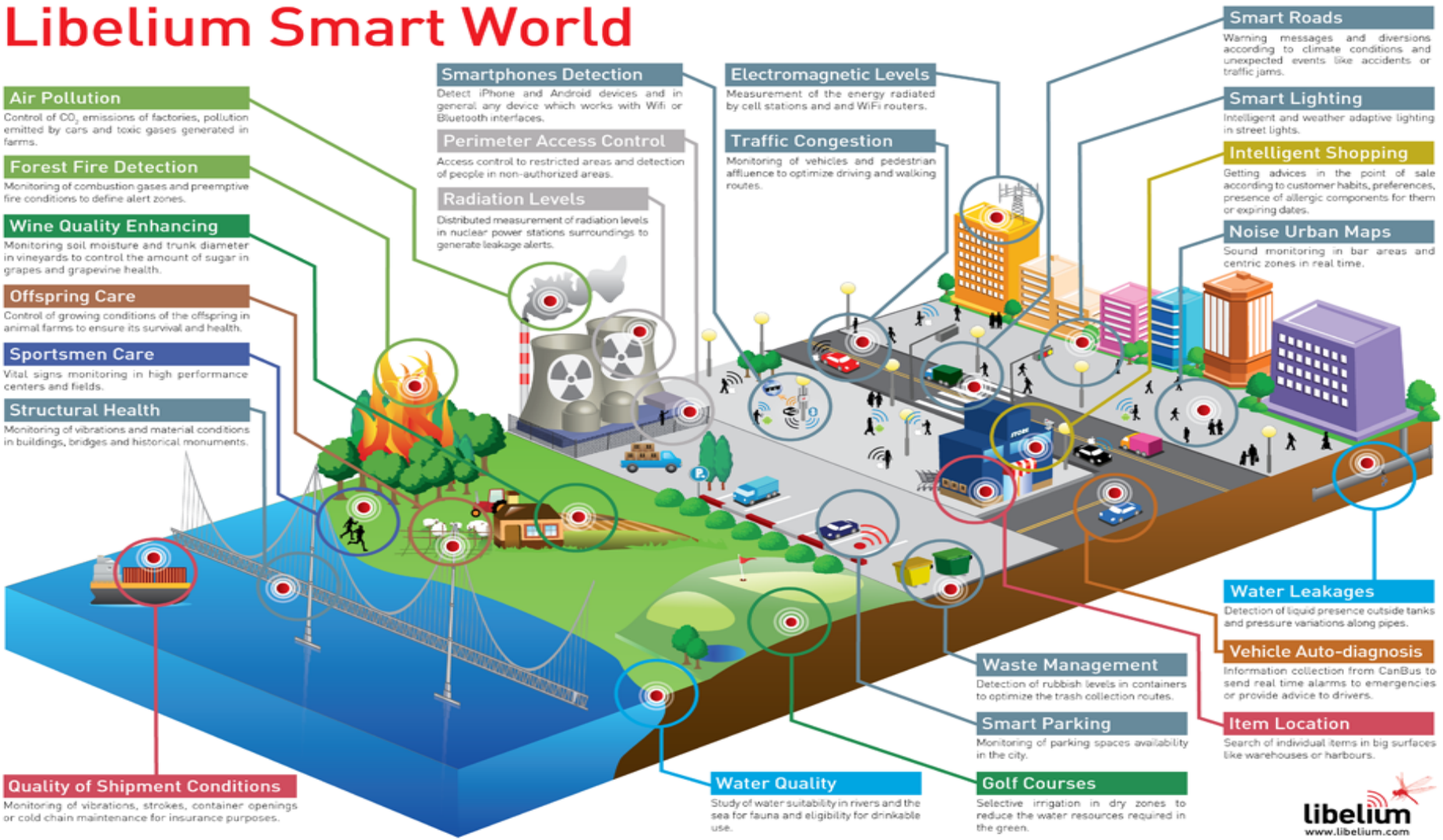


and Challenges

- Always on
- Increased connectivity
- Confidentiality
- Security
- Regulations
- New users

1. IoT Overview

IoTS – Internet of Things Smart Cities Solutions

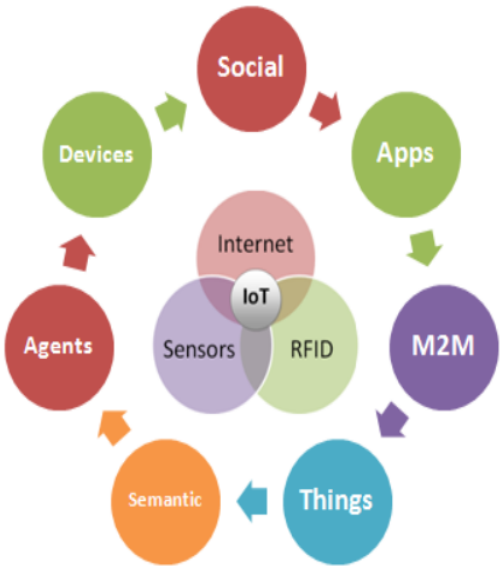


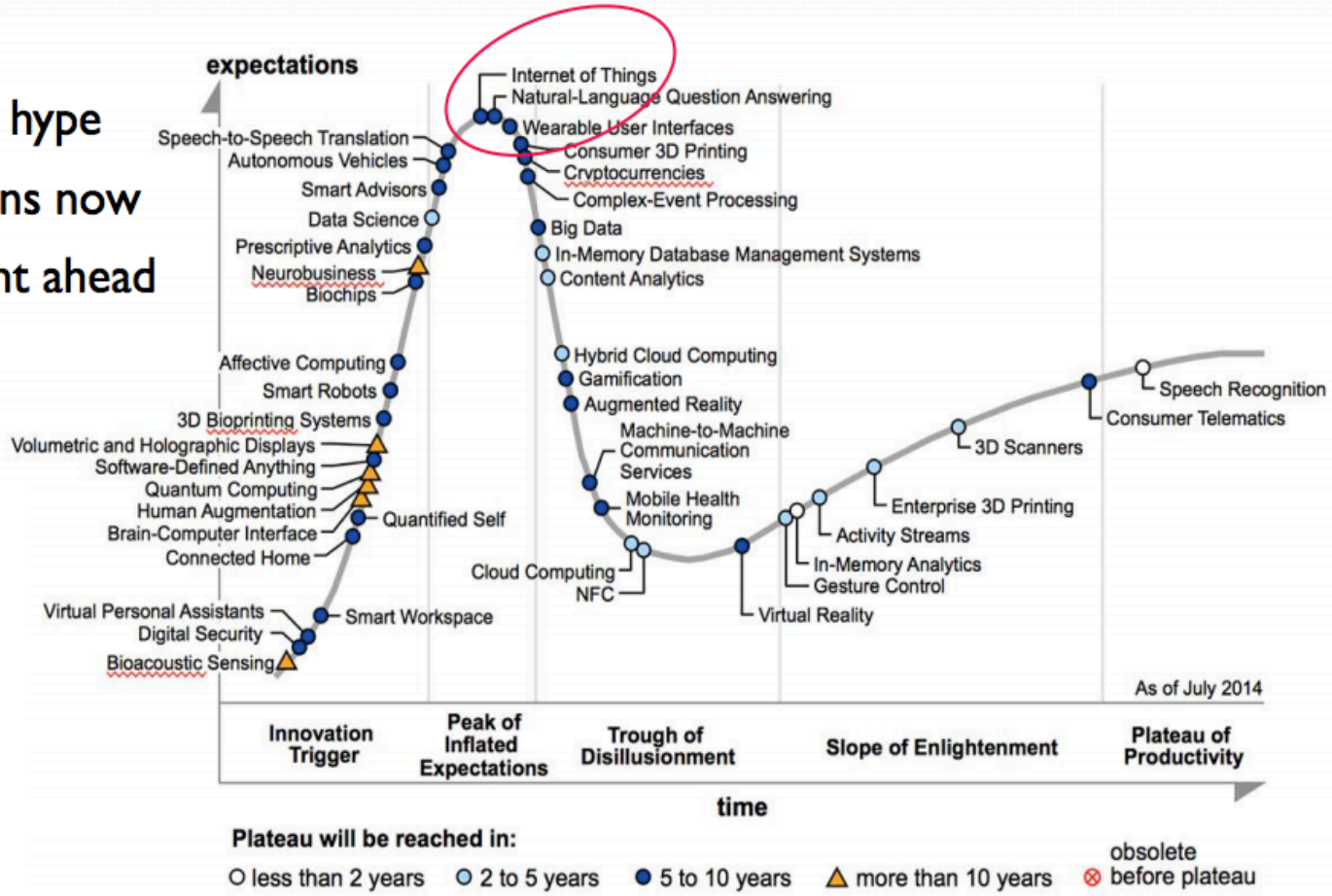
1. IoT Overview

IoTS – Internet of Things Intro

IoT and the Hype Cycle

- Gartner has IoT at peak hype
- Most inflated expectations now
- Trough of Disillusionment ahead
- 5-10 years to Plateau

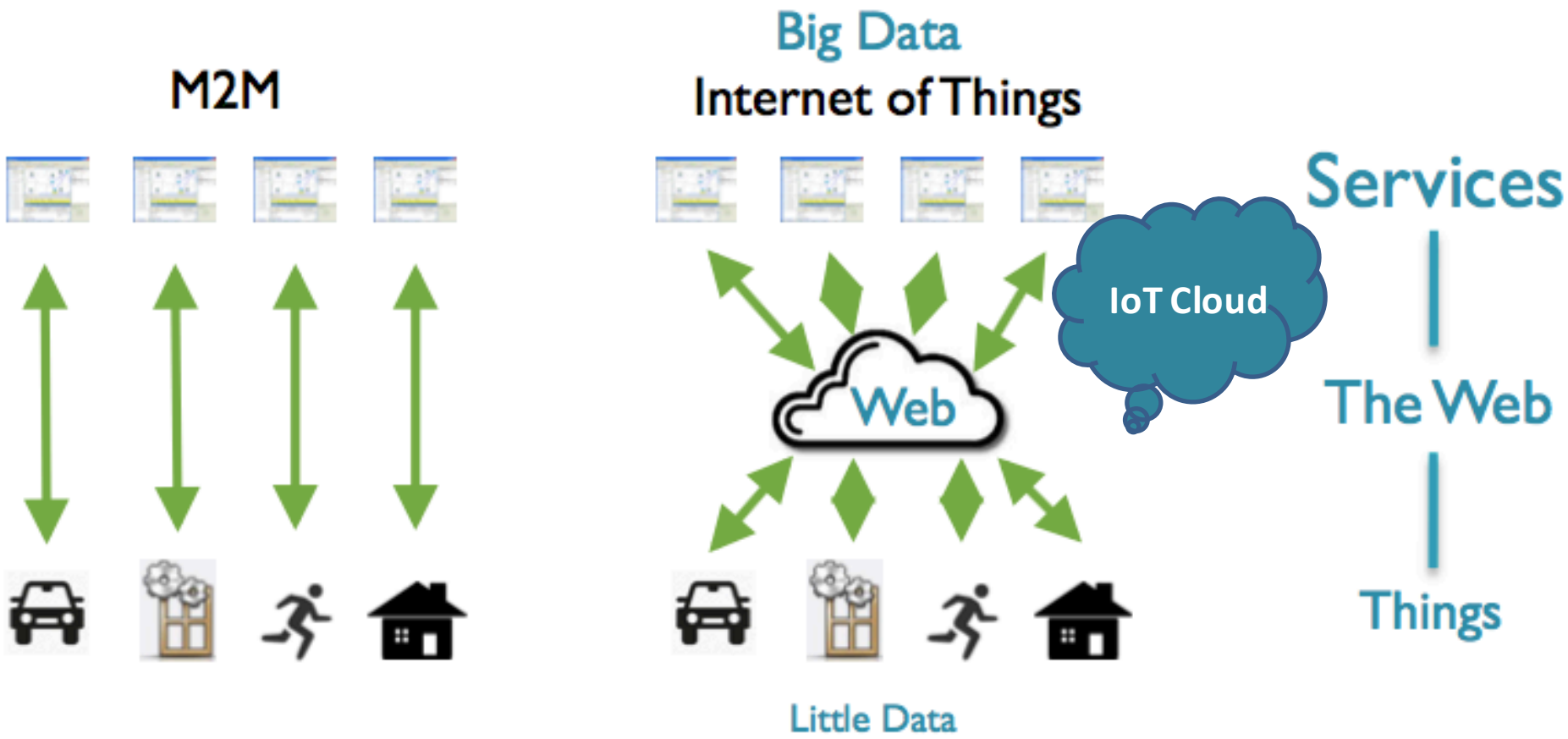




The Architecture for the Digital World® ARM

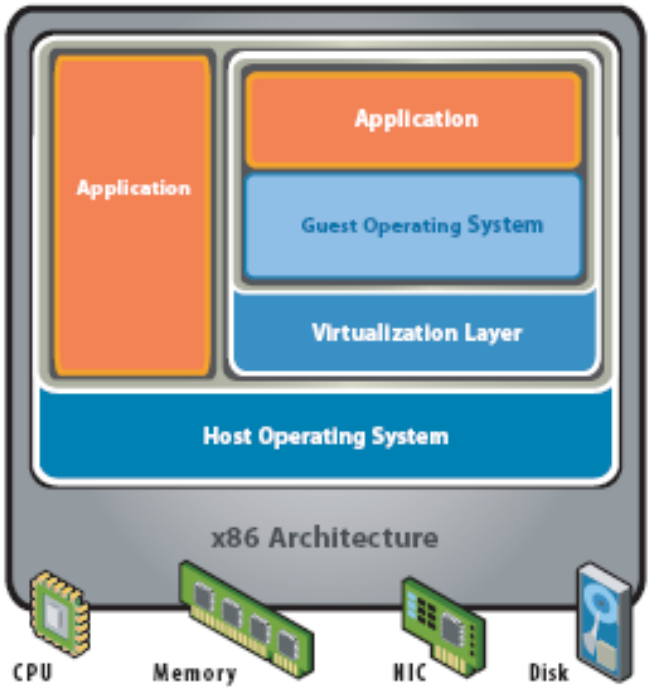
1. IoT Overview

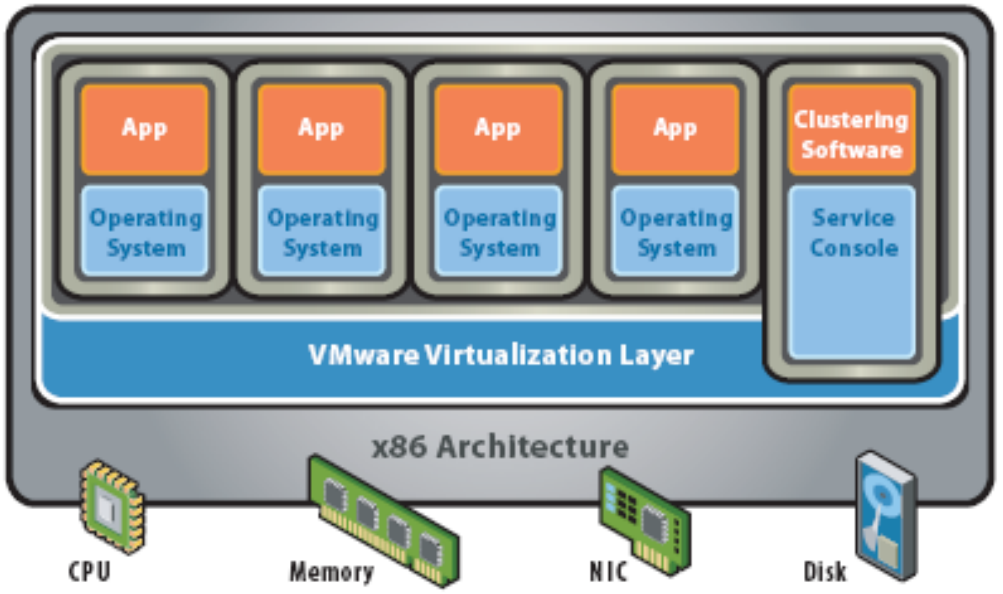
IoTS – Internet of Things Evolution from M2M to IoT



Copyright: https://community.arm.com/servlet/JiveServlet/downloadBody/8633-102-2-15471/ARM%20CoAP%20Tutorial%20April%2030%202014.pdf

RECAP: Cloud Concepts – Intro – Virtualization





Hosted Architecture

- Installs and runs as an application
- Relies on host OS for device support and physical resource management

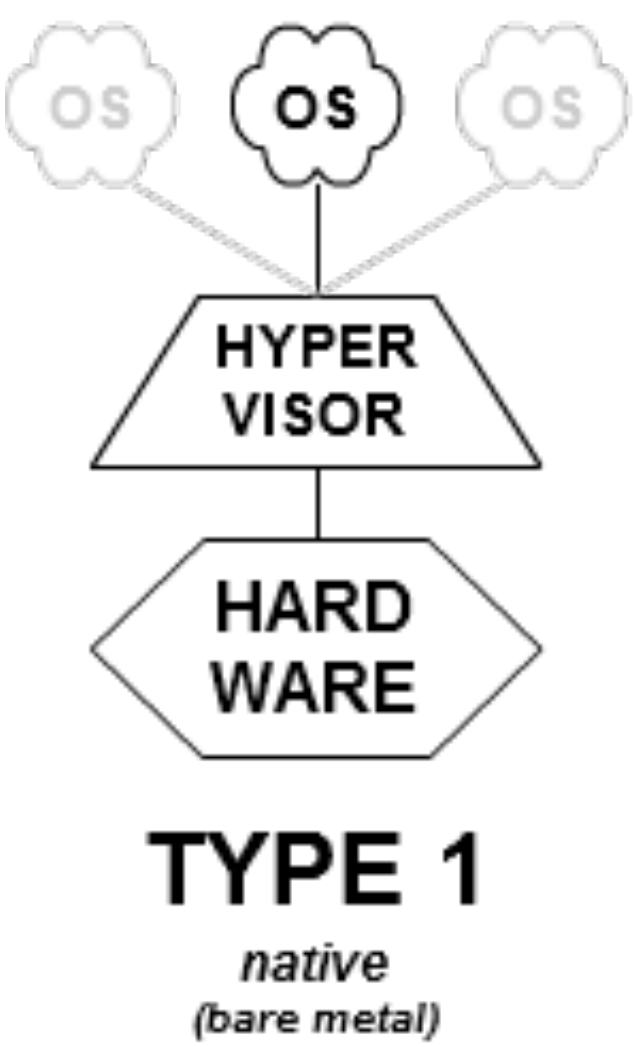
Bare-Metal (Hypervisor) Architecture

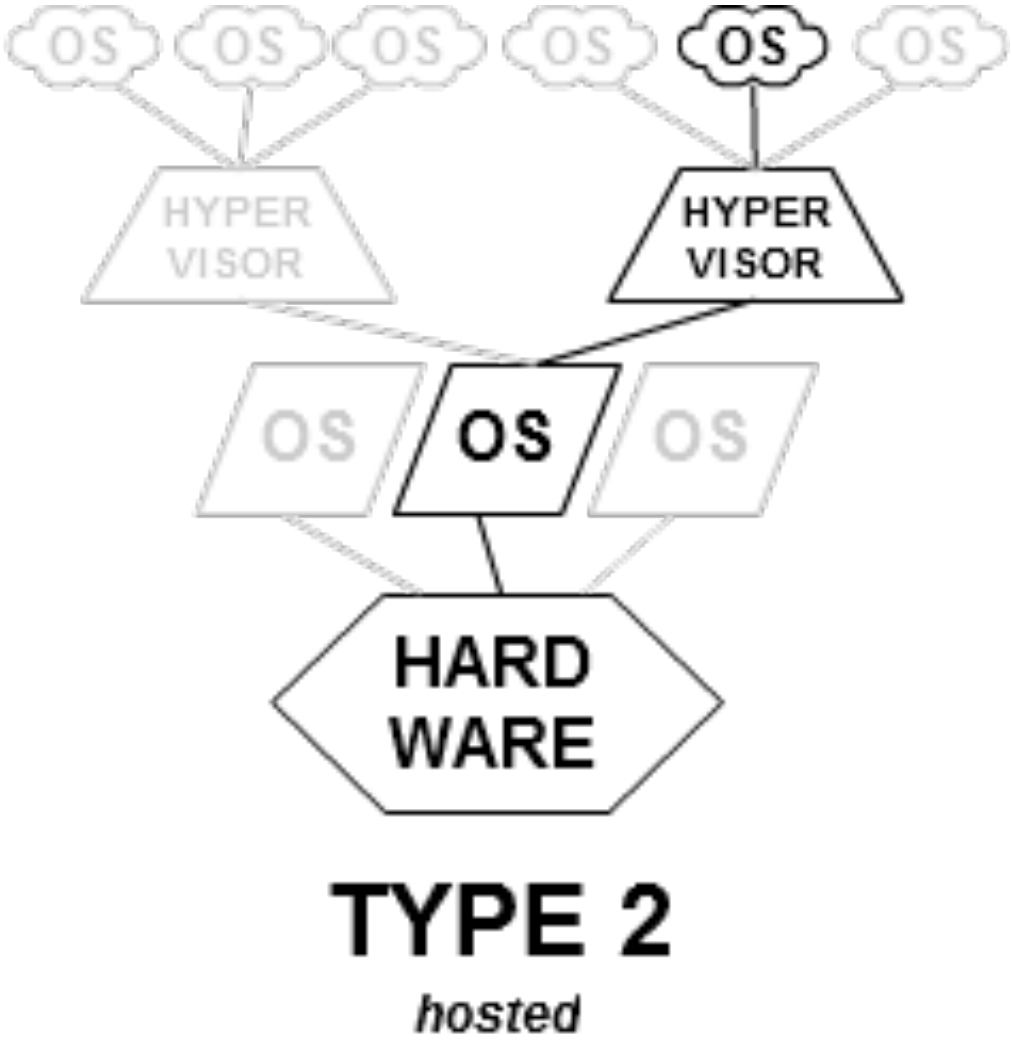
- Lean virtualization-centric kernel
- Service Console for agents and helper applications

Figure 2: Virtualization Architectures

Copyright to *www.vmware.com*

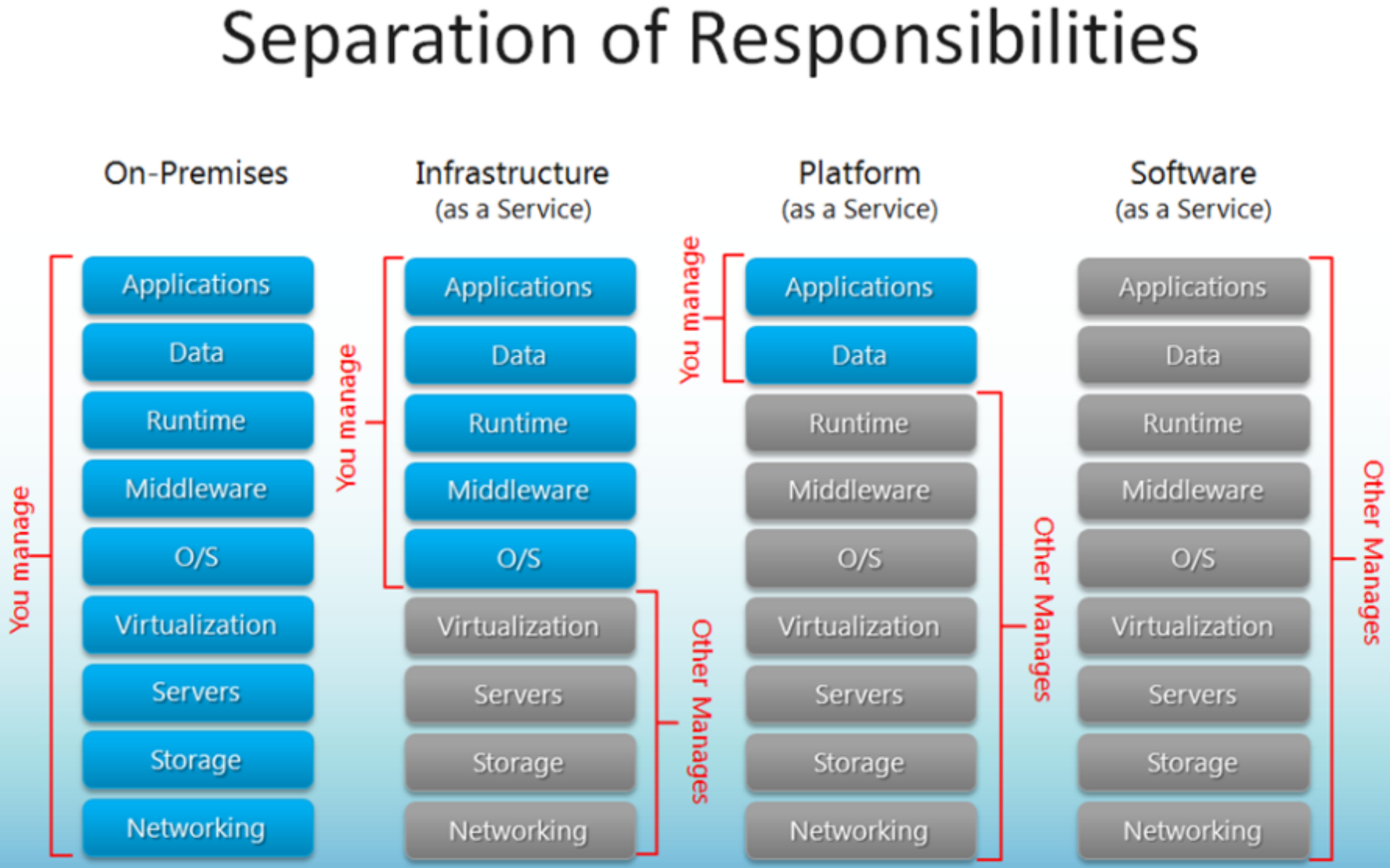
RECAP: Cloud Concepts – Intro – Virtualization Overview





Copyright to *www.wikipedia.org*

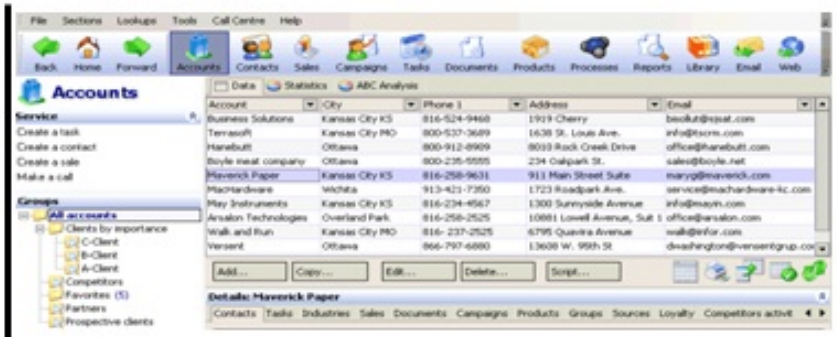
RECAP: Cloud Concepts – Intro – IaaS, PaaS, SaaS



Copyright to ***http://blogs.technet.com/b/kevinremde/archive/2011/04/03/saas-paas-and-iaas-oh-my-quot-cloudy-april-quot-part-3.aspx***

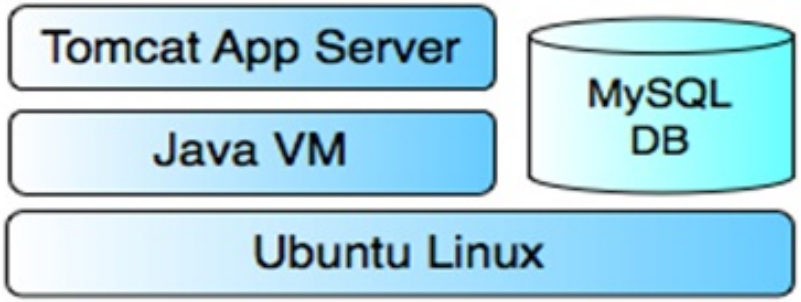
RECAP: Cloud Concepts – Intro – IaaS, PaaS, SaaS

SaaS



SalesForce.com, Google Apps

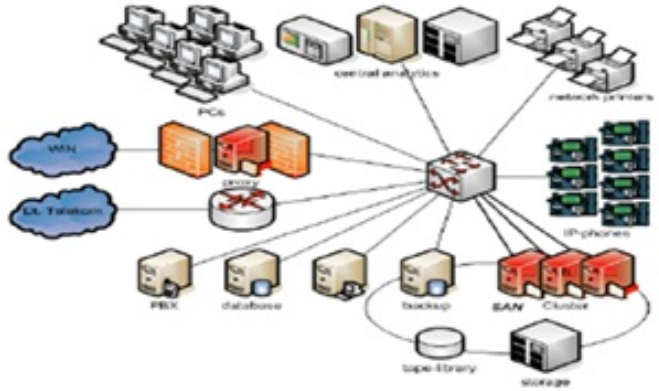
PaaS



Google App Engine for: Java, Ruby, Python & GO

VMForce.com, MS Azure

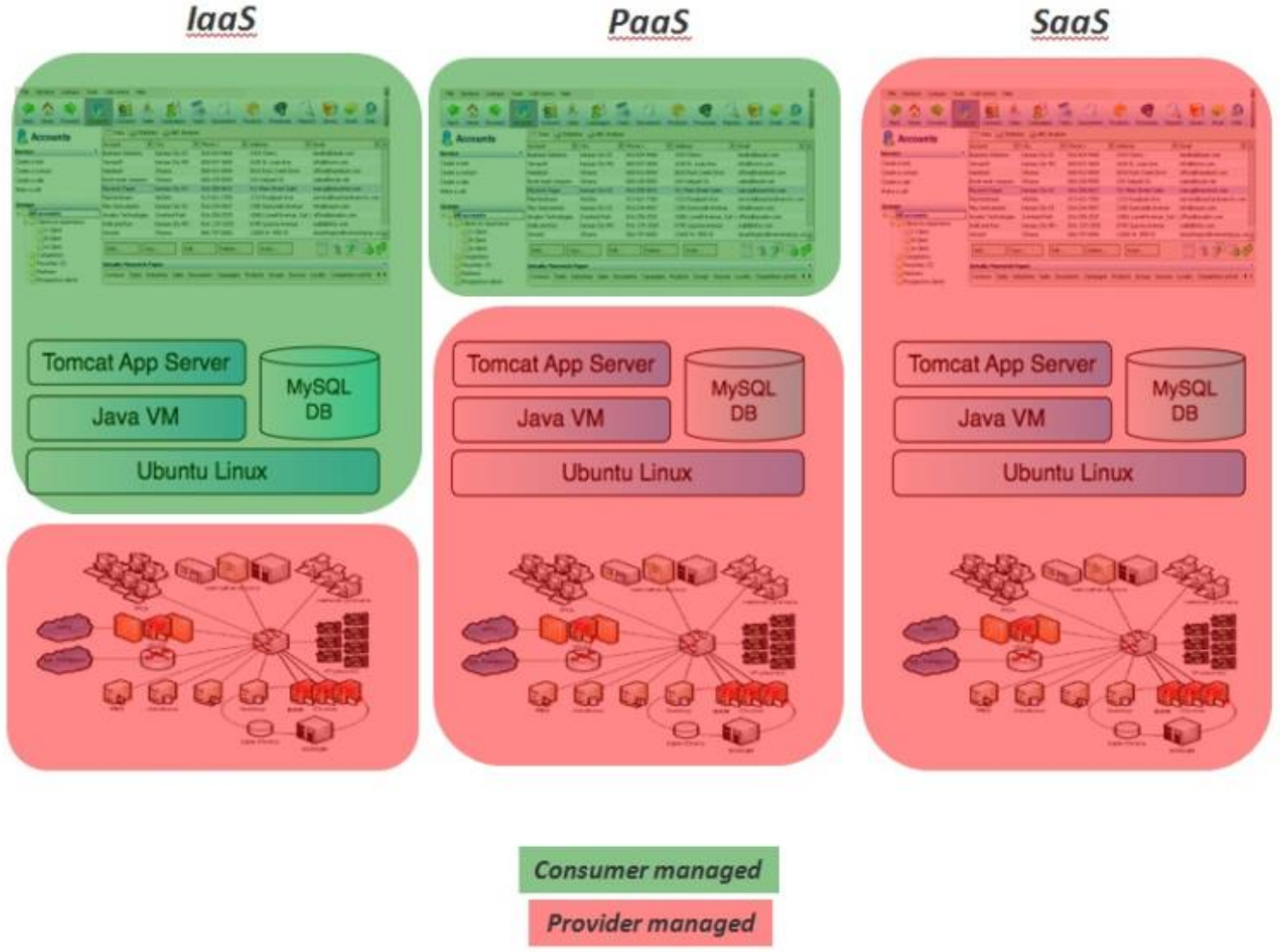
IaaS



vCloud Express/Datacenter, Amazon EC2

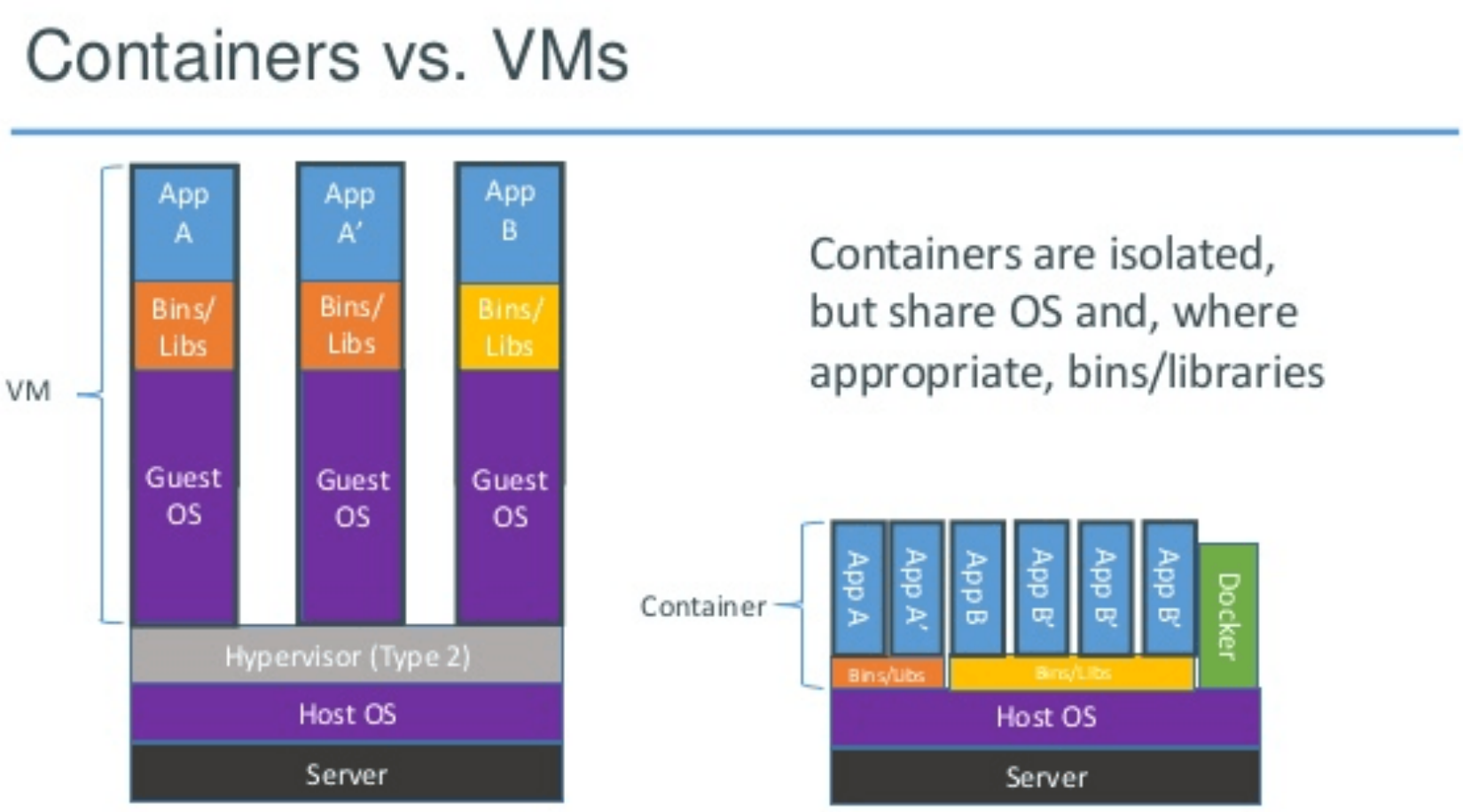
Copyright to http://it20.info/2010/11/random-thoughts-and-blasphemies-around-iaas-paas-saas-and-the-cloud-contract/

RECAP: Cloud Concepts – Intro – IaaS, PaaS, SaaS



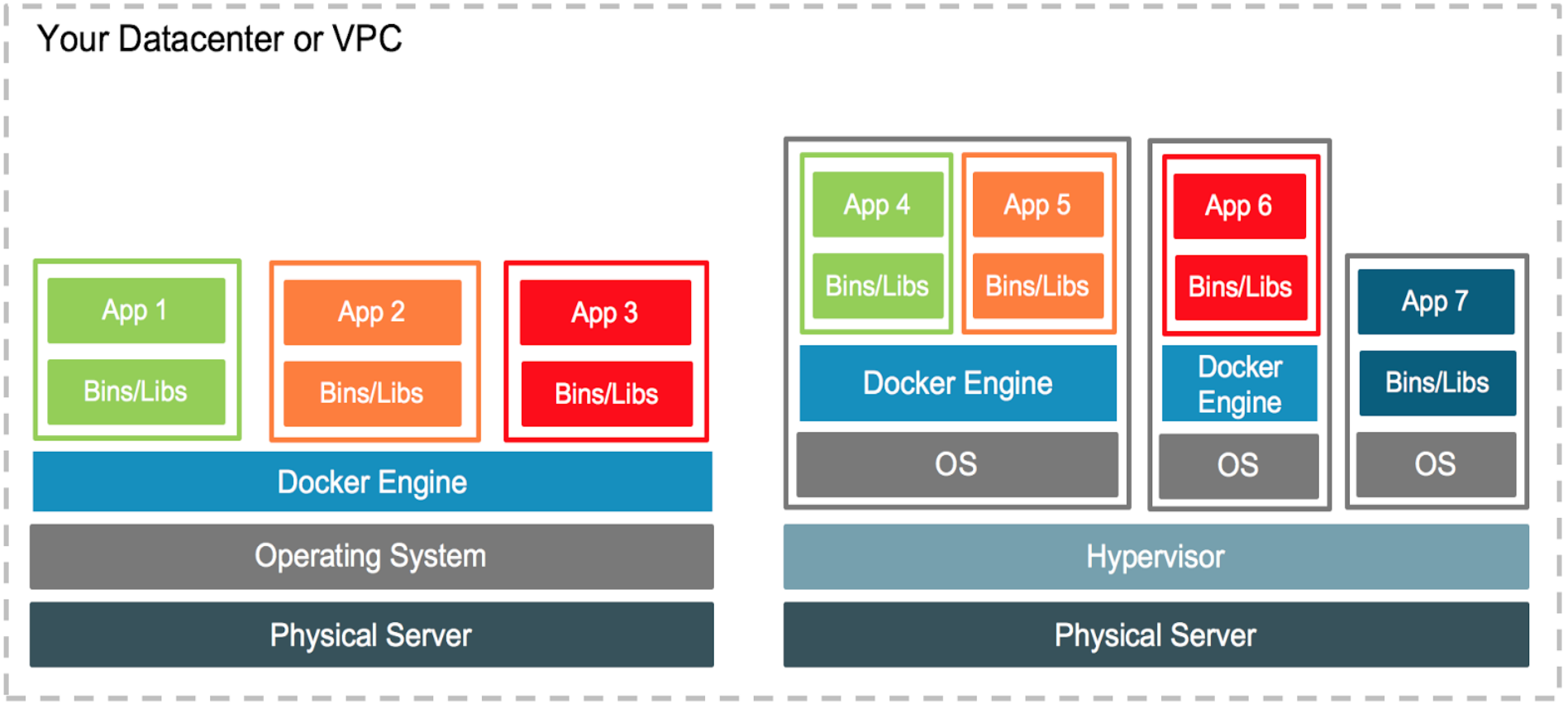
Copyright to
http://it20.info/2010/11/random-thoughts-and-blasphemies-around-iaas-paas-saas-and-the-cloud-contract/

RECAP: Cloud vs. Micro-Services/Containers Concepts



Copyright to http://www.rightscale.com/blog/sites/default/files/docker-containers-vms.png

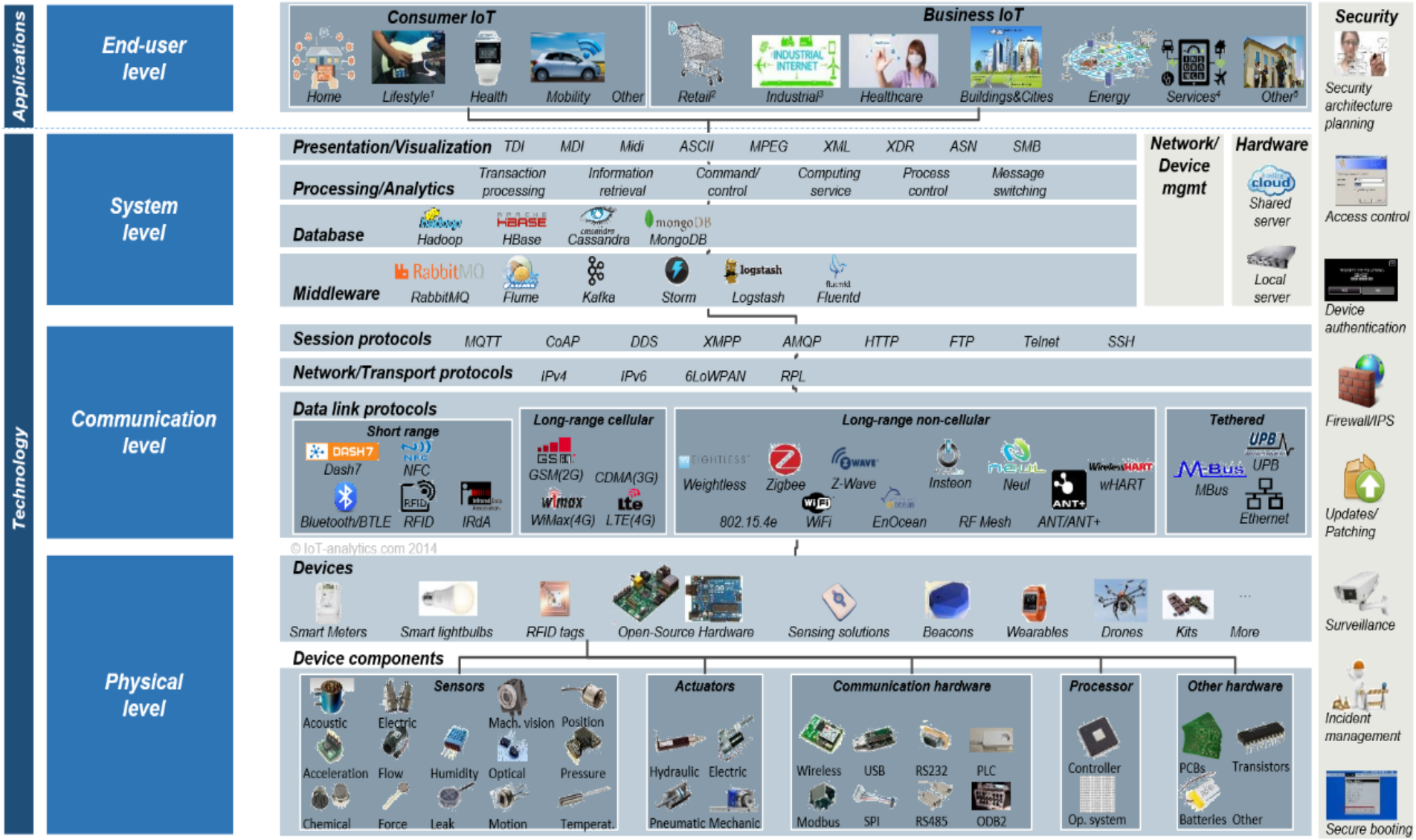
RECAP: Cloud vs. Micro-Services/Containers Concepts



Copyright to http://img.scoop.it/tImVj_1Pbqv0HJDyMWTmBbnTzqrqzN7Y9aBZTaXoQ8Q=

1. Embedded OS & IoT Architecture + Security

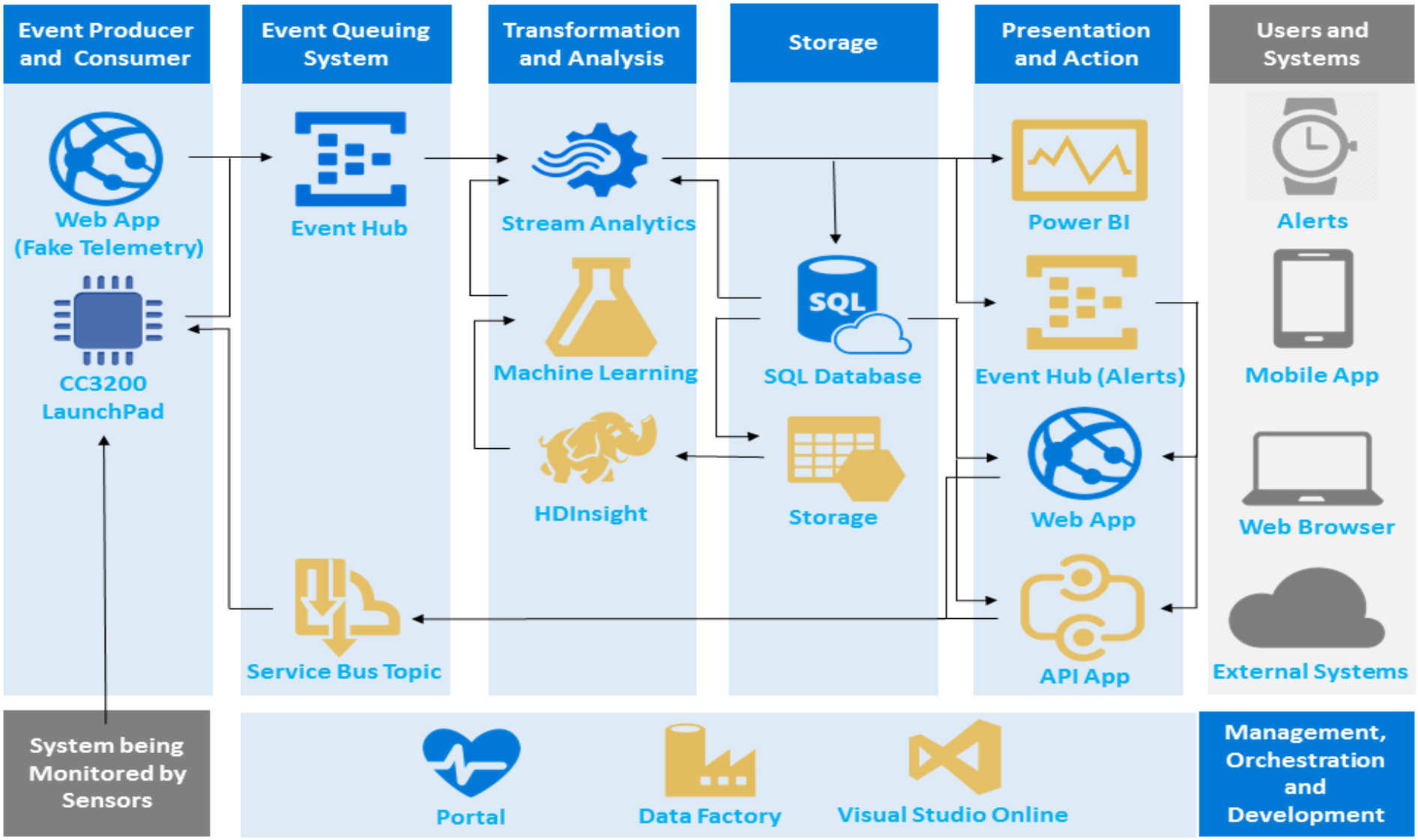
Iot Cloud Service Architecture + Security



Copyright: https://www.quora.com/Where-can-I-find-the-best-description-of-the-IoT-architecture-and-development-flow

1. Embedded OS & IoT Architecture + Security

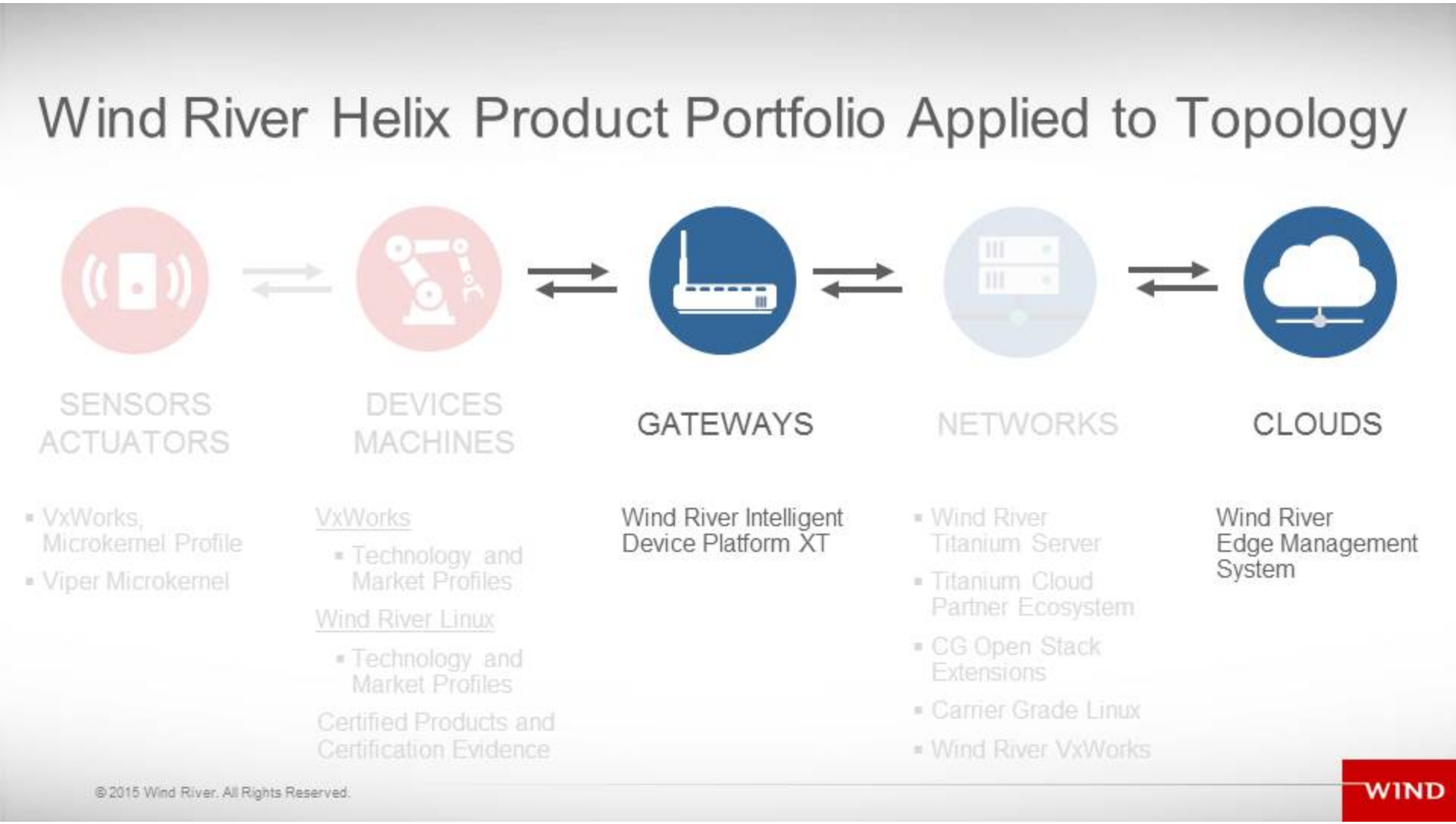
Embedded Hardware & OS within IoT Architecture + Security



Copyright: http://homearchi.xyz/iot-architecture/delightful-decoration-iot-architecture-on-architecture-ideas/

1. Embedded OS & IoT Architecture + Security

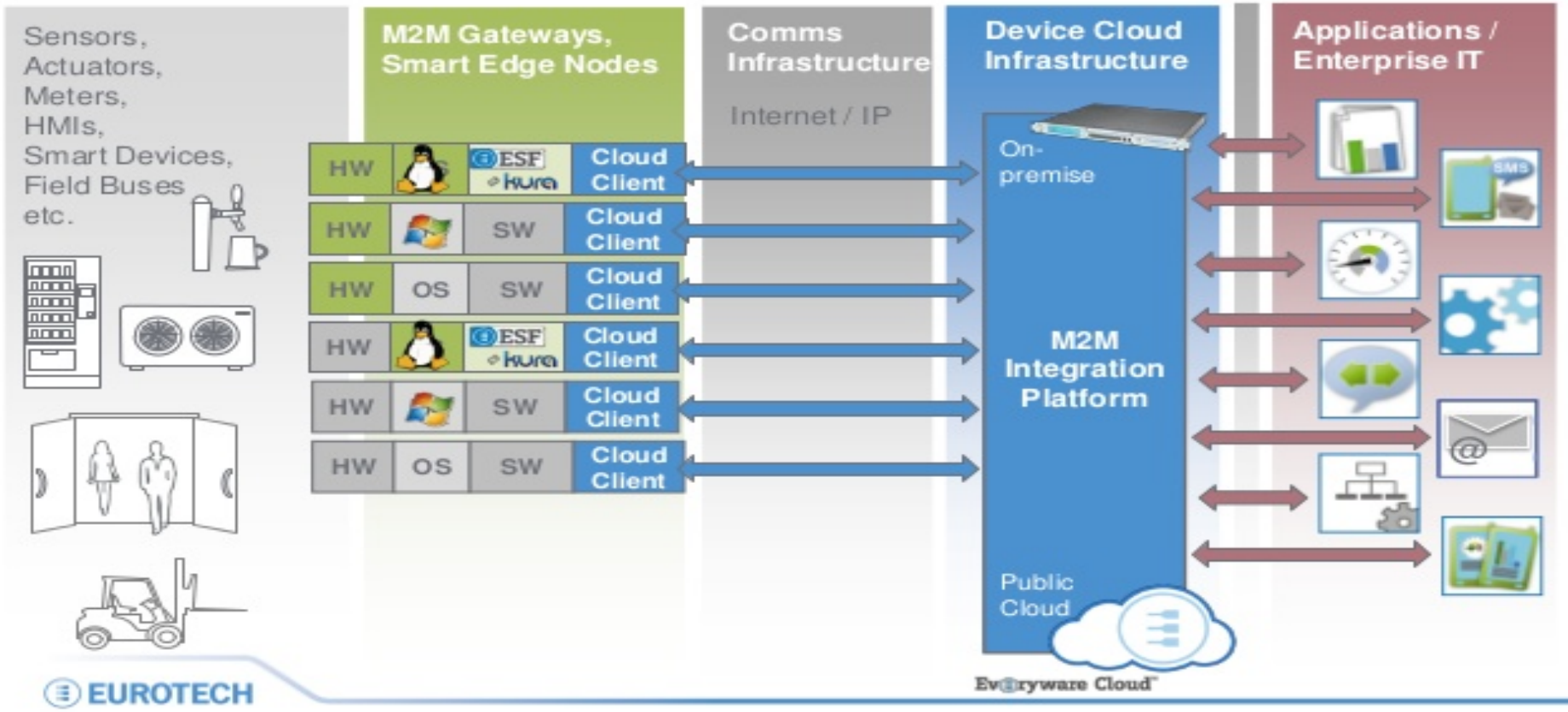
Iot Cloud Service Architecture + Security – Intel WindRiver



Copyright: http://blogs.windriver.com/wp-content/uploads/2015/02/Group-Nire-Helix-image.jpg

1. Embedded OS & IoT Architecture + Security

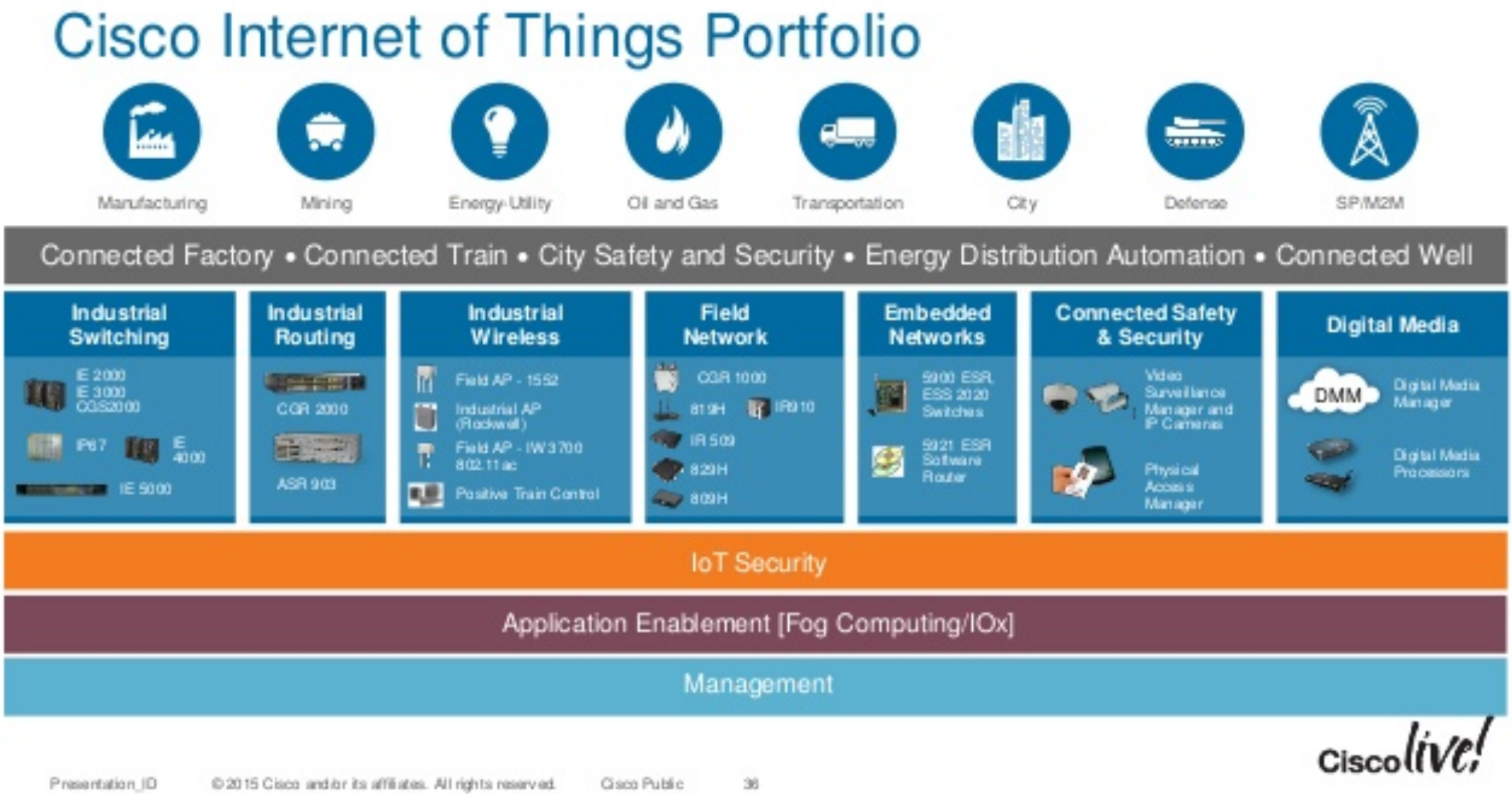
Iot Cloud Service Architecture + Security – Eurotech



Copyright: http://www.slideshare.net/Eurotechchannel/eth-encapsulating-complexityiotsolutionsiotwflondon

1. Embedded OS & IoT Architecture + Security

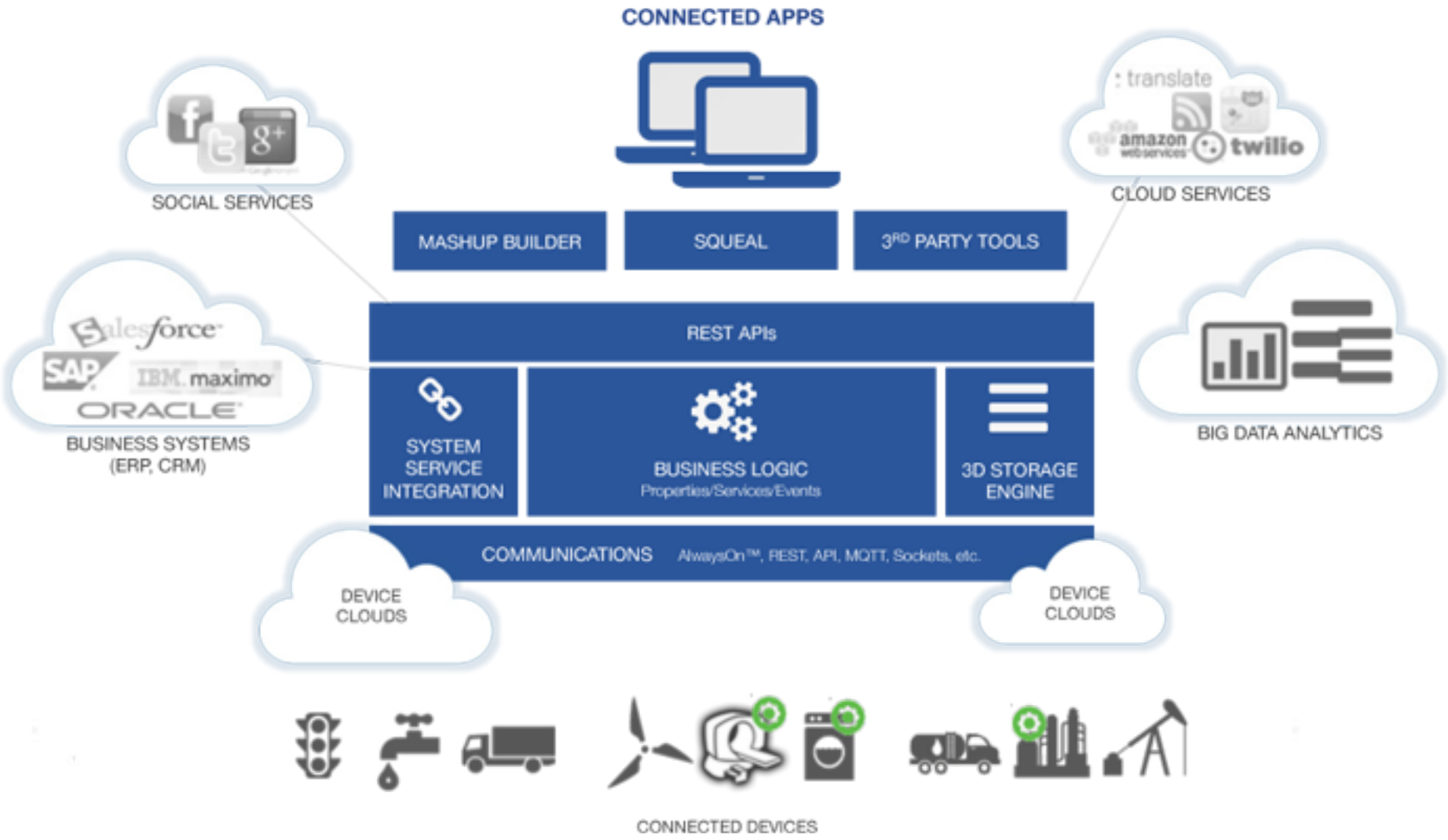
Iot Cloud Service Architecture + Security – Cisco



Copyright: http://www.slideshare.net/Cisco/enabling-the-internet-of-everything-ciscos-iot-architecture

1. Embedded OS & IoT Architecture + Security

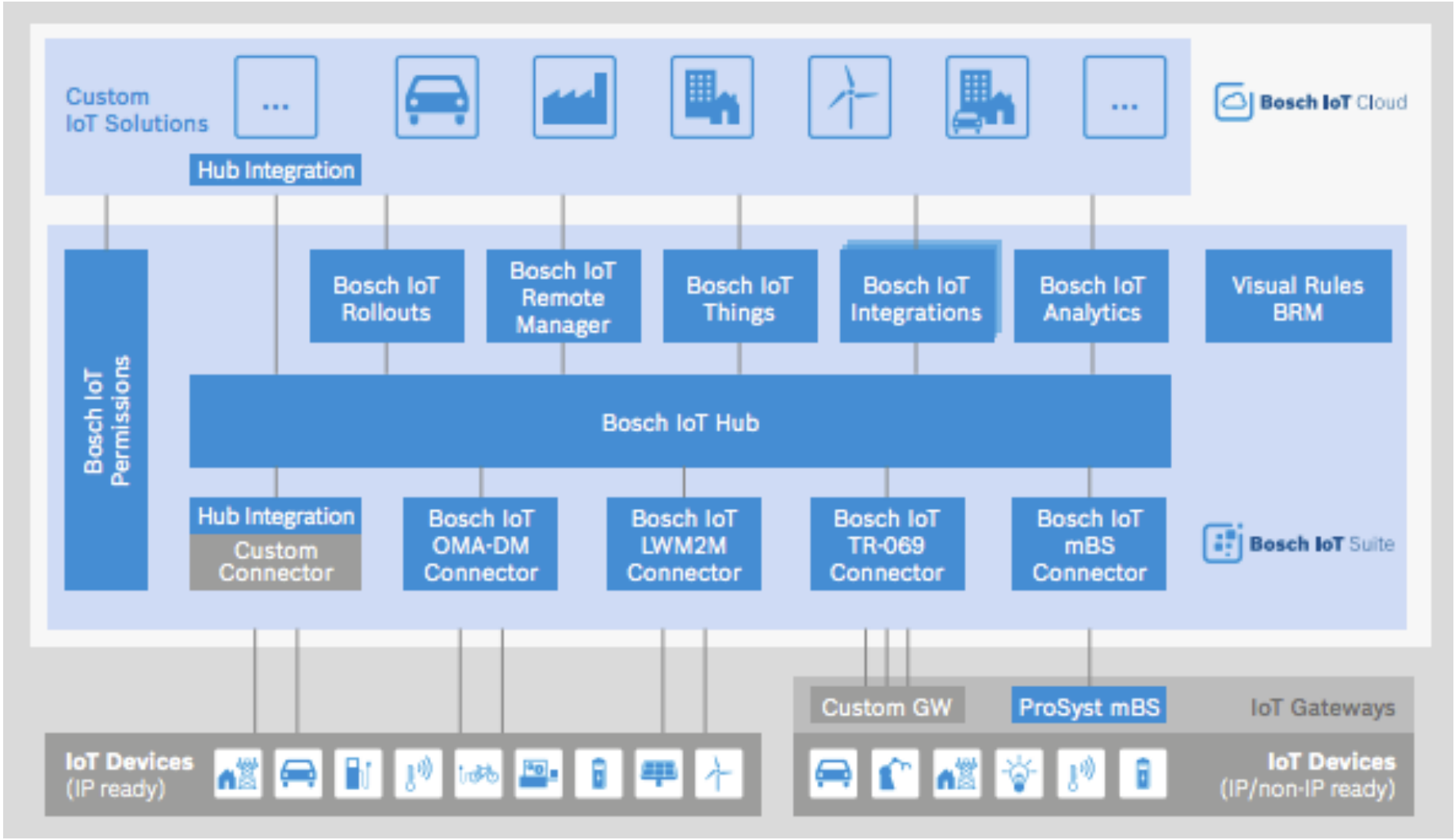
Iot Cloud Service Architecture + Security – **Salesforce/c9.io - ThingsWorx**



Copyright: https://deniscanty.files.wordpress.com/2015/04/imag.png

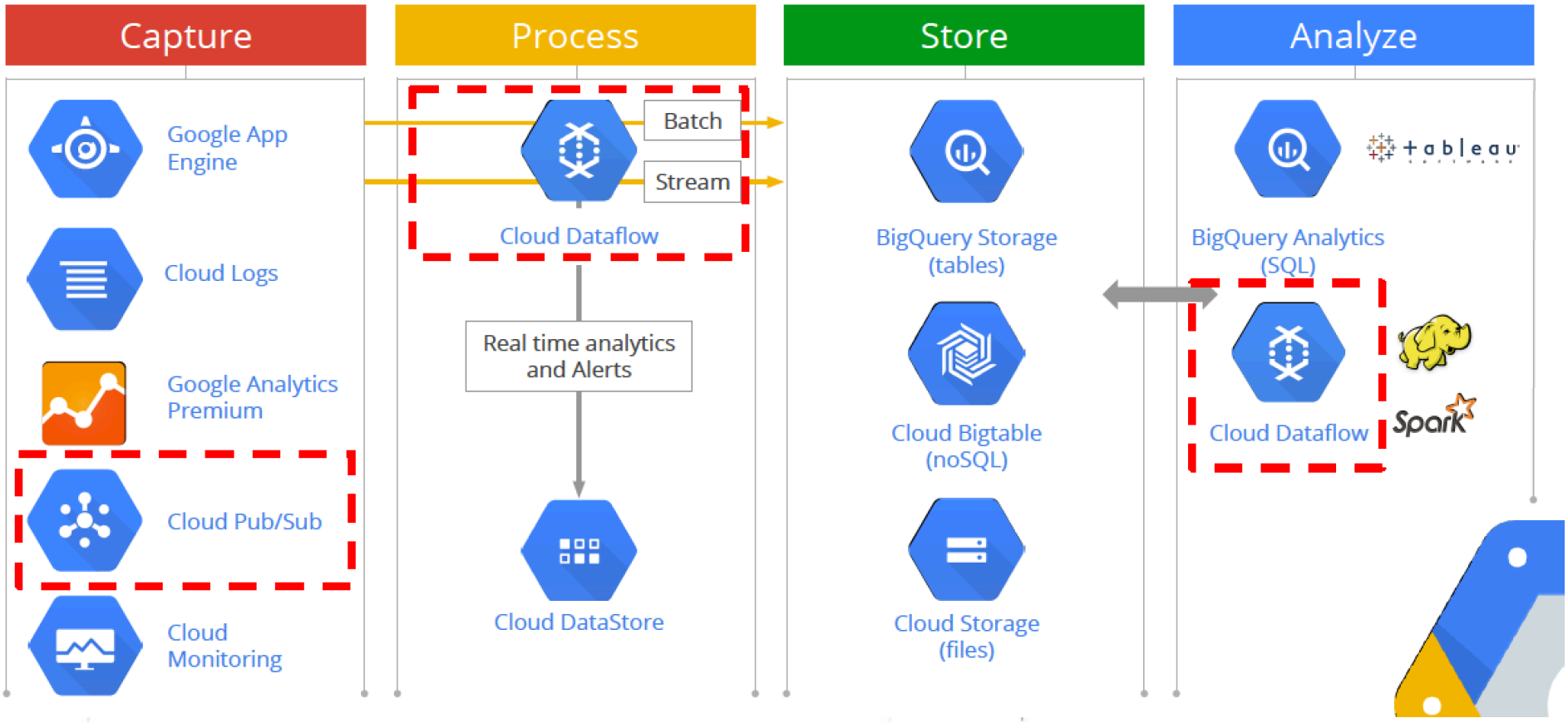
1. Embedded OS & IoT Architecture + Security

Iot Cloud Service Architecture + Security – Bosch



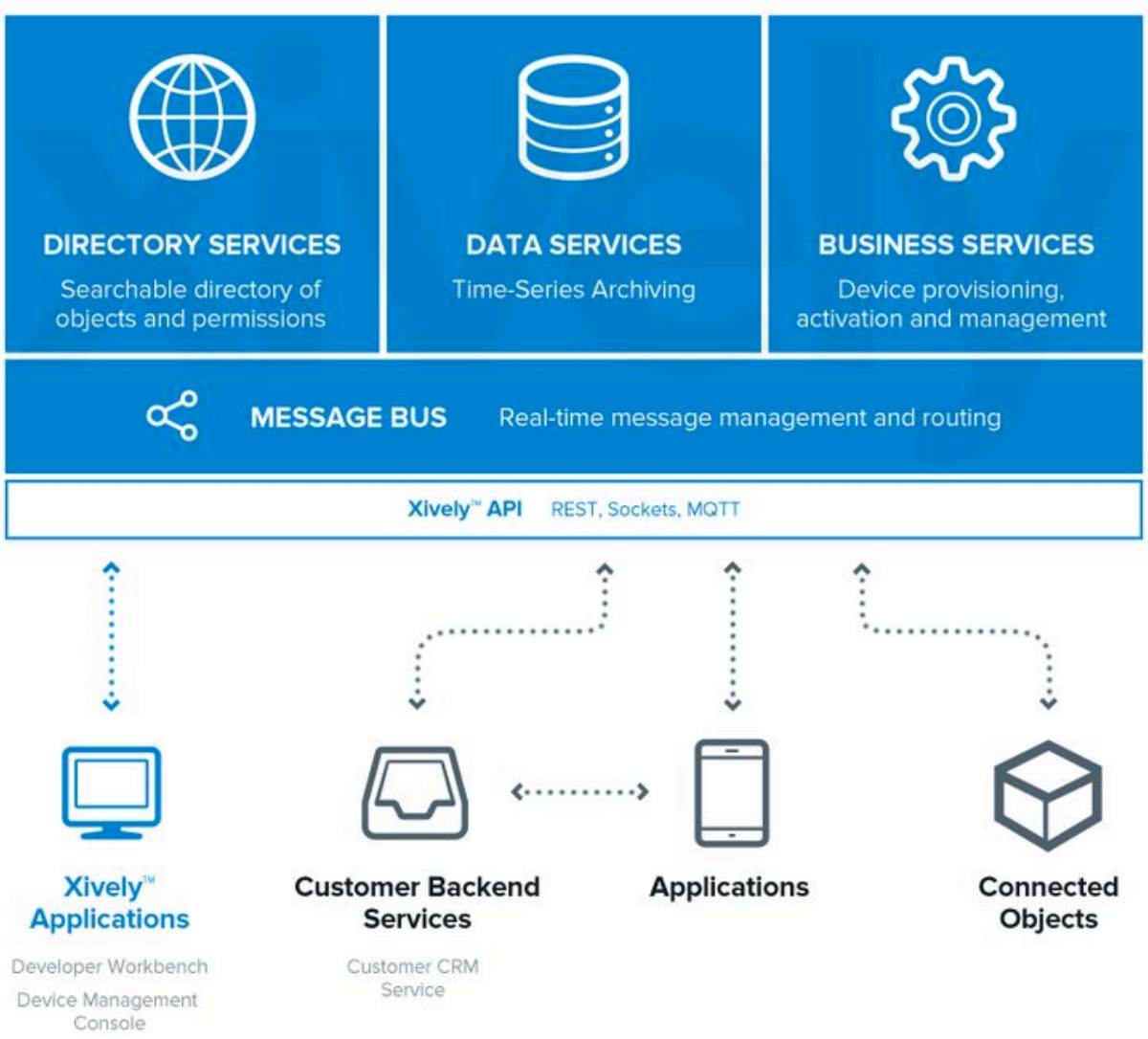
1. Embedded OS & IoT Architecture + Security

Iot Cloud Service Architecture + Security – Google



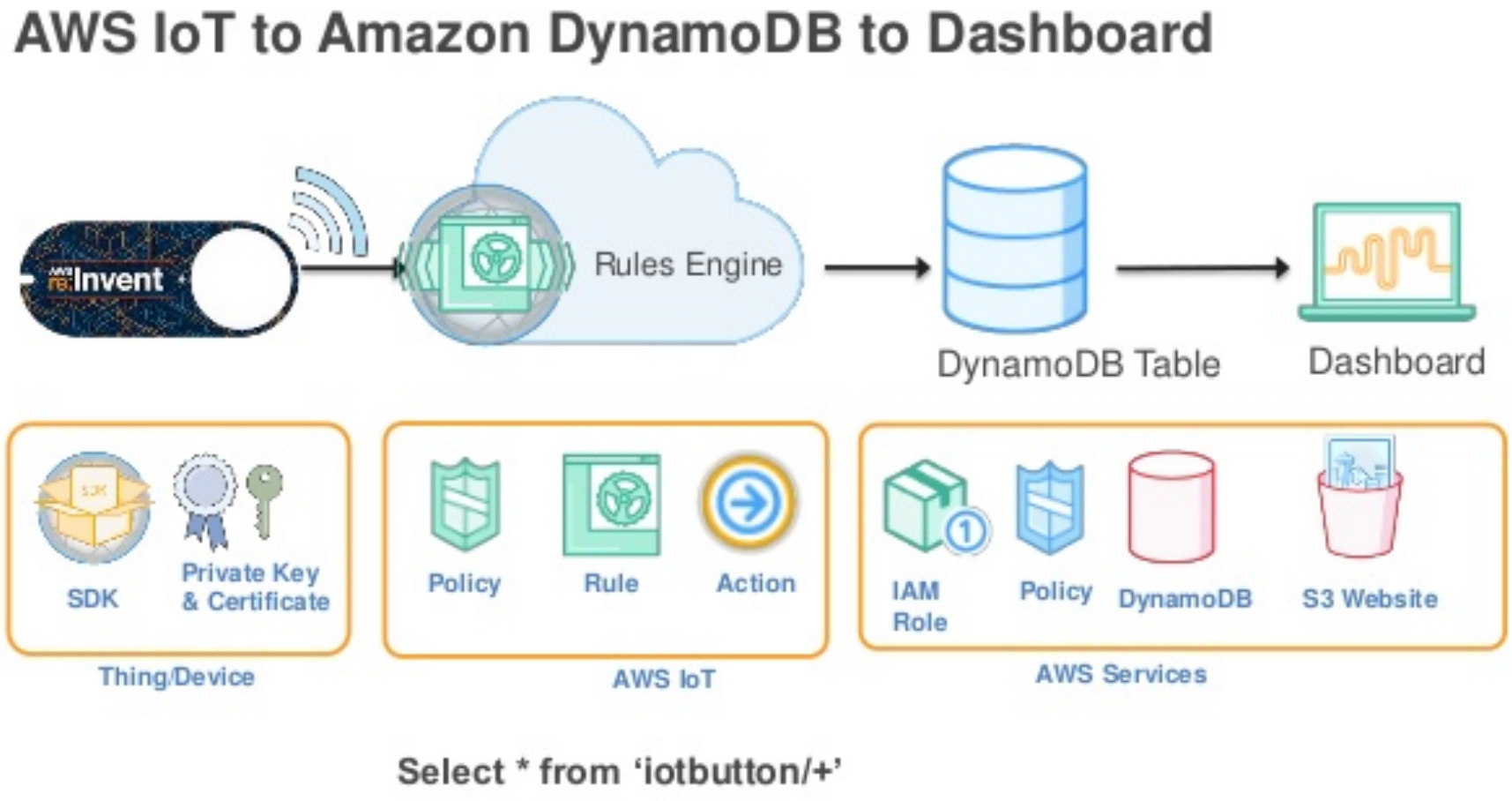
1. Embedded OS & IoT Architecture + Security

Iot Cloud Service Architecture + Security – Xively



1. Amazon IoT Architecture

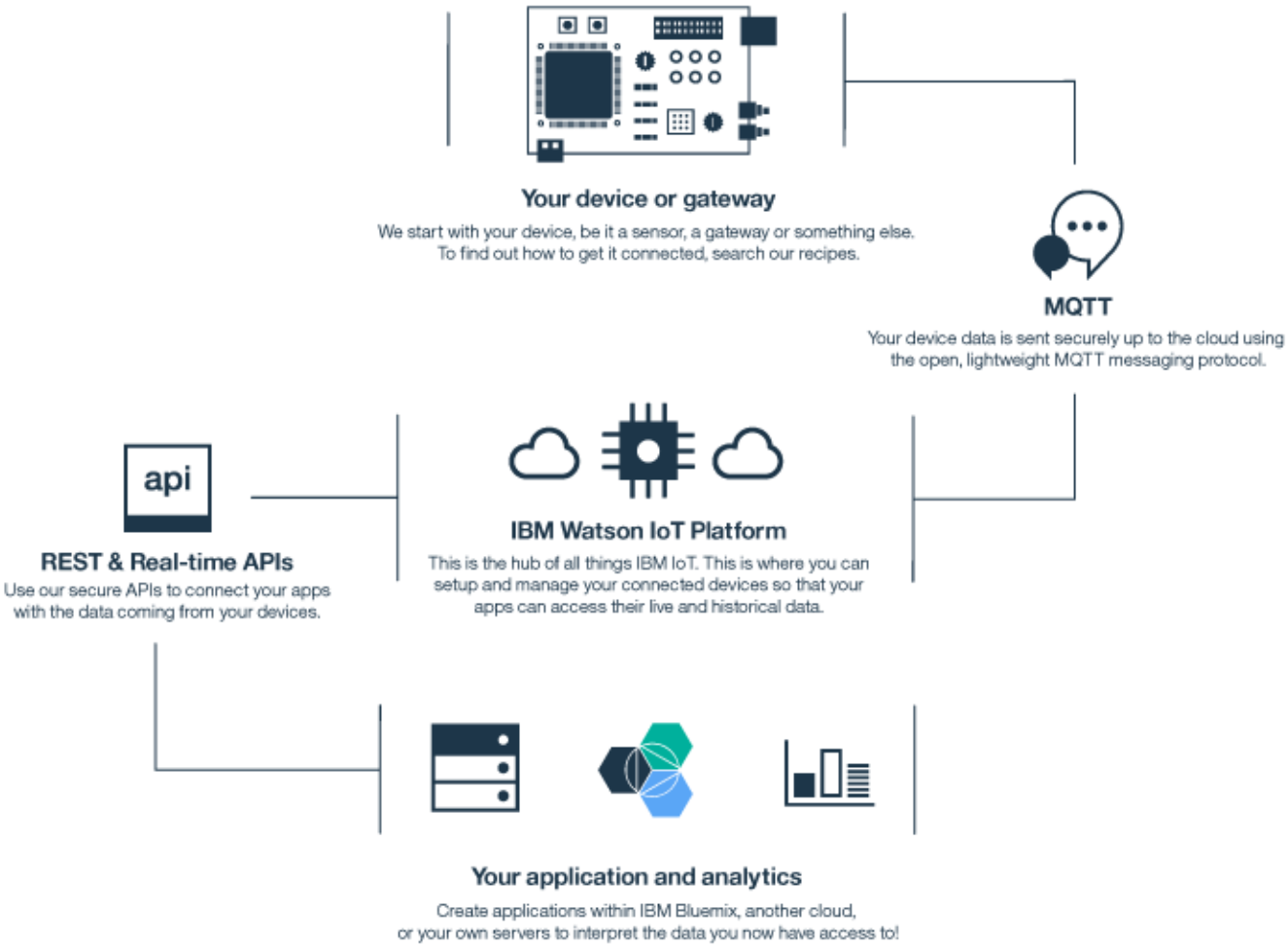
Iot Cloud Service Architecture + Security – **Amazon AWS IoT Cloud**



Copyright:http://www.slideshare.net/AmazonWebServices/mbl205-new-everything-you-want-to-know-about-aws-iot

1. IBM IoT Architecture

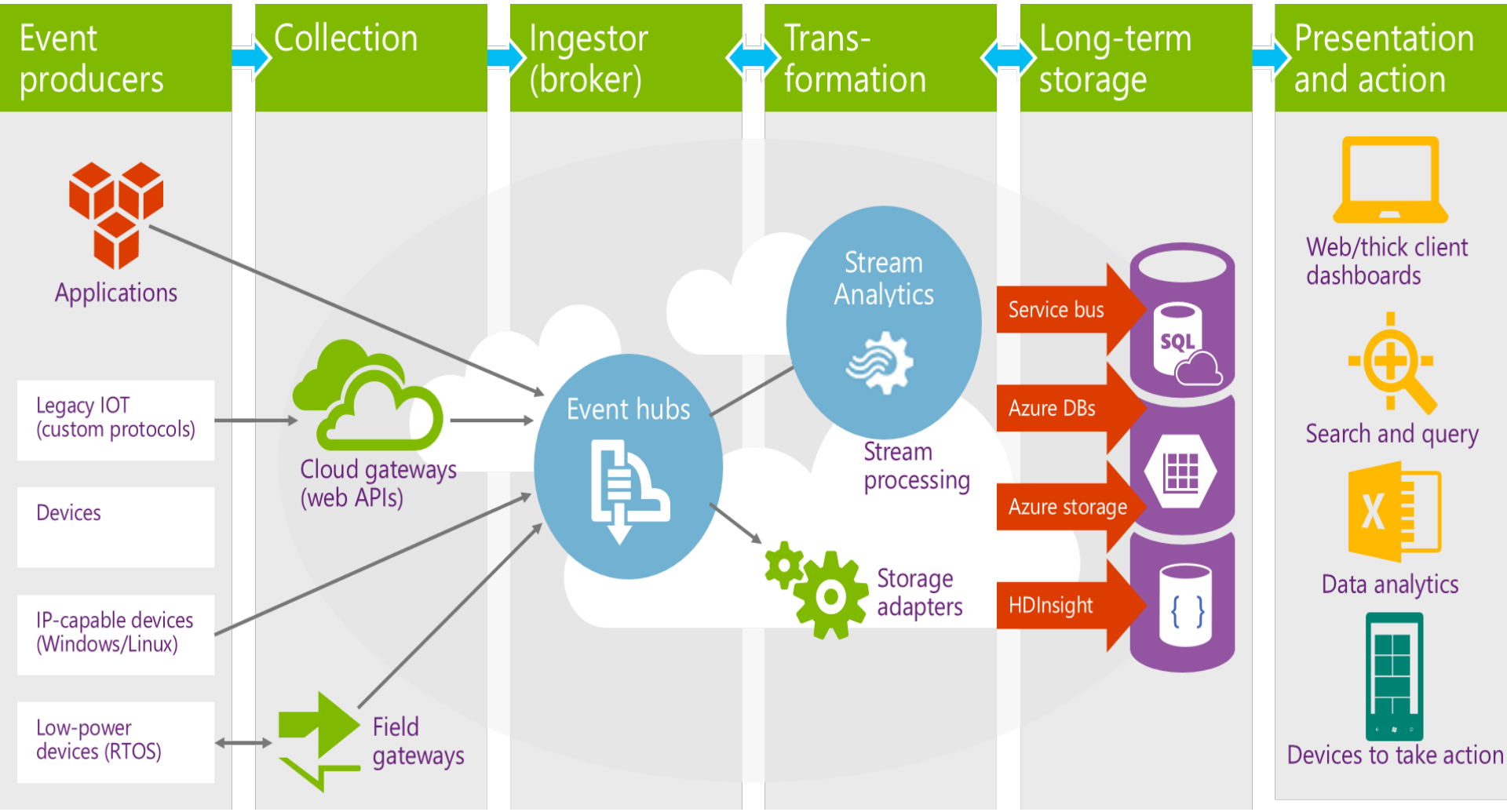
Iot Cloud Service Architecture + Security – IBM Watson IoT Platform



Copyright: https://console.ng.bluemix.net/catalog/services/internet-of-things-platform

1. Microsoft IoT Architecture

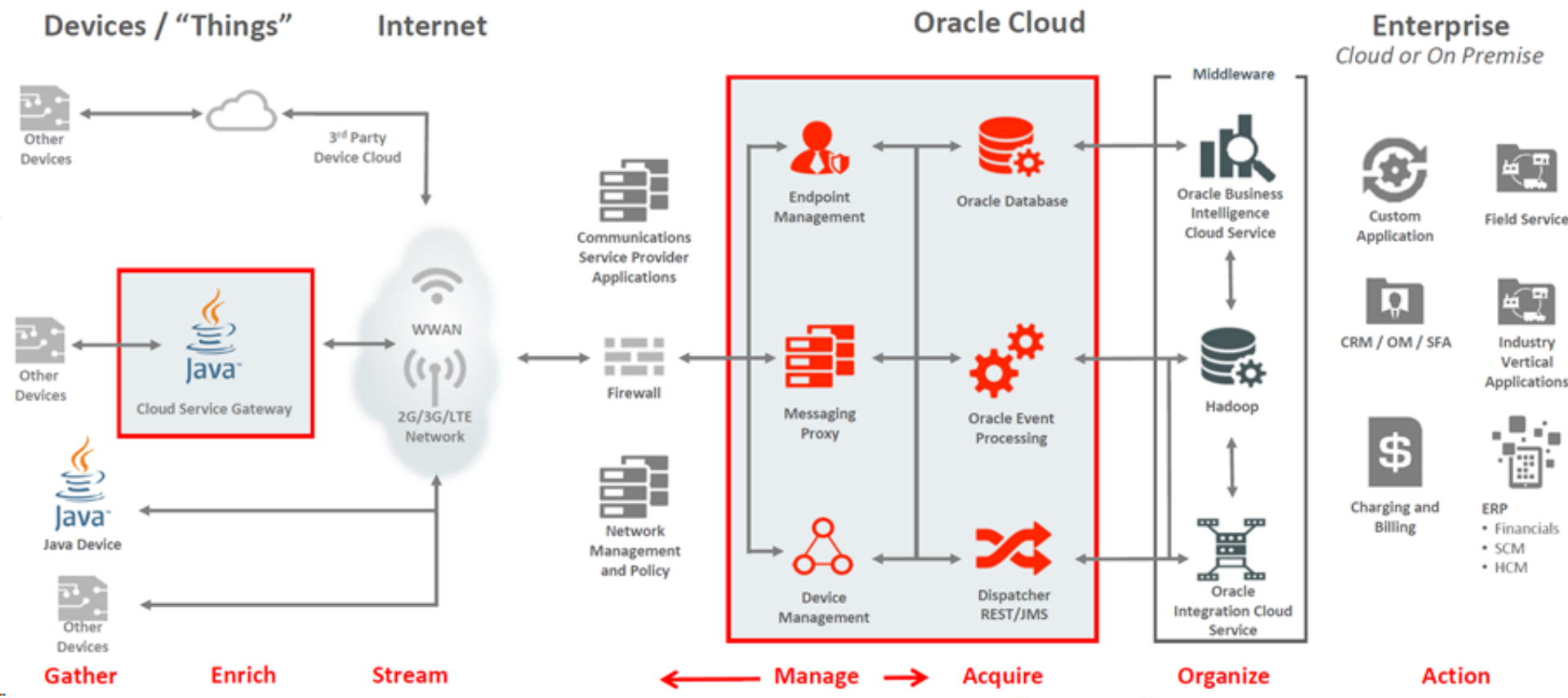
Iot Cloud Service Architecture + Security – **Microsoft Azure IoT**



Copyright: https://blogs.msdn.microsoft.com/petsablog/2014/12/11/iot-for-mere-mortals-part-i/

1. Oracle IoT Architecture

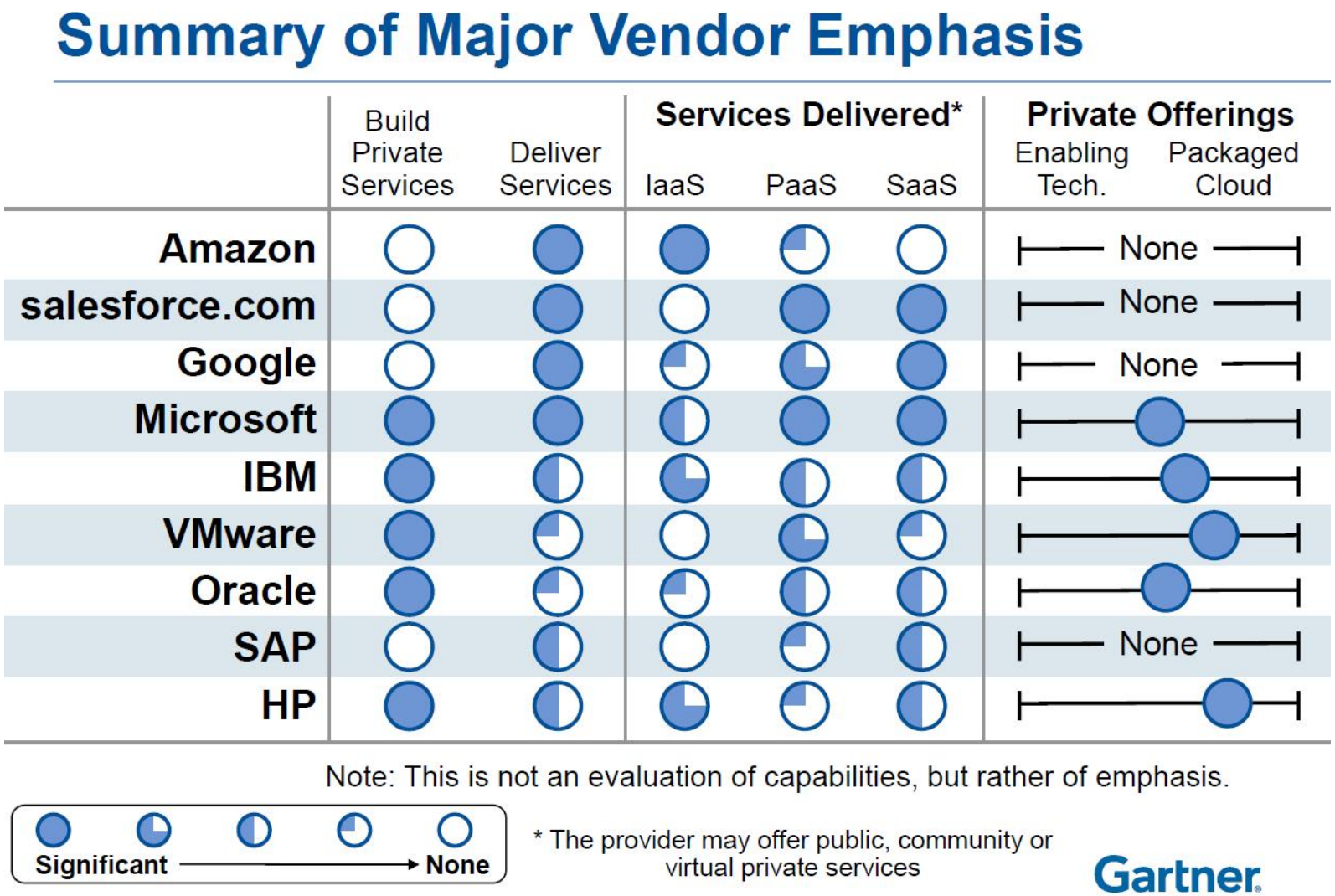
Iot Cloud Service Architecture + Security – **Oracle IoT Cloud Service**



Copyright: https://thecattlecrew.files.wordpress.com/2015/11/oracle_iot_cloud_architecture.png

1. Cloud Vendor Emphasis Comparison (Y 2013)

Gartner Cloud Comparison (Y 2013)



1. IoT Overview

IoTS – Internet of Things Intro

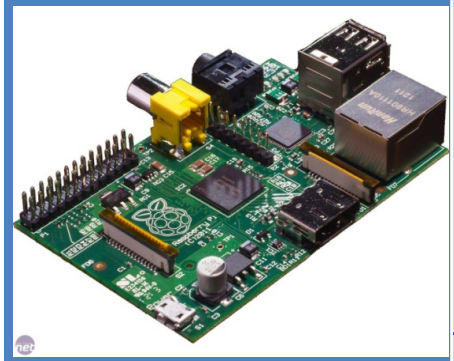
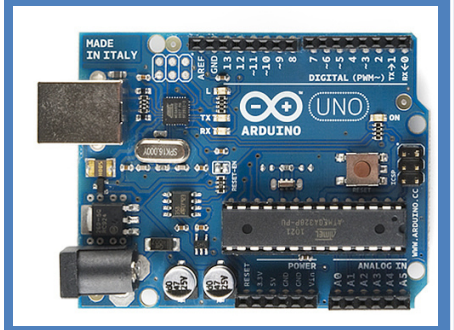
IoT - Internet of Things

Applications: Smart Cities (e.g. Waste Collection Management), Distributed Computing in IoT Cloud/Siloses, Semantic Sensor Network, Health Care App, E-Payment Solutions, Social Sensing Applications, Big Data / NoSQL processing.

IoT Middleware (M2M - Machine to Machine / Internet Protocols)	IoT Sensors & Actuators	Embedded Devices - Smart Objects	IoT Cloud / Back-End Systems
REST & WS-SOA Web Semantic (Web 2.0 & 3.0) - OWL/RDF	Temperature, Humidity, Motion, Camera, Polution - CO, Noise, Infrared – Actuators: Engines, Plugs, Boilers, etc.	Embedded IoT GWs and Nodes PoC: Rasberry PI, Arduino, BeagleBone /Ninja blocks Production: Eurotech, Cisco/Tehnicolor, HMS-Netbiter, etc.	Oracle IoT CS IBM BlueMix Microsoft Azure Amazon AWS IoT Public / Private PaaS or IaaS Clouds (ThingsWorks, Xively)
CoAP	Radio - ISO 14443 A/B (prox. card/tag) NFC / ISO 15693 & 18000 vicinity card/tag	Smart Objects API & Device Models/Types	Open Source Clouds / GRID / Distributed and Parallel Systems
MQTT M2M – MoM Protocols, Agents based Middleware		Gateway Services	

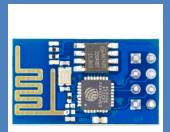
1. IoT Boards

IoTS – Internet of Things Smart Objects: IoT Nodes and Gateways (Dev Boards – Non-production)

Smart Object	Tools + Developers Web Page
Raspberry PI 	Micro-processor / Micro-controller: ARM | Linux based OS Programming Languages / Dev Platforms: Python, C/C++, ARM ASM or Java Embedded Web (UK): http://www.raspberrypi.org/
Arduino 	Micro-processor / Micro-controller: ATMega Programming Languages / Dev Platforms: Arduino Programming Language (based on Wiring), Atmel ASM, C/C++ Web (Italy): http://www.arduino.cc/

1. IoT Boards

IoTS – Internet of Things Smart Objects: IoT Nodes and Gateways (Dev Boards – Non-production)

Smart Object	Tools + Developers Web Page
Waspmote 	Micro-processor / Micro-controller: ATMega Programming Languages / Dev Platforms: Waspmote Scripts (looks like Atmel C/C++ combined with Java) Web (Spain): http://www.libelium.com/development/
BeagleBone / Ninja Blocks 	Micro-processor / Micro-controller: ARM | Linux based OS: Ubuntu or Android 4.0 Programming Languages / Dev Platforms: C/C++, Java for Android, Ubuntu Programming Languages, ASM ARM Web (US): http://beagleboard.org/Products/BeagleBone
ESP-8266 ESP-01/ESP-12 / NodeMcu Lua ESP8266 CH340G WIFI 	Micro-processor / Micro-controller: ESP8266EX | HAL/OS: Native Firmware with AT commands or Lua interpreter capabilities Programming Languages / Dev Platforms: Lua or AT commands Web: https://benlo.com/esp8266/esp8266QuickStart.html

1. IoT Mobile Smart Devices as Gateways

IoTS – Internet of Things Smart Objects: Mobile Smart Devices (Dev & Production)

Mobile Convergence for M-App Development

HTML 5 / CSS 3 / JavaScript

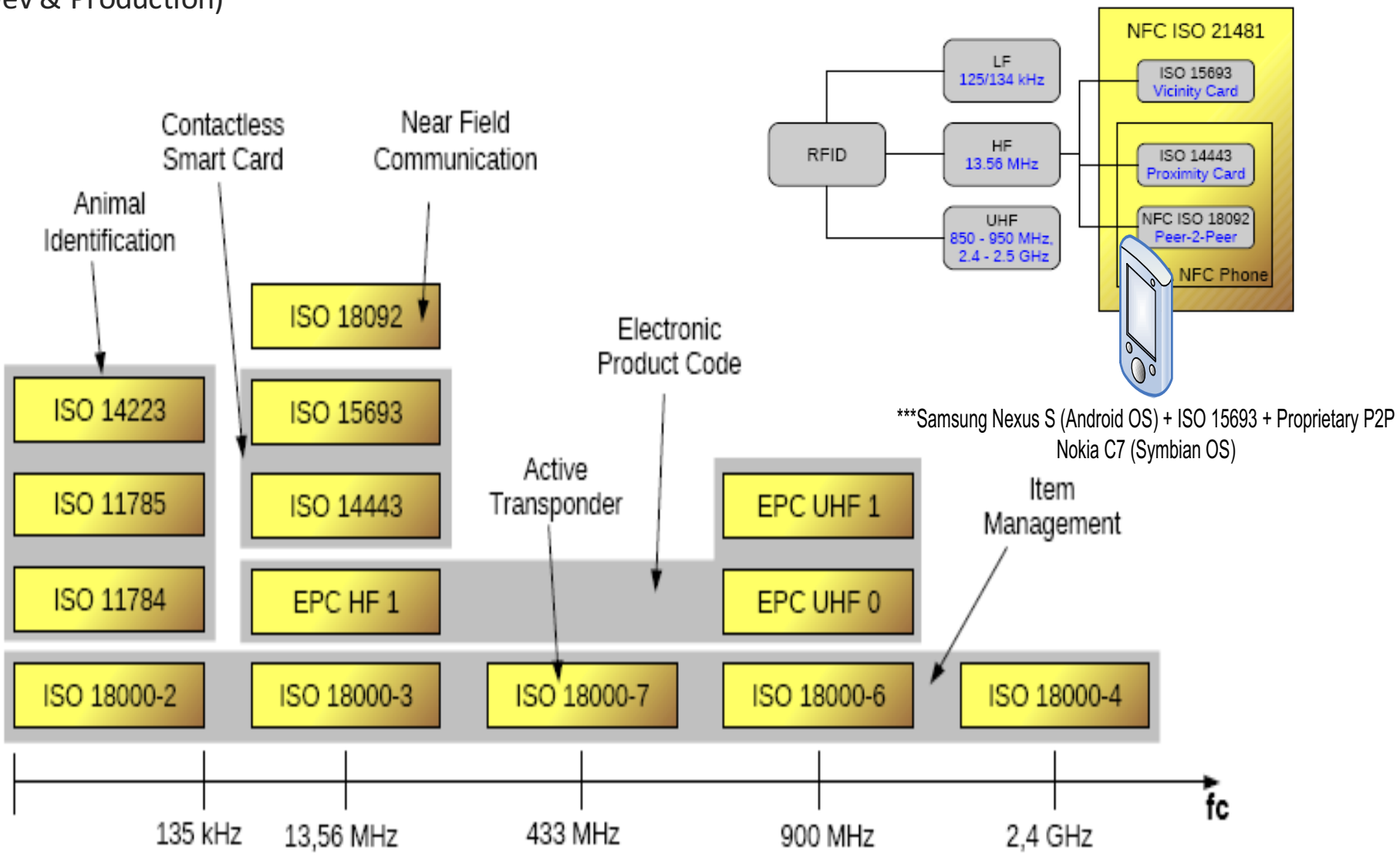
WebKIT Engine / Similar Engine

Mobile IoT: Android/Java, iOS, C-Posix, JavaScript

ANDROID	iOS Apple	BlackBerry	Windows 8/10 IoT Core	Intel Tizen	SailfishOS & Ubuntu Touch	Firefox OS
Java	C/C++	Swift / Objective-C	Java & Native C/C++	Web: HTML 5	C#.NET	C/C++	Web: HTML 5	Native: C++ Qt	Web: QML or HTML 5	Web: HML 5

1. IoT Mobile Vicinity and Proximity Techs

IoTS – Internet of Things Smart Objects: Mobile NFC | RFID Sensors & Standards – Vicinity vs. Proximity (Dev & Production)



Embedded OS HW Details (for IoT GW) – Real Devices vs. Development Boards

No	Vendor/Provider	Model	OS / VM	CPU	RAM (MB)	Flash/SD/SSD (GB)	Main Promoted Dev Platforms/SDKs
1	**Eurotech**						
1.1		DynaGATE 15-10	Yocto Linux, Intel Gateway Solutions for IoT ready; It may support Java SE	Intel Quark SoC X1020D, 400Mhz	512	8 GB	1. Everyware Software Framework (ESF) support to extend capabilities further, including remote device management and Java programmability 2. POSIX C 3. Java SE-e
1.2		ReliaGATE 20-11	Wind River Linux (intel); it may support Java SE	Intel ATOM, E620 (T) 600MHz, E640(T) 1GHz, E660(T) 1.3GHz, E680(T) 1.6GHz	512	8 GB	1. Everyware Software Framework (ESF) Ready compatible with Eurotech's Everyware Device Cloud (EDC) 2. POSIX C 3. Java SE-e
1.3		ReliaGATE 15-10	Yocto Linux, Intel Gateway Solutions for IoT ready; It may support Java SE	Intel Quark SoC X1020D, 400Mhz	512	4 GB	1. Everyware Software Framework (ESF) support to extend capabilities further, including remote device management and Java programmability 2. POSIX C 3. Java SE-e

Embedded OS HW Details

2	**IBM**	it supports including Raspberry Pi and Arduino					
2.1		Intel IoT GW D200 (D50/100/200)	Wind River Linux (Intel) - IoT Ready (including MQTT distro); it may support Java SE	Intel Quark SoC X1020D	512	8 GB	1. Python with MQTT Paho lib connected to IBM Bluemix Cloud 2. POSIX C 3. Java SE-e 4. OSGi - Eclipse Kura / Lua
3	**Netbiter / HMS**						
3.1		Netbiter EC350	Linux BusyBox, Optional activation via Argos Cloud and it is Java SE and C enabled	ARM Cortex	1024	0,5 (512 MB - with restricted partitions))	1. HMS GW SW Stack -with ZeroMQ, Native SNMP, Native Protocol Agents (with samples for Java and C) and Netbiter Argos Cloud 2. POSIX C 3. Java SE-e
3.2		Netbiter EC310	Linux BusyBox, Optional activation via Argos Cloud and it is Java SE and C enabled	ARM Cortex	1024	0,5 (512 MB - with restricted partitions))	1. HMS GW SW Stack -with ZeroMQ, Native SNMP, Native Protocol Agents (with samples for Java and C) and Netbiter Argos Cloud 2. POSIX C 3. Java SE-e

Embedded OS HW Details

4	**Gemalto**						
4.1		Cinterion Concept Board	Proprietary OS? With Java ME 3.2 embedded on EHS6 chip	ARM 11	10	0,01 GB (10 MB user space for JME Midlets)	1. Java ME 3.2
5	**Microsoft (boards promoted by Microsoft)**						
5.1		MinnowBoard MAX / Rpi 2 Model B+	Windows 10 IoT Core, Debian GNU/Linux, Yocto Project Compatible, Android 4.4 System	Intel Atom x64 E38xx Series SoC	1024 / 2048	8 GB	1. Microsoft C# .NET / C++ 2. POSIX C 3. Java SE-e 4. JavaScript on Node.js
5.2		Sharks Cove	Windows 8.1	Intel ATOM Z3735G	1024	16 GB	1. Microsoft C# .NET / C++
5.3		Qualcomm DragonBoard 410C	Linux based on Ubuntu, and planned support for Windows 10 IoT Core RT	ARM Cortex A7	1024	8 GB	1. POSIX C 2. C++ 3. Java SE-e 4. JavaScript - Node.js / Microsoft C#

Embedded OS HW Details

6	**BoundaryDevice**						
6.1		Nitrogen6 SabreLite	Android or Linux Ubuntu 14 Trusty T	ARM Cortex A9 dual-core	512	4 GB	1. POSIX C 2. C++ 3. Java SE-e 4. JavaScript - Node.js / Python
6.2		Nitrogen6 MAX	Android or Linux Ubuntu 14 Trusty T	ARM Cortex A9 quad-core	1024	8 GB	1. POSIX C 2. C++ 3. Java SE-e 4. JavaScript - Node.js / Python
7	**Intel (GW HW provider - please see the above models)**						
7.1		Intel Edison + Arduino Breakout Board	Yocto Linux, Arduino, Node.js (nodeRED)	Intel ATOM SoC	1024	8 GB	1. C-Arduino 2. Gnu C/C++
7.2		Galileo Gen 2	Yocto Linux	Intel Quark SoC X1000	256	8 GB	1. C-Arduino 2. Python 3. JavaScript - Node.js 4. POSIX C
7.3		Intel Galileo	Yocto Linux	Intel ATOM x64	256	8 GB	1. C-Arduino 2. Python 3. JavaScript - Node.js

Embedded OS HW Details

8	**Beagle Boards**						
8.1		BeagleBone Black	Debian Android Ubuntu Cloud9 IDE on Node.js w/ BoneScript library	ARM Cortex A8 Sitara	512	4 GB	1. Java SE-e 2. C/C++ 3. JavaScript - Node.js 4. Python
9	**Raspberry**						
9.1		Rpi Modelb / Raspberry Pi 2 Model B+	Raspbian Wheezy OS/Debian Linux Embedded	Broadcom BCM2835 SoC/BCM2836 SoC, single-Core/Quad-core ARM Cortex-A7	512 / 1024	4GB	1. Python 2. C/C++ 3. JavaScript - Node.js + node-Red 4. Java 8 SE-e 5. Java 8 ME-e
10	**Arduino**						
10.1		Arduino Yun	Linux eMbedded OS + Arduino Environment	Atheros AR9331 MIPS/ARM + ATmega32u4	64	2 GB	1. C-Arduino + Bridge Library - HTTP and Python

Embedded OS HW Details (for IoT Nodes)

1	**ESP**						
1.1		ESP8266 ESP-01 board	Proprietary OS/Firmware with Interpretors	106micro Diamond Standard core (LX3)	64 KB Instruction 96 KB data	64 KB	1. AT/HTTP Commands 2. Lua (with NodeMCU Lua)
1.2		NodeMcu Lua WIFI IoT dev board based ESP8266 module	Proprietary OS/Firmware with Interpretors	106micro Diamond Standard core (LX3)	64 KB Instruction 96 KB data	64 KB	1. AT/HTTP Commands 2. Lua (with NodeMCU Lua)
1.2		ESP8266 ESP-12 board	Proprietary OS/Firmware with Interpretors	106micro Diamond Standard core (LX3)	?	?	1. HTTP Commands 2. Lua (with NodeMCU Lua)
2	**NXP (NXP acquired Freescale & Qualcomm acquired NXP)**						
2.1		ARM mbed NXP LPC1768 Development Board	?	ARM Cortex-M3	32 KB	512 KB	1. ARM mbed C and Library
2.2		Freescale FRDM-K64	?	ARM Cortex-M3	256 KB	1024 KB	1. ARM mbed C

1. IoT Communications Protocols HTTP-REST, CoAP, MQTT

1.Infrastructure (e.g.: 6LowPAN, IPv4/IPv6, RPL, ZigBee, Z-Wave, SigFox, LoraWAN, GSM-3G/4G. WiMAX)

2.Identification (e.g.: EPC, uCode, IPv6, URIs)

3.Comms / Transport (e.g.: Wi-Fi, Bluetooth, BLE, LPWAN, ZigBee, Z-Wave, SigFox, LoraWAN, GSM 3G/4G. WiMAX)

4.Discovery (e.g.: Physical Web, mDNS, DNS-SD)

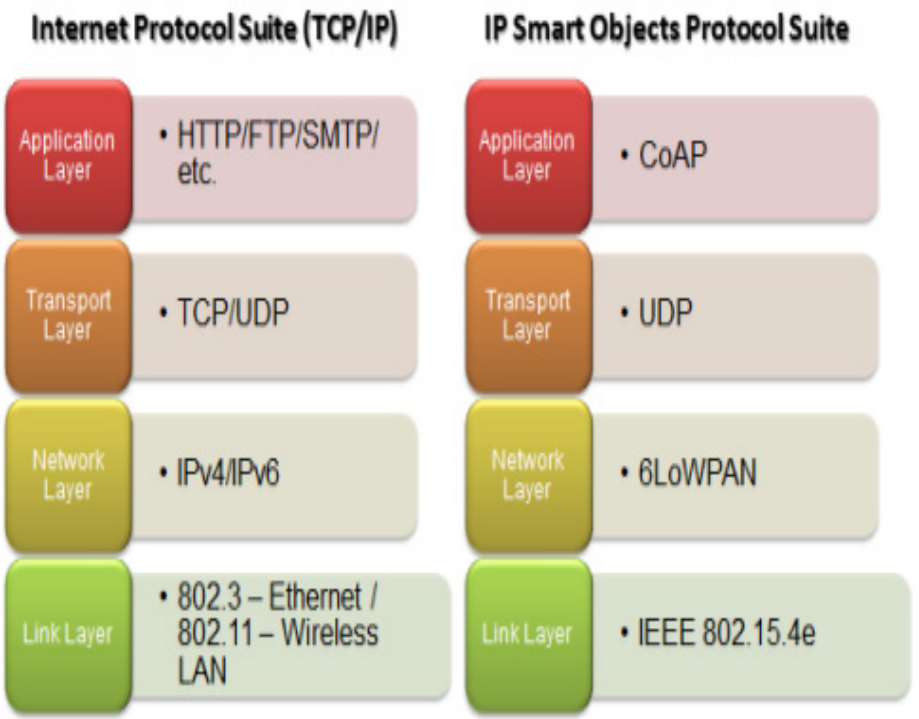
5.Data Protocols (e.g.: MQTT, CoAP/HTTP-REST, AMQP, Websocket, LWM2M, ModBus)

6.Device Management (e.g.: TR-069, OMA-DM)

7.Semantic (e.g.: JSON-LD, Web Thing Model)

8.Multi-layer Frameworks (e.g.: Alljoyn, IoTivity, Weave, Homekit, etc.)

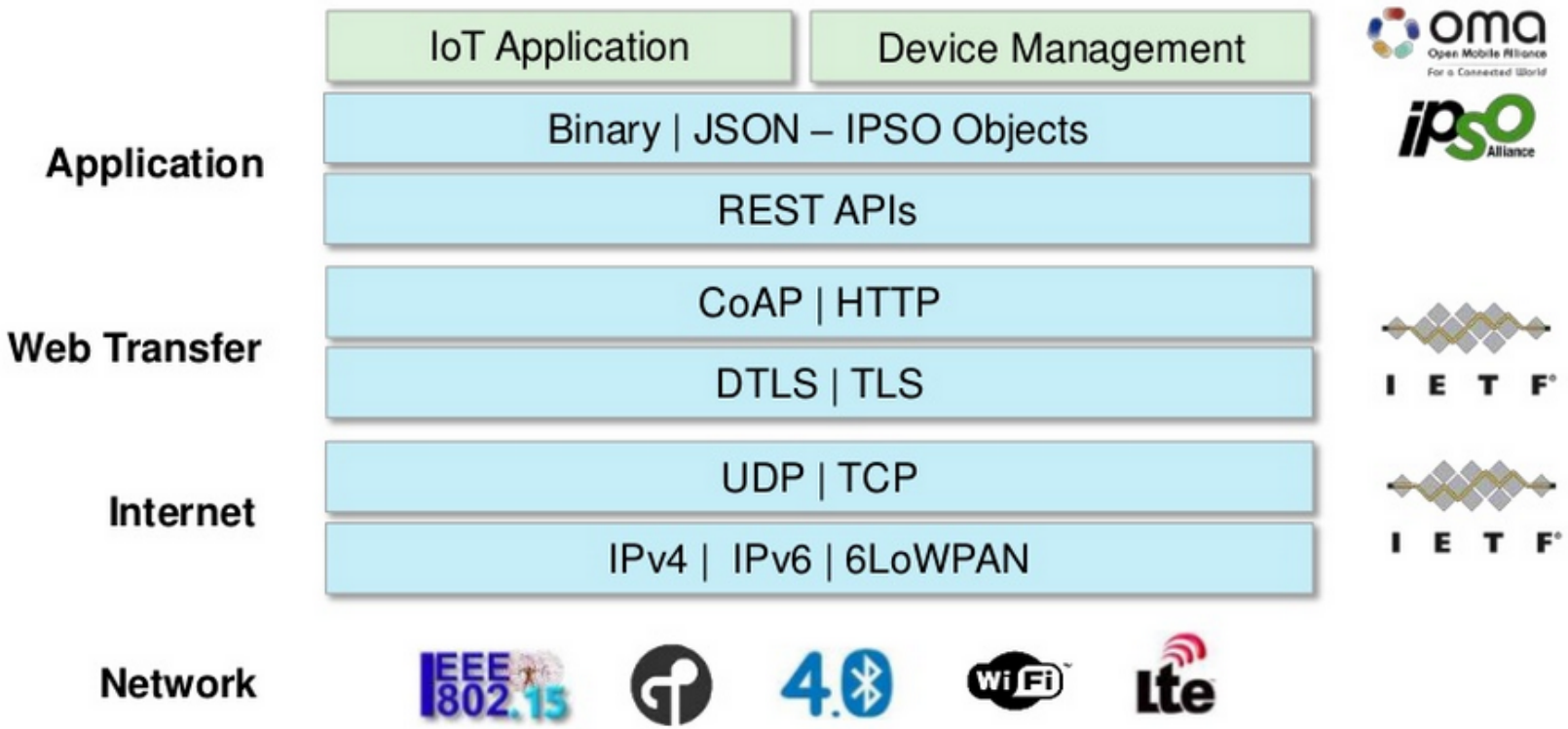
9. **Security** – Transversal (e.g. Open Trust Protocol (OTrP) in TEE, X509)



Copyright: http://www.postscapes.com/internet-of-things-protocols/#iot-architecture-graphics

1. IoT Communications Protocols HTTP-REST, CoAP, MQTT

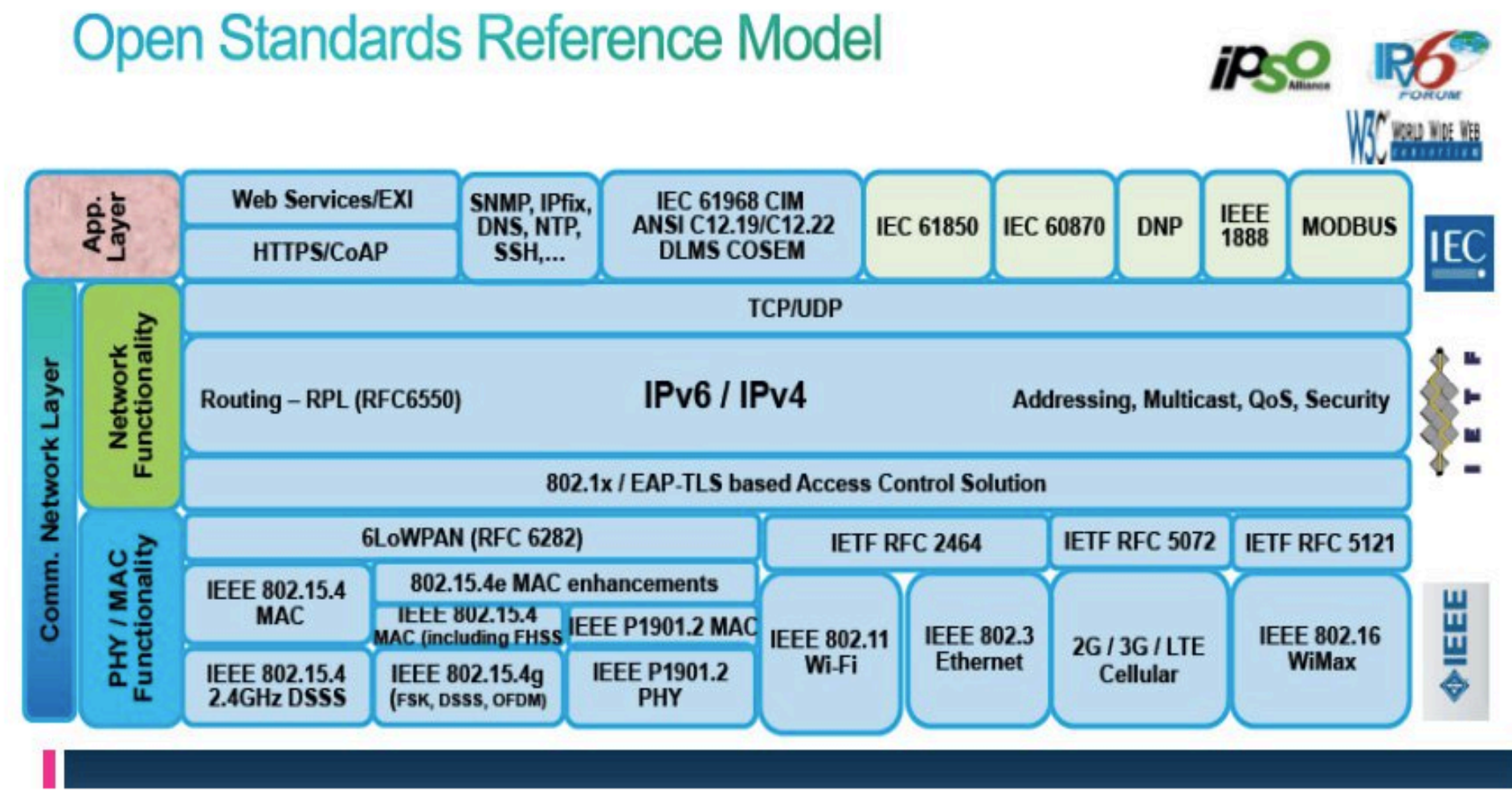
Remember the I in IoT!



ARM

Copyright: http://www.postscapes.com/internet-of-things-protocols/#iot-architecture-graphics

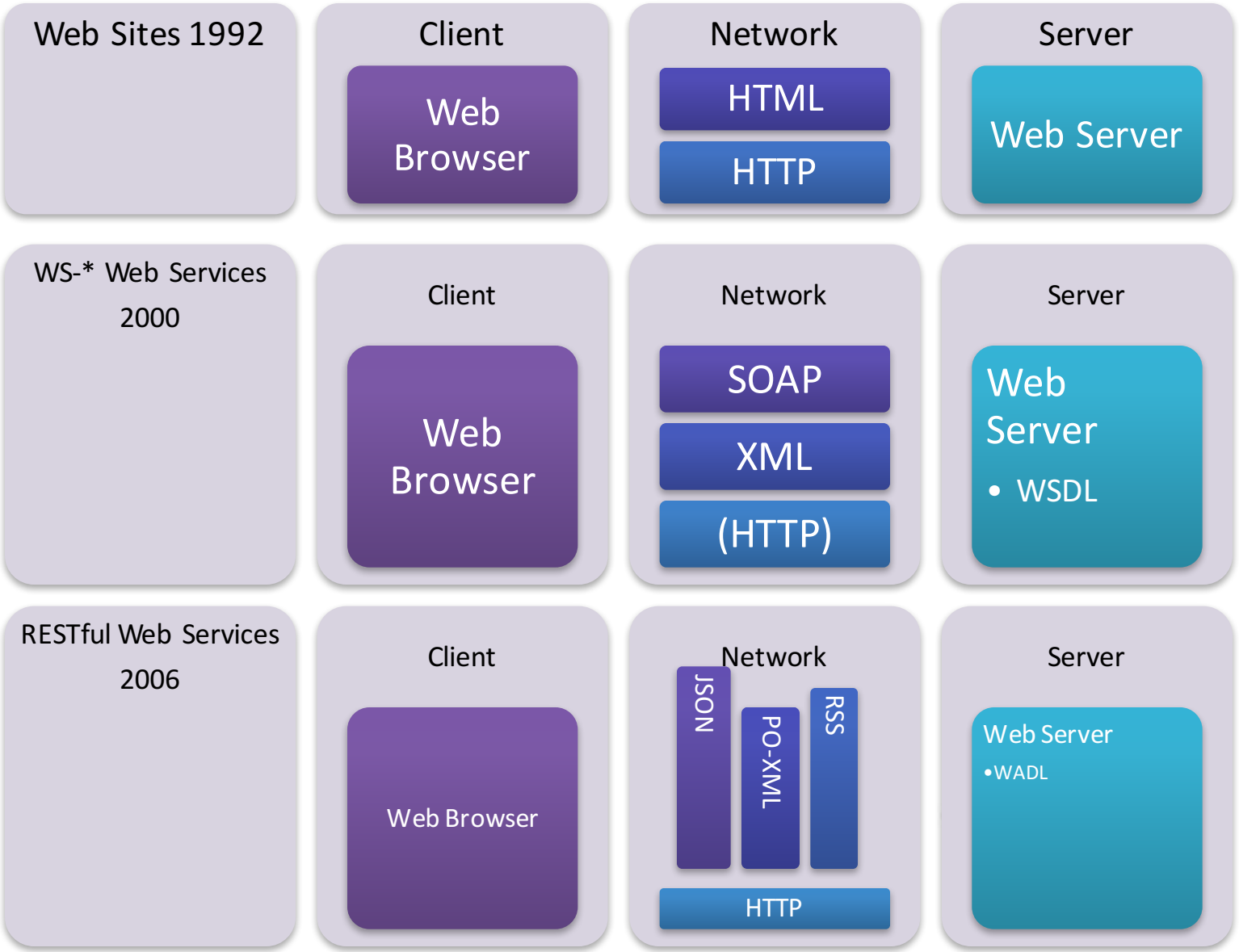
1. IoT Communications Protocols HTTP-REST, CoAP, MQTT



David E Culler Open Standards Reference Model

Copyright: http://www.postscapes.com/internet-of-things-protocols/#iot-architecture-graphics

1. IoT Communications Protocols HTTP-REST, CoAP, MQTT



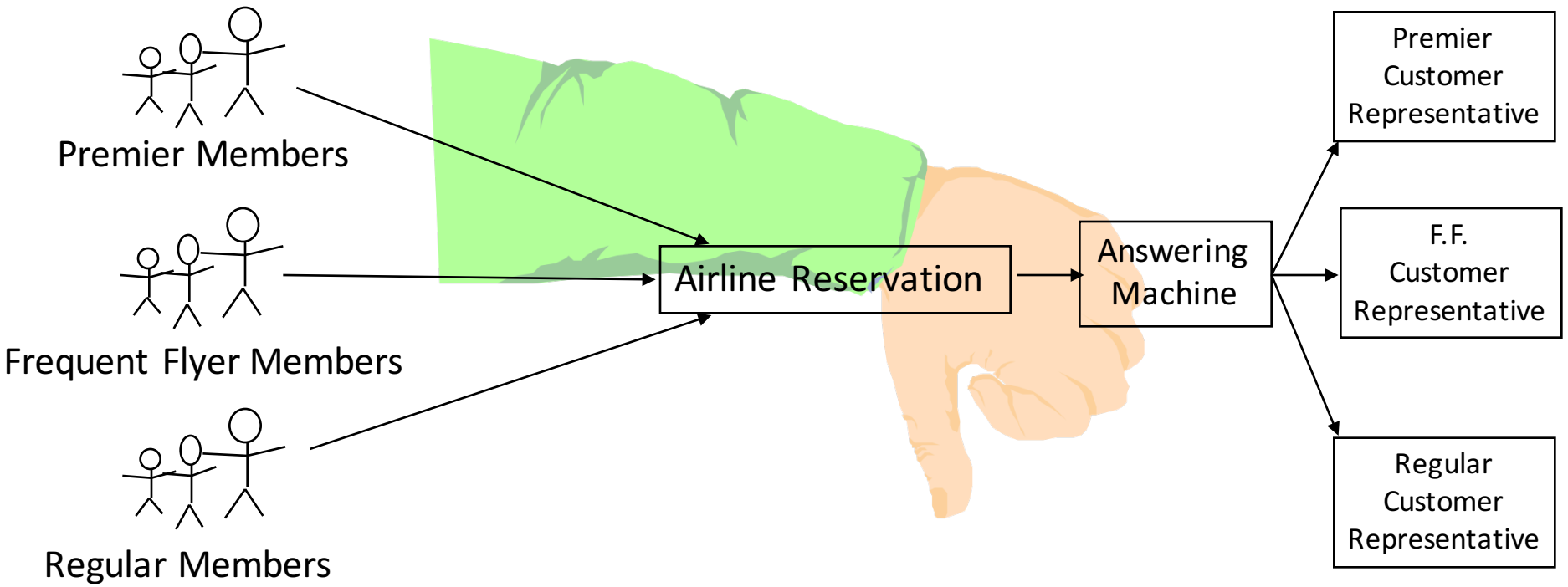
Copyright: http://www.jopera.org/files/soa-amsterdam-restws-pautasso-talk.pdf

1. IoT Communications Protocols HTTP-REST, CoAP, MQTT

WS-* vs. REST Comparison

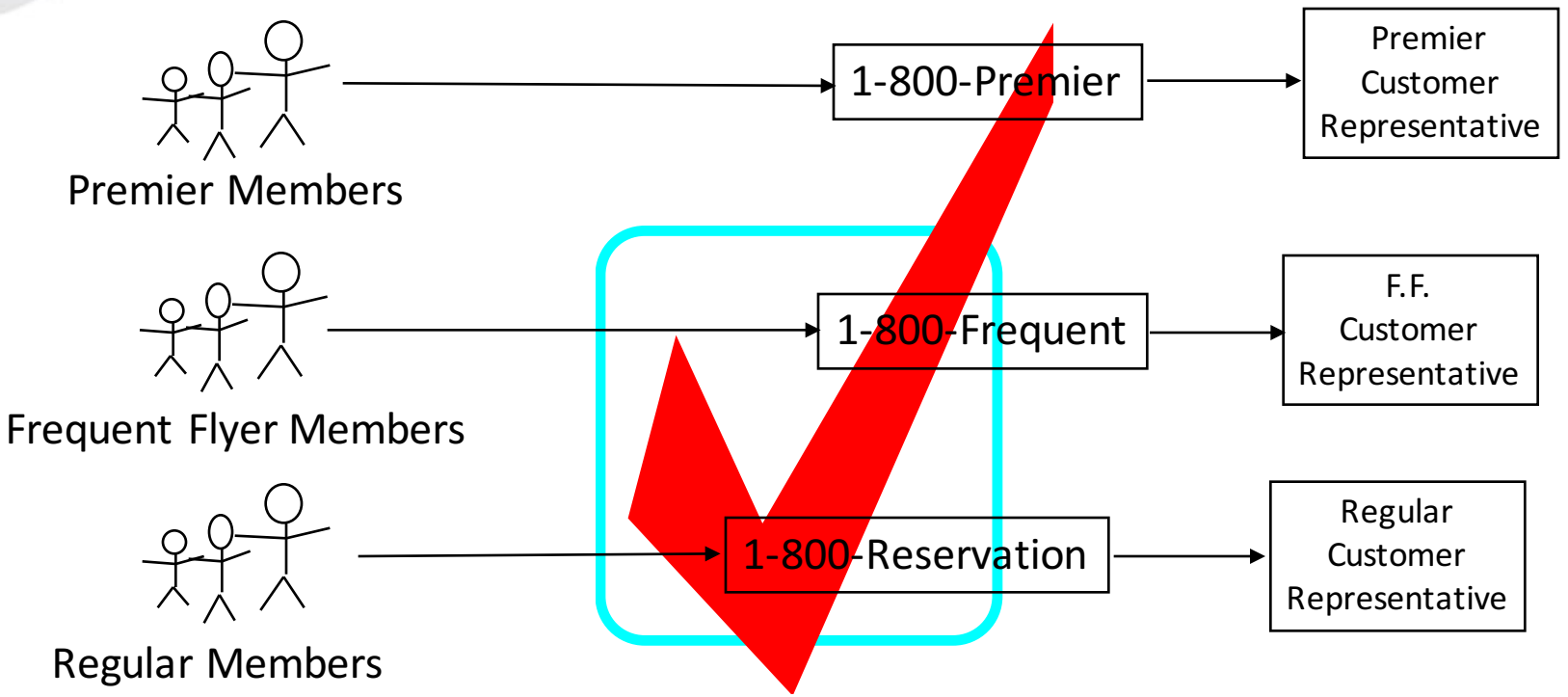
Copyright: http://www.jopera.org/files/soa-amsterdam-restws-pautasso-talk.pdf

This Ain't the REST Design Pattern



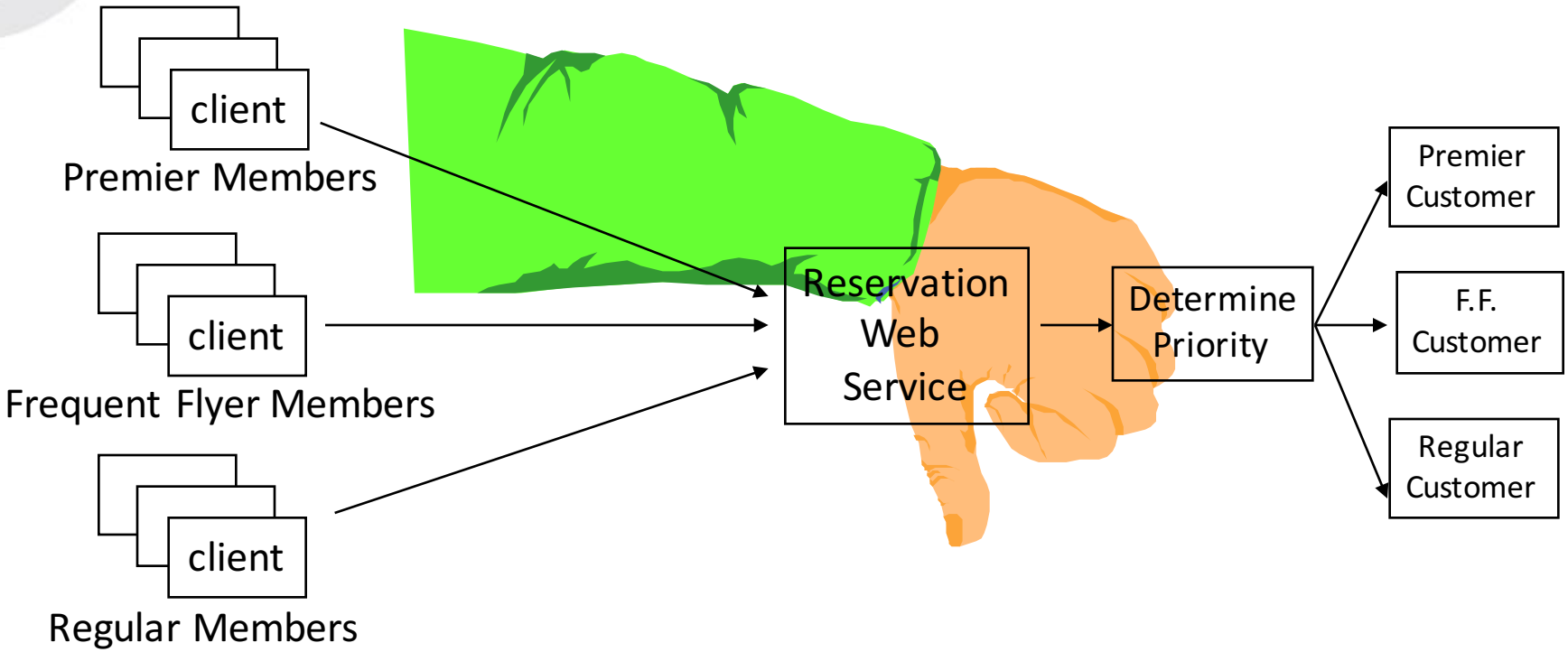
Copyright: http://www.xfront.com/5-minute-intro-to-REST.ppt

This is the REST Design Pattern



Copyright: http://www.xfront.com/5-minute-intro-to-REST.ppt

This ain't the REST Design Pattern



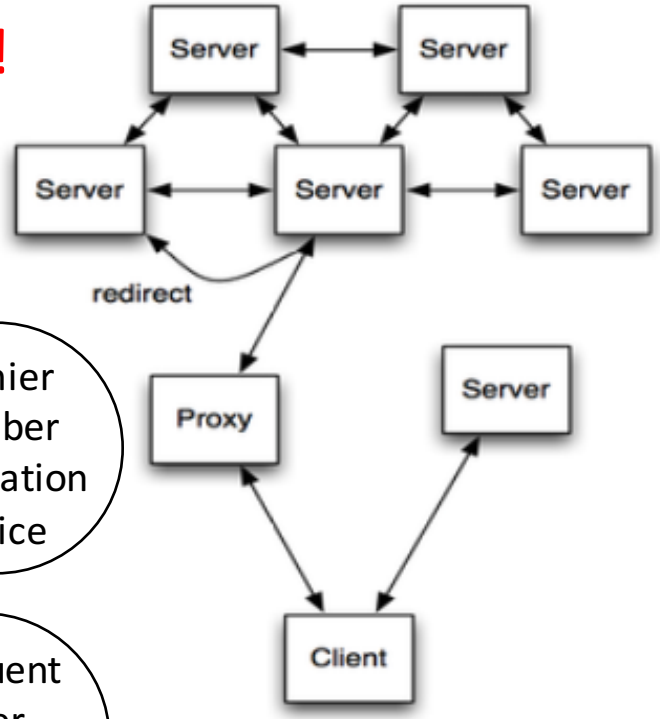
Copyright: http://www.xfront.com/5-minute-intro-to-REST.ppt

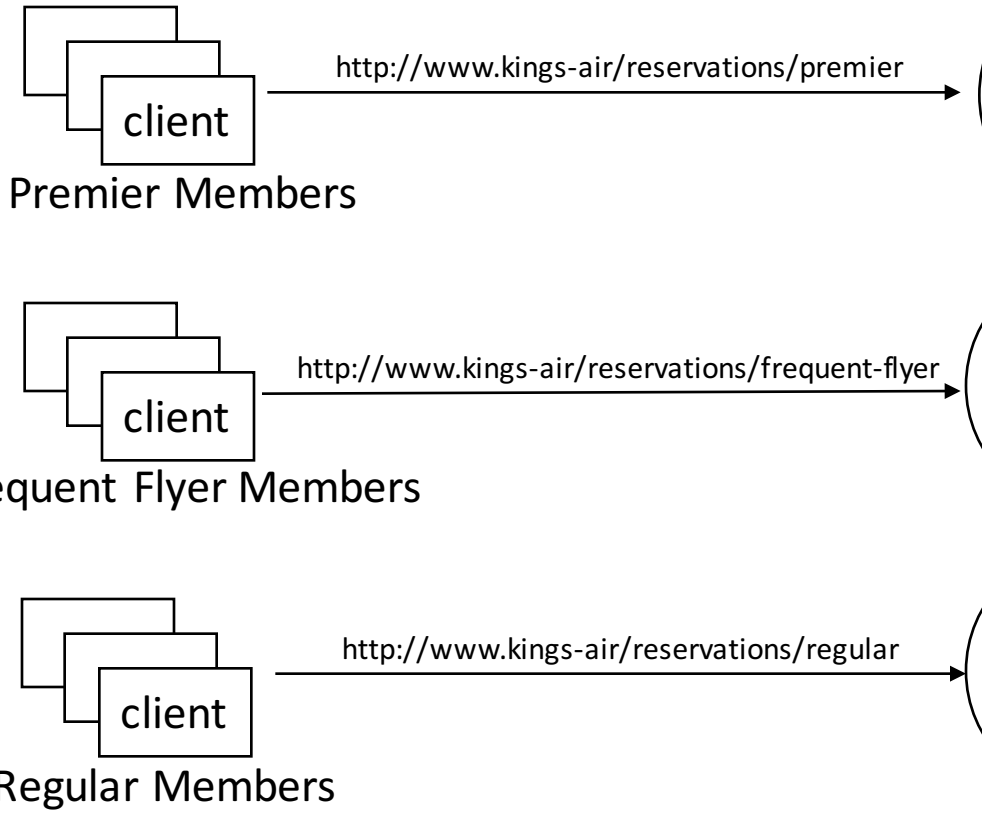
1. IoT Communications Protocols HTTP-REST vs. WS Recap

REST Approach in Web Architecture:

URLs are Cheap! Use Them!

The airline provides several URLs - one URL for premier members, a different URL for frequent flyers, and still another for regular customers.

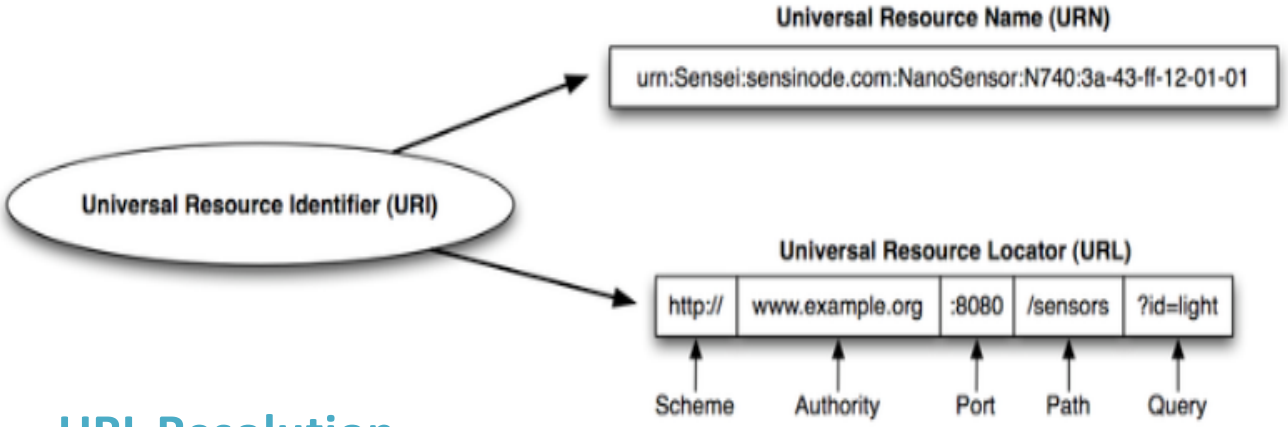




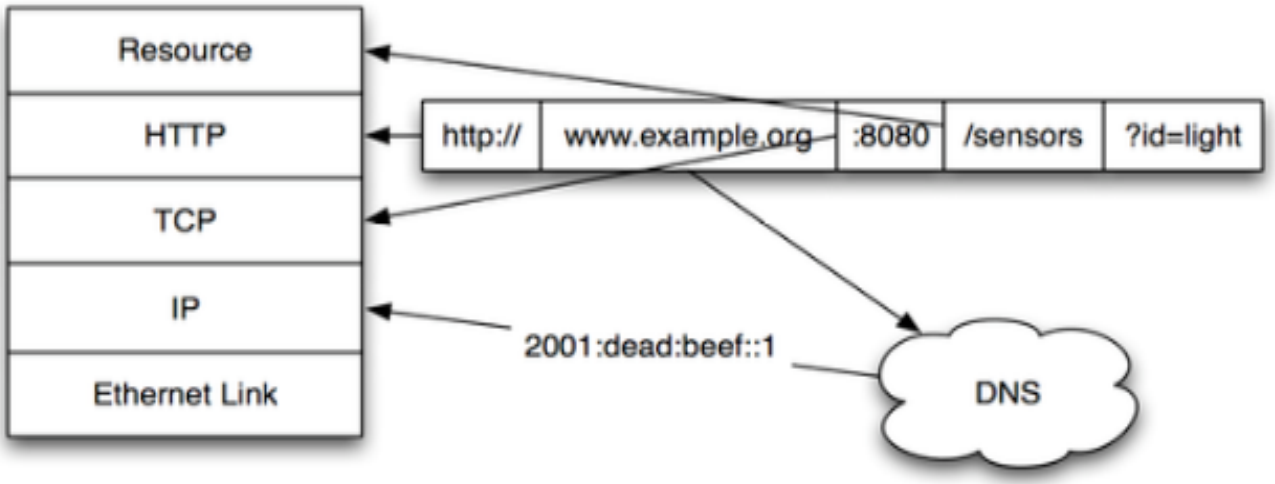
Copyright: http://www.xfront.com/5-minute-intro-to-**REST**.ppt

1. IoT Communications Protocols HTTP-REST Recap

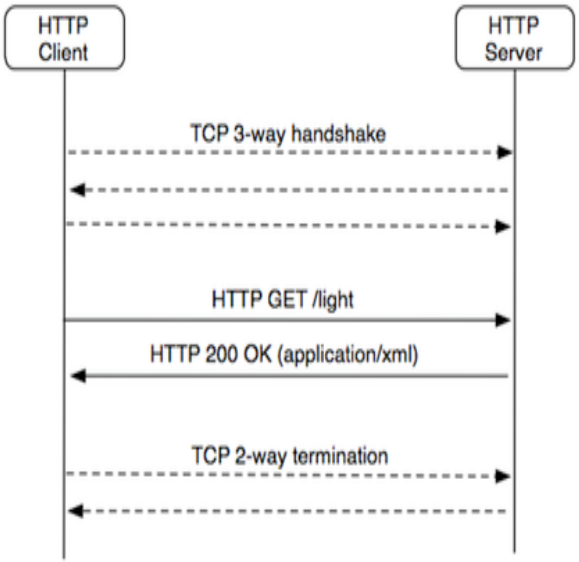
Web Naming



URL Resolution



An HTTP Request

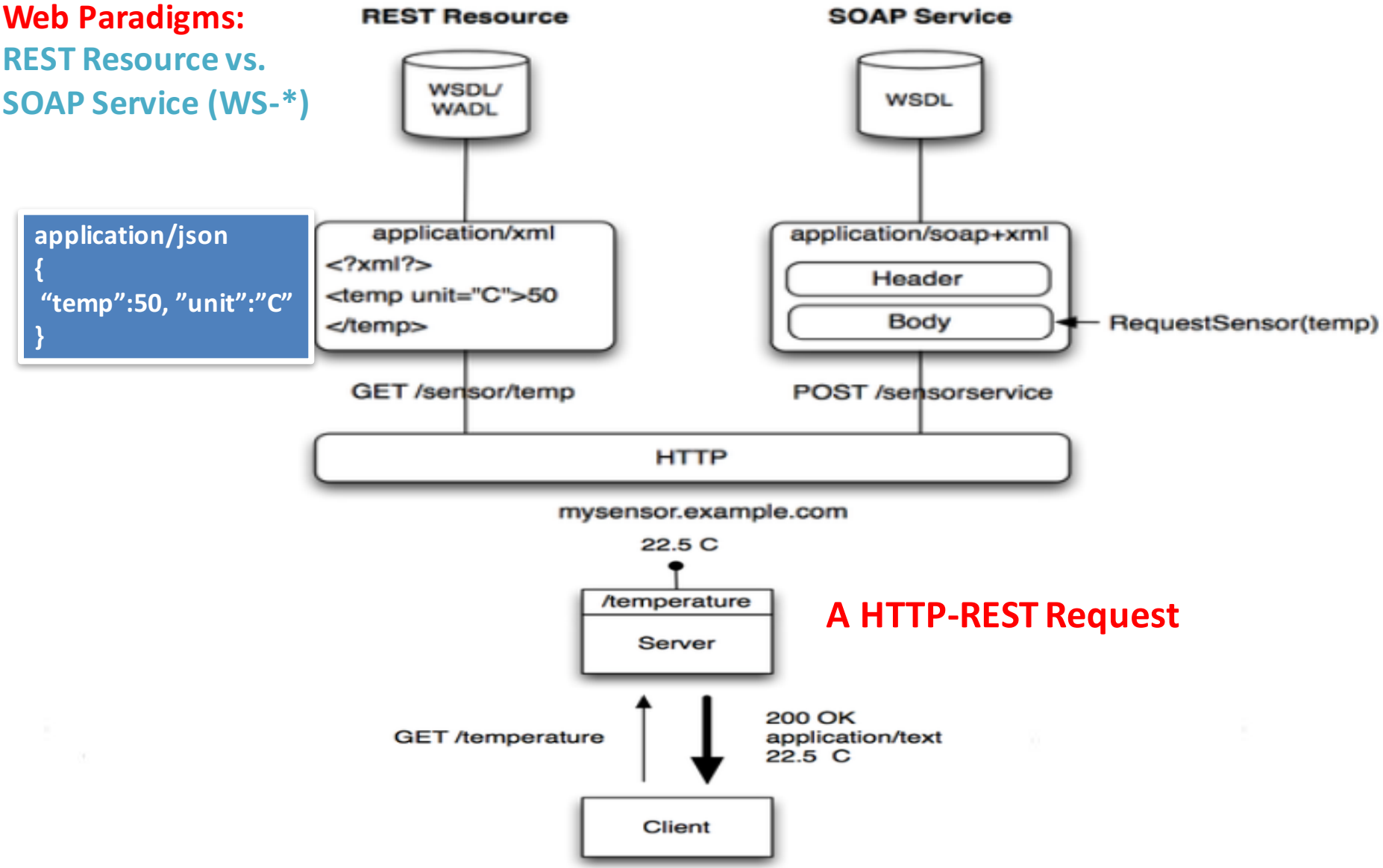


See RFC2616 - Hypertext Transfer Protocol v1.1

Copyright: https://community.arm.com/servlet/JiveServlet/downloadBody/8633-102-2-15471/ARM%20CoAP%20Tutorial%20April%2030%202014.pdf

1. IoT Communications Protocols HTTP-REST Recap

Web Paradigms:
REST Resource vs.
SOAP Service (WS-*)



A HTTP-REST Request

Copyright: https://community.arm.com/servlet/JiveServlet/downloadBody/8633-102-2-15471/ARM%20CoAP%20Tutorial%20April%2030%202014.pdf

1. IoT Communications Protocols HTTP-REST vs. WS Recap

- Simple **web service** as an example: querying a phonebook application for the details of a given user

- Using Web Services and SOAP, the request would look something like this:

```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
        <soap:body
  pb="http://www.acme.com/phonebook">
                <pb:GetUserDetails>
                <pb:UserID>12345</pb:UserID>
                </pb:GetUserDetails>
        </soap:Body>
</soap:Envelope>
```

- Simple **REST service** as an example

- And with REST? The query will probably look like this:

  http://www.acme.com/phonebook/UserDetails/12345

- GET /phonebook/UserDetails/12345 HTTP/1.1

  Host: www.acme.com

  Accept: application/xml

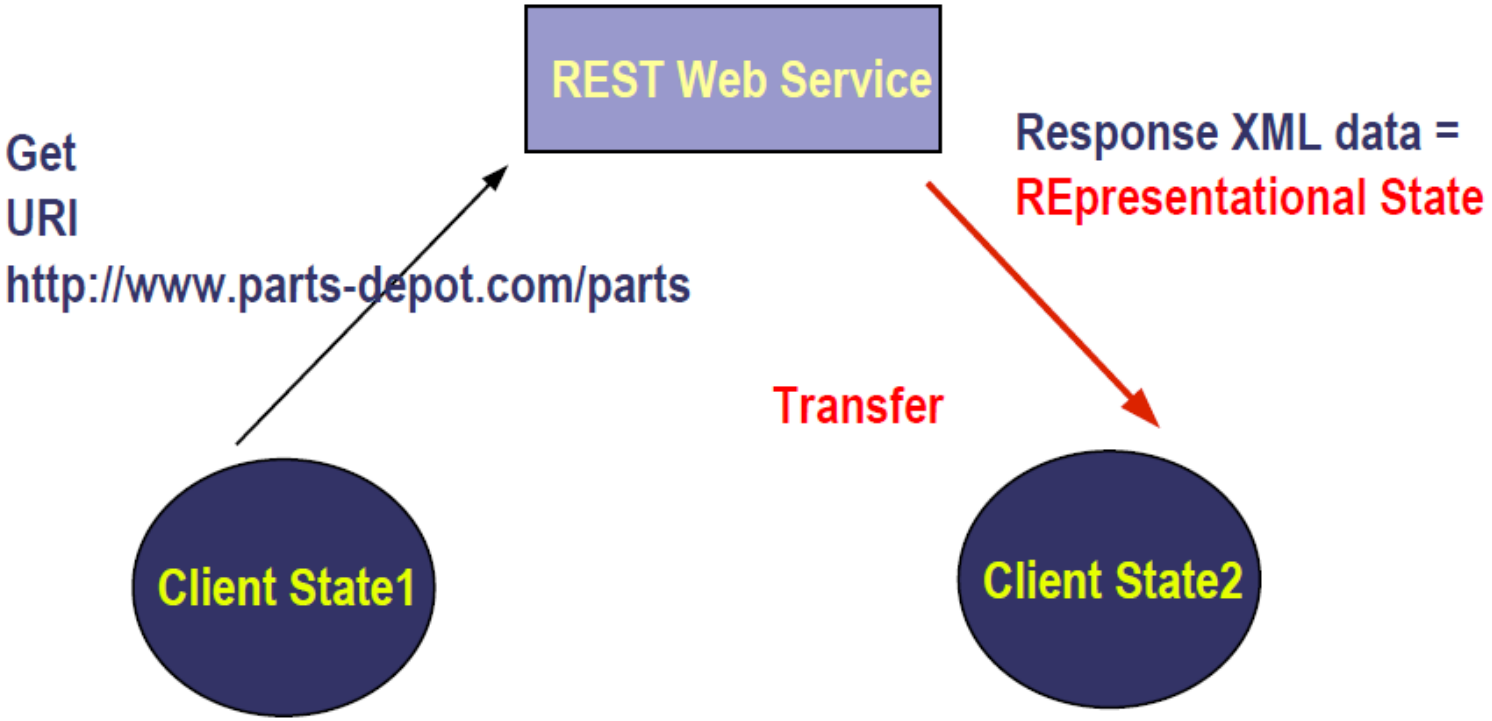
- Complex query:

  http://www.acme.com/phonebook/UserDetails?firstName=John&lastName=Doe

Copyright: http://cobweb.cs.uga.edu/~mullangi/db/**REST.ppt**

What is REST?

REpresentational State Transfer



HTTP Request/Response As REST

http://www.javapassion.com/webservices/RESTPrimer.pdf

1. IoT Communications Protocols HTTP-REST Recap

REST over HTTP – Uniform interface

- CRUD operations on resources
  - Create, Read, Update, Delete
- Performed through HTTP methods + URI

Operation	SQL	HTTP
Create	INSERT	PUT / POST
Read (Retrieve)	SELECT	GET
Update (Modify)	UPDATE	PUT / PATCH
Delete (Destroy)	DELETE	DELETE

CRUD Operations	4 main HTTP methods	
	Verb	**Noun**
Create (Single)	POST	Collection URI
Read (Multiple)	GET	Collection URI
Read (Single)	GET	Entry URI
Update (Single)	PUT	Entry URI
Delete (Single)	DELETE	Entry URI

POST = Create
GET = Read
PUT = Update
DELETE = Delete

http://www.javapassion.com/webservices/RESTPrimer.pdf

Copyright: http://cobweb.cs.uga.edu/~mullangi/db/REST.ppt

1. IoT Communications Protocols CoAP vs. MQTT

HTTP + HTML – The Web Protocol

What was the element of success for the Web?

HTML

Uniform representation of documents;

URIs

Uniform Referents for Data and Services on the Web;

HTTP

Universal transfer protocol;

Enables a Distribution System of Proxies and Reverse Proxies

Copyright: http://www.inginfpoliba.eu/index.php?action=downloadfile&filename=8-Application%20Layer%20for%20IoT.pdf&directory=Interet%20of%20Things

1. IoT Communications Protocols CoAP vs. MQTT

REST and Web Architecture

REpresentational State Transfer

Relies on a stateless, client-server, cacheable communication protocol

Instead of using complex mechanisms to connect between machines, simple HTTP is used to make call between machines

RESTful applications use HTTP requests to post data (create and/or update), read data (e.g., make queries), and delete data.

Thus, REST uses HTTP for all four CRUD (Create/Read/Update/Delete) operations.

Do Not Forget: REST is not a protocol nor a standard, but an **ARCHITECTURAL STYLE**

Copyright: http://www.inginfpoliba.eu/index.php?action=downloadfile&filename=8-Application%20Layer%20for%20IoT.pdf&directory=Interet%20of%20Things

1. IoT Communications Protocols CoAP vs. MQTT

HTTP: Why not in IoT? – although possible to be in IoT GWs but rare in IoT Nodes

8/16-bit Microcontrollers, with limited RAM and ROM;

Constrained networks such as 6LoWPAN support the fragmentation of IPv6 packets into small link- layer frames, however incurring significant reduction in packet delivery probability;

TCP as the Transport Protocol, too heavy for LLN motes;

SSL/TLS for security: too heavy;

Copyright: http://www.inginfpoliba.eu/index.php?action=downloadfile&filename=8-Application%20Layer%20for%20IoT.pdf&directory=Interet%20of%20Things

1. IoT Communications Protocols CoAP

CoAP – Default UDP Port 5683

IETF CoRE, designed to ensure interoperability with the WEB (GET, PUT, POST,DELETE).

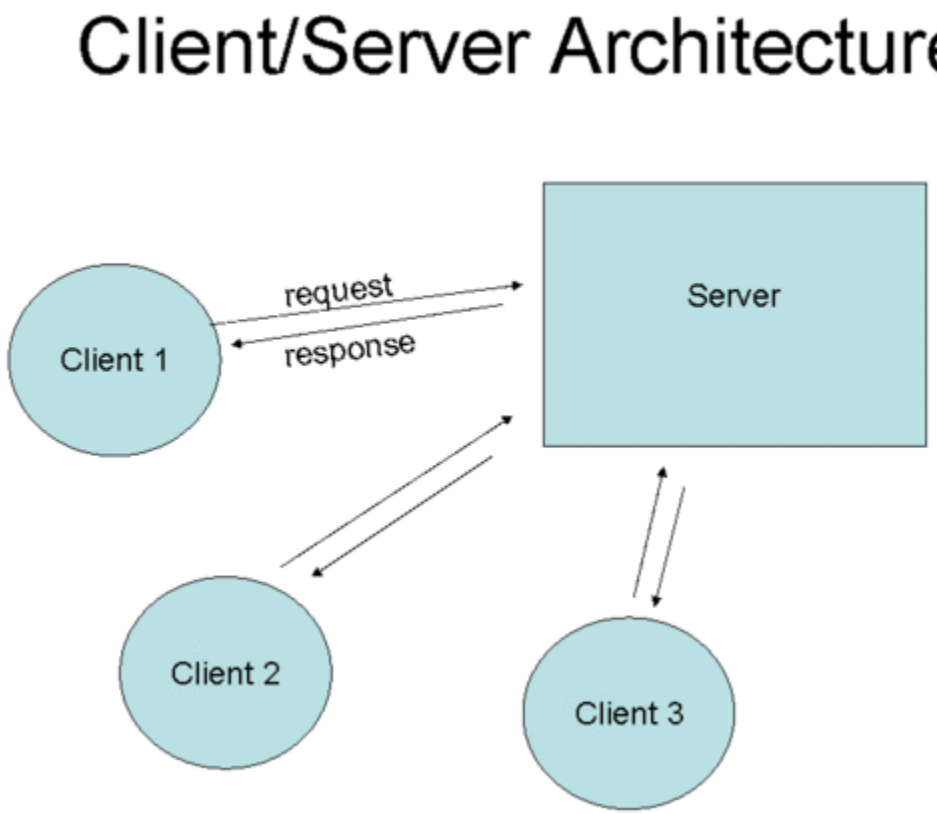
Last Update: v18, 28 June 2013, Link:
http://tools.ietf.org/html/draft-ietf-core-coap-18

- Document-Centric.
- Request/Response mode, with the Observe flag.
- UDP binding, with optional reliability supporting unicast and multicast request (5683 UDP Port)
- Asynchronous Messages Exchanges
- Low Header Overhead and Parsing Complexity
- Simple proxying and Caching Capabilities
- Security binding to DTLS

Copyright: http://www.inginfpoliba.eu/index.php?action=downloadfile&filename=8-Application%20Layer%20for%20IoT.pdf&directory=Interet%20of%20Things

1. IoT Communications Protocols CoAP

CoAP – Interaction Model



A CoAP implementation acts both in client and server role

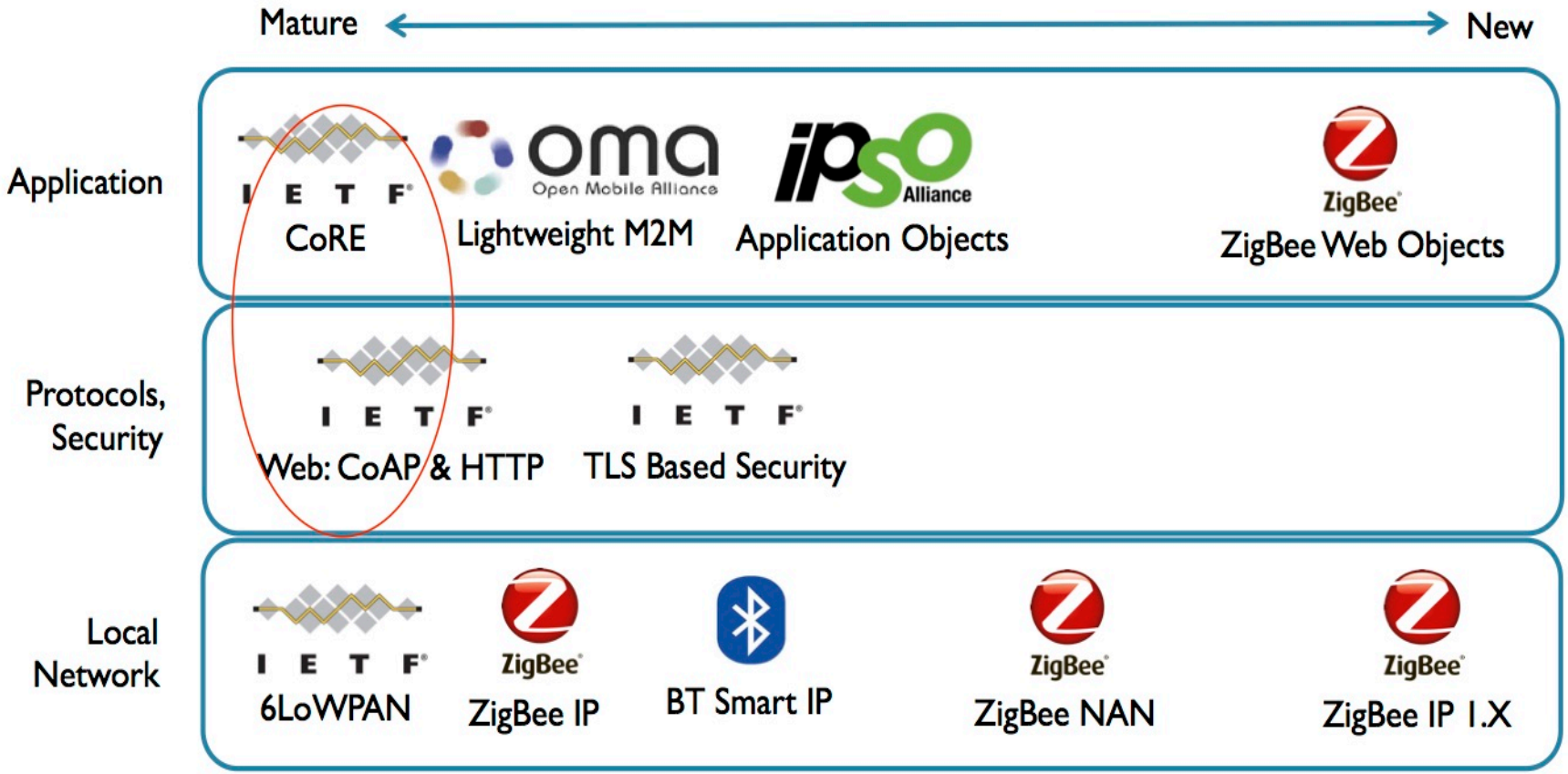
Response Code;

Asynchronous Exchange

Copyright: http://www.inginfpoliba.eu/index.php?action=downloadfile&filename=8-Application%20Layer%20for%20IoT.pdf&directory=Interet%20of%20Things

1. IoT Communications Protocols CoAP

CoAP is One Key IoT Standard

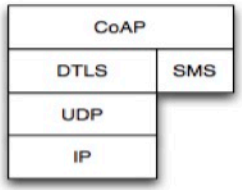


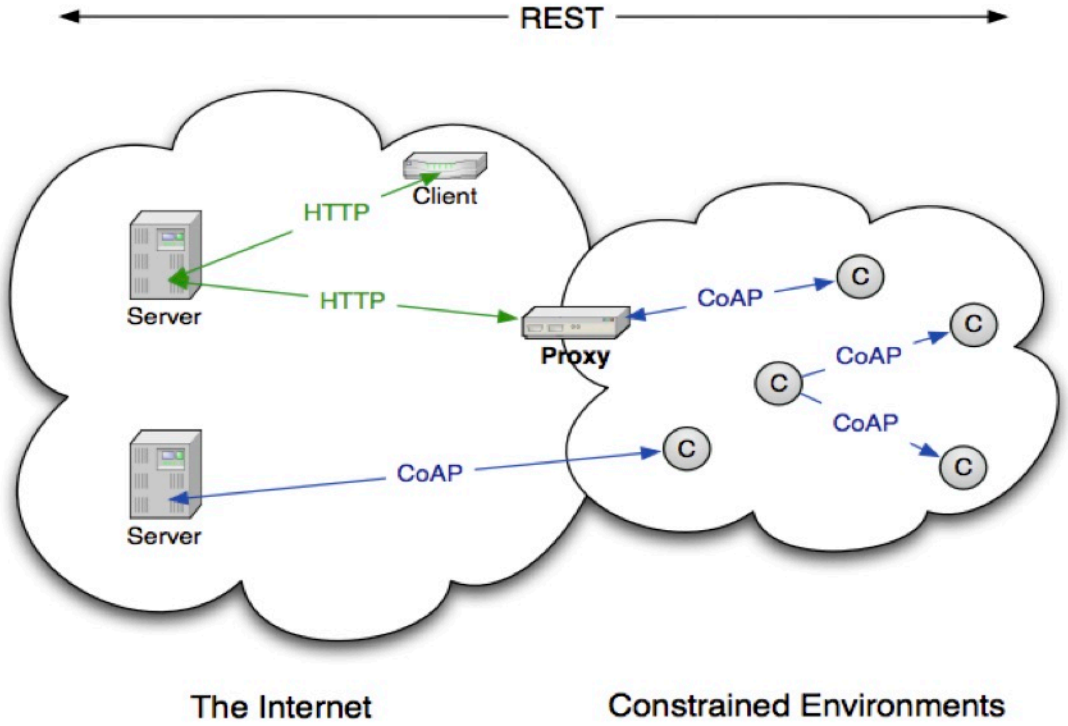
ARM

Copyright: https://community.arm.com/servlet/JiveServlet/downloadBody/8633-102-2-15471/ARM%20CoAP%20Tutorial%20April%2030%202014.pdf

CoAP: The Web of Things Protocol

- Open IETF Standard
- Compact 4-byte Header
- UDP, SMS, (TCP) Support
- Strong DTLS Security
- Asynchronous Subscription
- Built-in Discovery





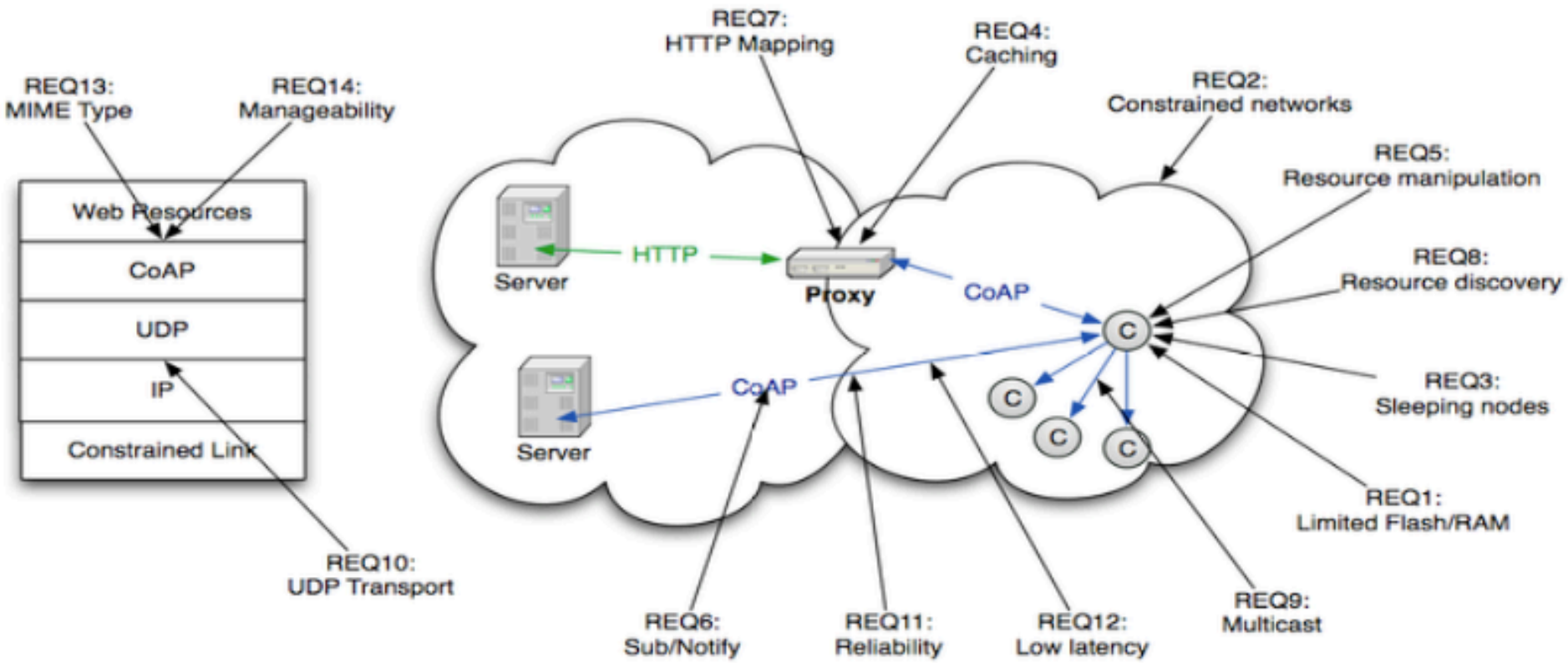
IETF

ARM

Copyright: https://community.arm.com/servlet/JiveServlet/downloadBody/8633-102-2-15471/ARM%20CoAP%20Tutorial%20April%2030%202014.pdf

1. IoT Communications Protocols CoAP

CoAP Design Requirements



Copyright: https://community.arm.com/servlet/JiveServlet/downloadBody/8633-102-2-15471/ARM%20CoAP%20Tutorial%20April%2030%202014.pdf

1. IoT Communications Protocols CoAP

What CoAP is (and is not)

- Sure, CoAP is
  - A very efficient RESTful protocol
  - Ideal for constrained devices and networks
  - Specialized for M2M applications
  - Easy to proxy to/from HTTP
- But hey, CoAP is not
  - A general replacement for HTTP
  - HTTP compression
  - Restricted to isolated "automation" networks

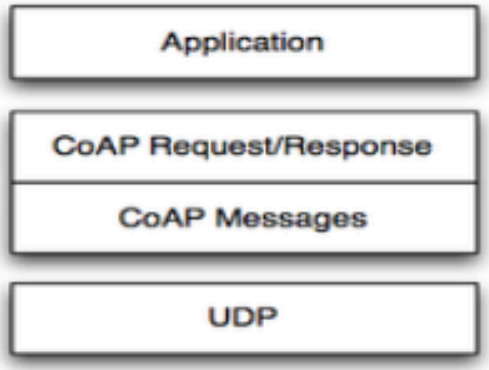
CoAP Features

- Embedded web transfer protocol (coap://)
- Asynchronous transaction model
- UDP binding with reliability and multicast support
- GET, POST, PUT, DELETE methods
- URI support
- Small, simple 4 byte header
- DTLS based PSK, RPK and Certificate security
- Subset of MIME types and HTTP response codes
- Built-in discovery
- Optional observation and block transfer

Copyright: https://community.arm.com/servlet/JiveServlet/downloadBody/8633-102-2-15471/ARM%20CoAP%20Tutorial%20April%2030%202014.pdf

1. IoT Communications Protocols CoAP

Transaction Model

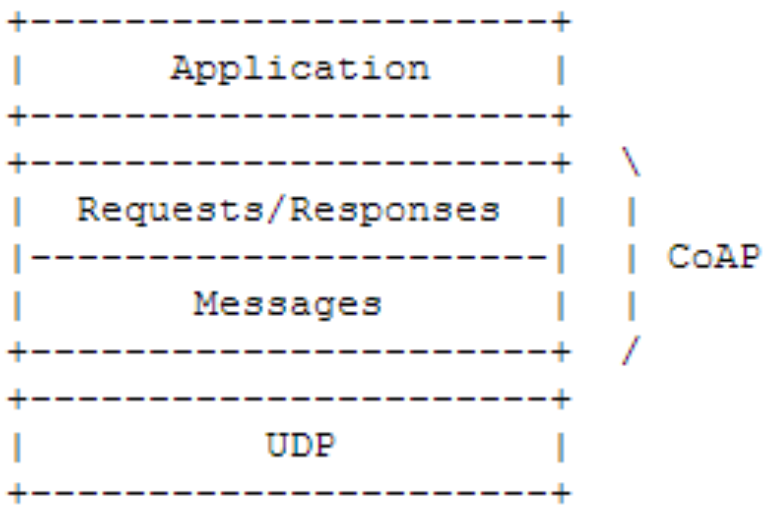


- Transport
  - CoAP currently defines:
  - UDP binding with DTLS security
  - CoAP over SMS or TCP possible
- Base Messaging
  - Simple message exchange between endpoints
  - Confirmable or Non-Confirmable Message answered by Acknowledgement or Reset Message
- REST Semantics
  - REST Request/Response piggybacked on CoAP Messages
  - Method, Response Code and Options (URI, content-type etc.)

Copyright: https://community.arm.com/servlet/JiveServlet/downloadBody/8633-102-2-15471/ARM%20CoAP%20Tutorial%20April%2030%202014.pdf

1. IoT Communications Protocols CoAP

CoAP – Two Layer Approach



Messages Layer: deal with UDP and the asynchronous nature of the interactions

Request Response Layer: Method and Response Codes

CoAP is however a single protocol, with messages and request/response just features of the CoAP header

Copyright: http://www.inginfpoliba.eu/index.php?action=downloadfile&filename=8-Application%20Layer%20for%20IoT.pdf&directory=Interet%20of%20Things

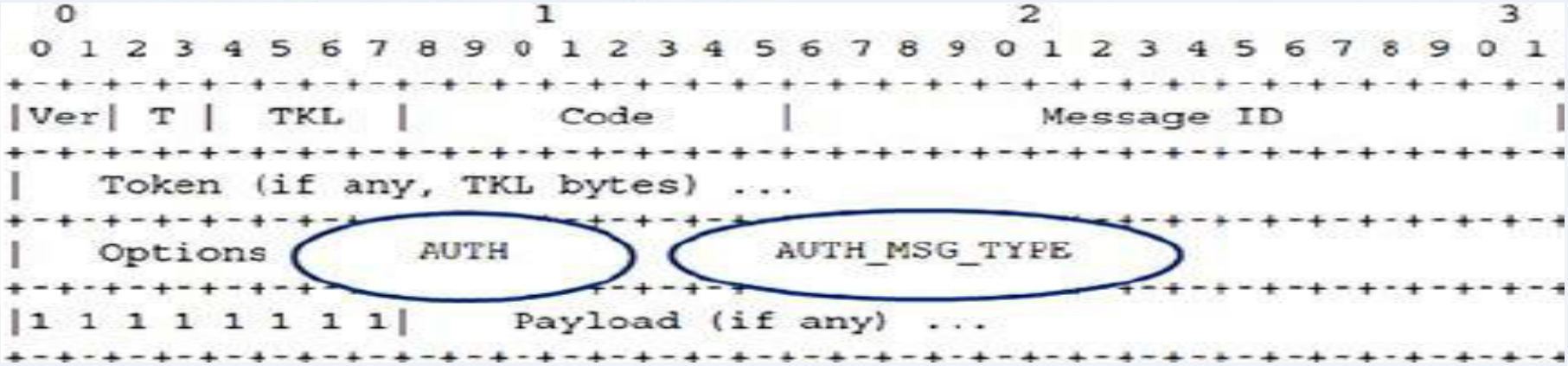
1. IoT Communications Protocols CoAP

Message Format

CoAP messages are encoded in a simple binary format.

The Message Header (4 bytes).

The variable-length token value 0 and 8 bytes long.



Ver - Version (1) → 2 bit unsigned integer . Implementations of this field to 1 (01 binary).

T – Message Type → 2 - bit unsigned integer. (Confirmable, Non-Confirmable, Acknowledgement, Reset).

TKL- Token Length → 4-bit unsigned integer. Indicates the length of the variable-length Token field (0-8 bytes).

Code – 8-bit unsighted integer. 3 bit class(most signification bits). 5 bits detail (least significant bits).

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

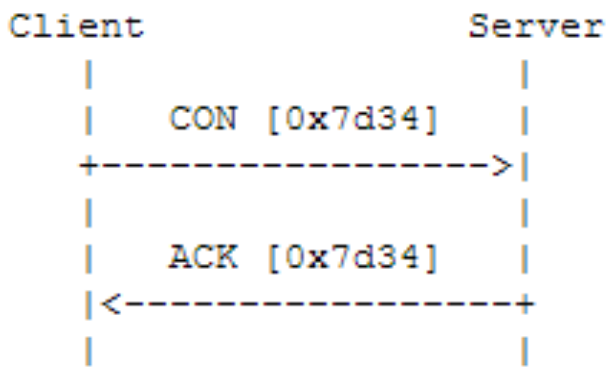
Message ID – 16-bit identifier for matching responses

Token – Optional response matching token

Standards Track RFC 7252
June, 2014

1. IoT Communications Protocols CoAP

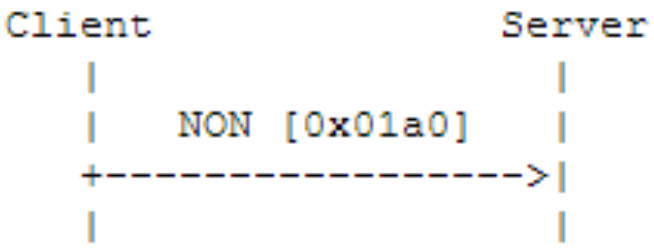
CoAP – Messaging Model



Confirmable (CON):

Default Timeout and Exponential Backoff, ACK with the same Message ID.

Reset option: if the server cannot support confirmable mode.



Non-Confirmable (NON):

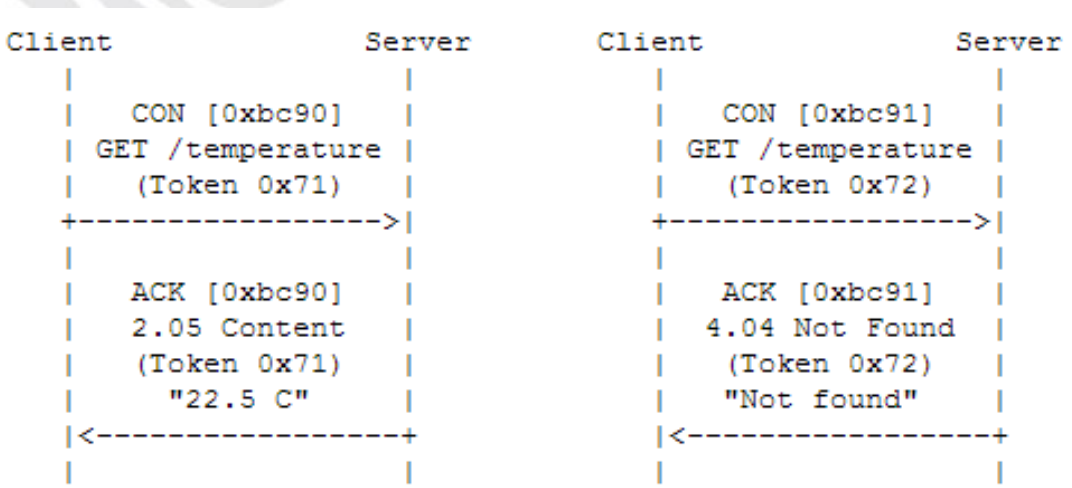
Simple data, Message ID for duplicate detection

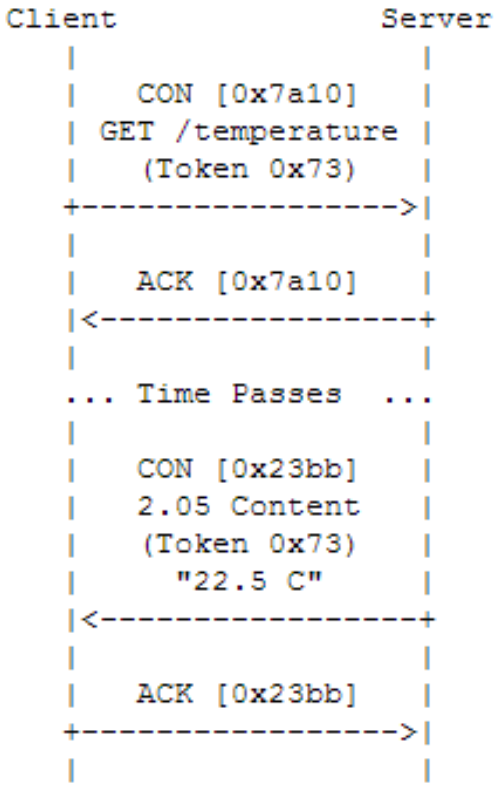
Reset Option: if the server cannot accept NON messages.

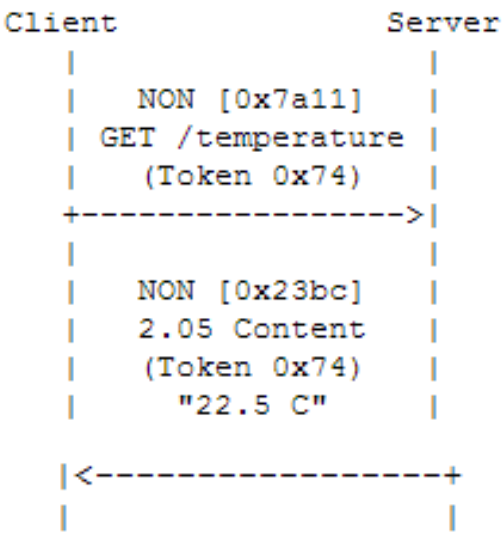
Copyright: http://www.inginfpoliba.eu/index.php?action=downloadfile&filename=8-Application%20Layer%20for%20IoT.pdf&directory=Interet%20of%20Things

1. IoT Communications Protocols CoAP

CoAP – Interaction Model





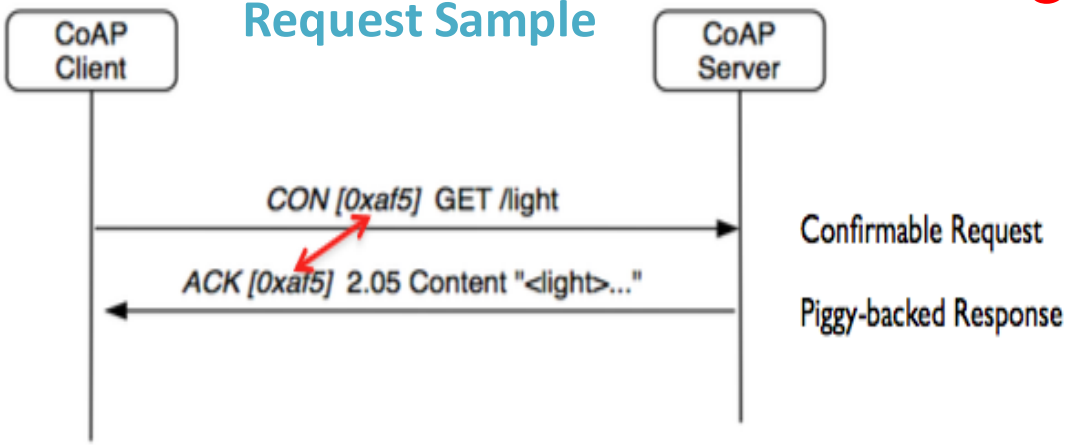


Copyright: http://www.inginfpoliba.eu/index.php?action=downloadfile&filename=8-Application%20Layer%20for%20IoT.pdf&directory=Interet%20of%20Things

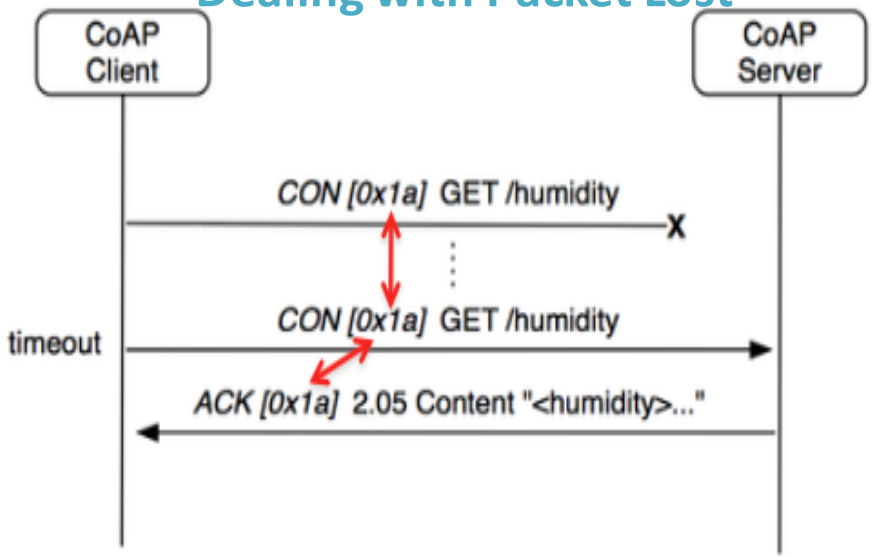
1. IoT Communications Protocols CoAP

CoAP – Interaction Model

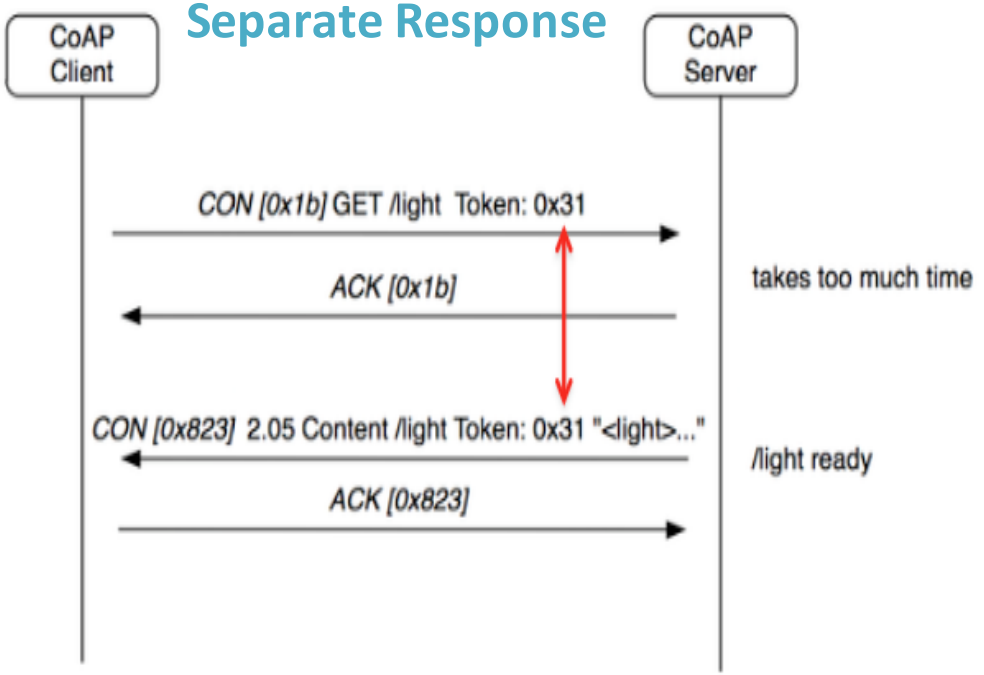
Request Sample



Dealing with Packet Lost



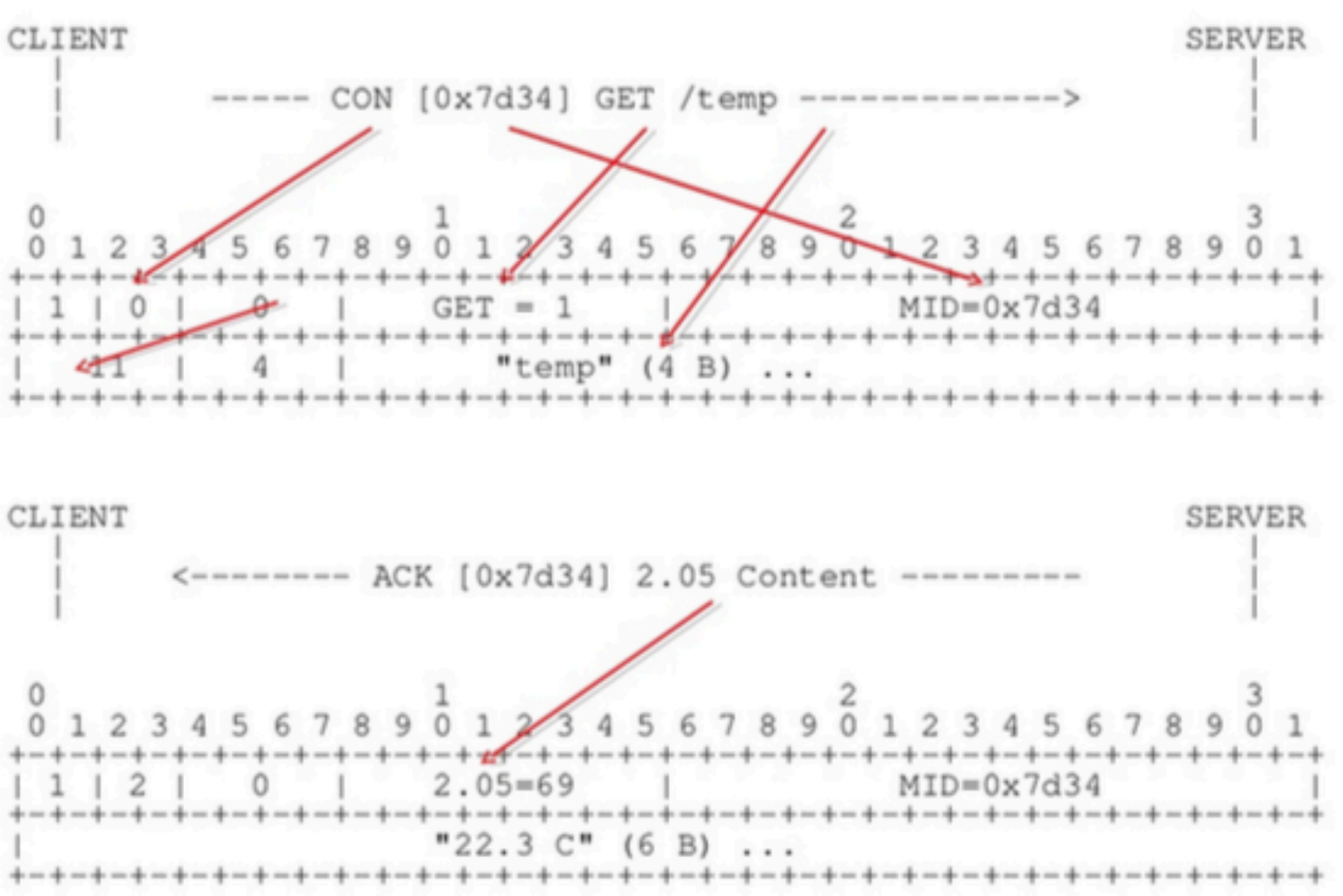
Separate Response



Copyright: https://community.arm.com/servlet/JiveServlet/downloadBody/8633-102-2-15471/ARM%20CoAP%20Tutorial%20April%2030%202014.pdf

1. IoT Communications Protocols CoAP

Bits and bytes...



Copyright: https://community.arm.com/servlet/JiveServlet/downloadBody/8633-102-2-15471/ARM%20CoAP%20Tutorial%20April%2030%202014.pdf

1. IoT Communications Protocols CoAP

CoAP – Other Features

- Caching

- CoAP supports caching of responses to efficiently fulfill requests. Simple Caches is particularly useful in constrained networks for several reasons, including traffic limiting, performance improving, resources accessing times and security.

- Resource Discovery

- CoAP Multicast: "All CoAP Nodes"

GET ./well-known/core

Copyright: http://www.inginfpoliba.eu/index.php?action=downloadfile&filename=8-Application%20Layer%20for%20IoT.pdf&directory=Interet%20of%20Things

1. IoT Communications Protocols CoAP

CoAP – COnstrained Application Protocol getting started

- There are many open source implementations available:
  - mbed includes CoAP support
  - Java CoAP Library Californium
  - C CoAP Library Erbium
  - libCoAP C Library
  - jCoAP Java Library
  - OpenCoAP C Library
  - TinyOS and Contiki include CoAP support

- CoAP is already part of many commercial products/systems
  - ARM Sensinode NanoService
  - RTX 4100 WiFi Module

- Firefox has a CoAP plugin called Copper
- Wireshark has CoAP dissector support
- Implement CoAP yourself, it is not that hard! – if time available!

1. IoT Communications Protocols MQTT

Message Queueing Telemetry Transport – MQTT – Default TCP Port 1883

1998, Dave Locke & Ian Craggs, IBM. From March, 2013, start of standardization process at OASIS. Now v3.1 (2013)

http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html

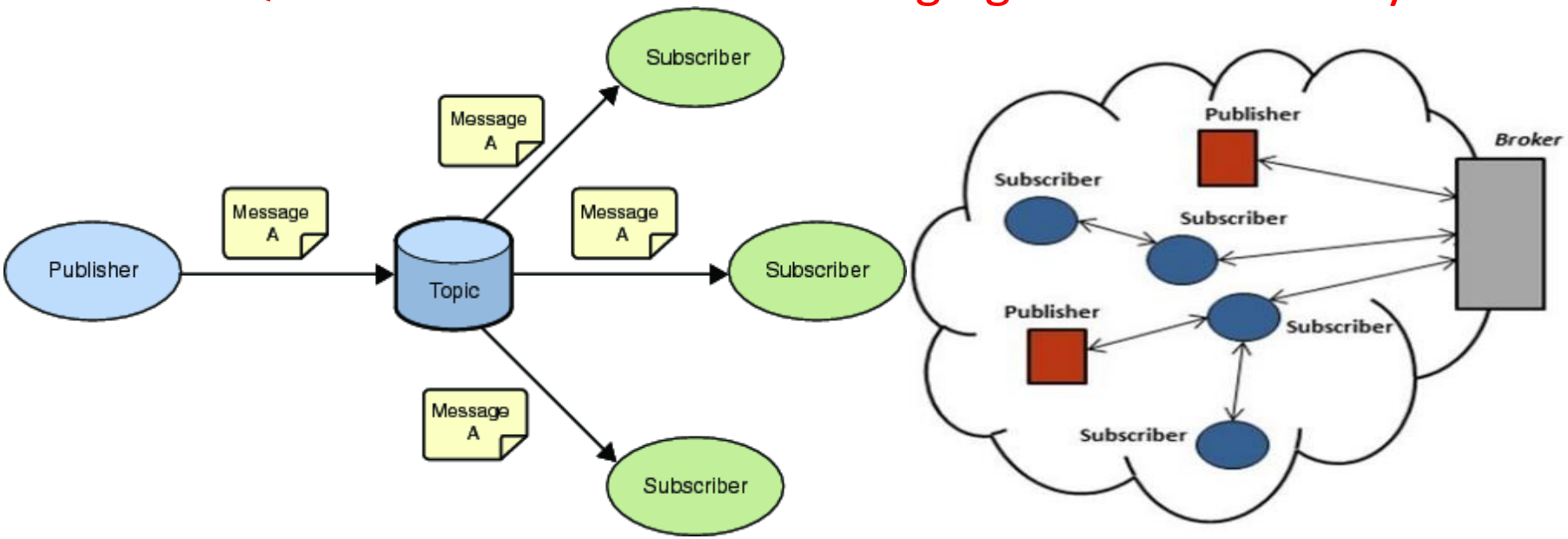
Main Features:

- Publish / Subscribe message pattern → one-to-many messaging distribution, applications decoupling;
- Message transport payload-agnostic;
- Assumes the use of the TCP/IP protocol stack;
- 3 QoS Levels: At Most Once, At Least Once, Exactly Once;
- Small Transport Overhead, minimal messages exchanges;
- Will Mechanism, to indicate to the other part an abnormal disconnection

Copyright: http://www.inginfpoliba.eu/index.php?action=downloadfile&filename=8-Application%20Layer%20for%20IoT.pdf&directory=Interet%20of%20Things

1. IoT Communications Protocols MQTT

MQTT - Publish Subscribe Messaging aka One to Many



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

A producer sends (publishes) a message (publication) on a topic (subject)
A consumer subscribes (makes a subscription) for messages on a topic (subject)
A topic is managed within a MQTT Broker

A message server / broker matches publications to subscriptions
- If no matches the message is discarded
- If one or more matches the message is delivered to each matching subscriber/consumer

Copyright: https://www.oasis-open.org/committees/download.php/48647/MQTT%20Contribution.ppt

MQTT

MQ TELEMETRY TRANSPORT

AN INTRODUCTION TO MQTT, A PROTOCOL FOR M2M AND IoT APPLICATIONS

Peter R. Egli
INDIGOO.COM

Rev. 1.80

Copyright: http://www.indigoo.com/dox/wsmw/1_Middleware/MQTT.pdf

Contents

© Peter R. Egli 2015

1. What is MQTT?

MQTT is a lightweight message queueing and transport protocol.
MQTT, as its name implies, is suited for the transport of telemetry data (sensor and actor data).

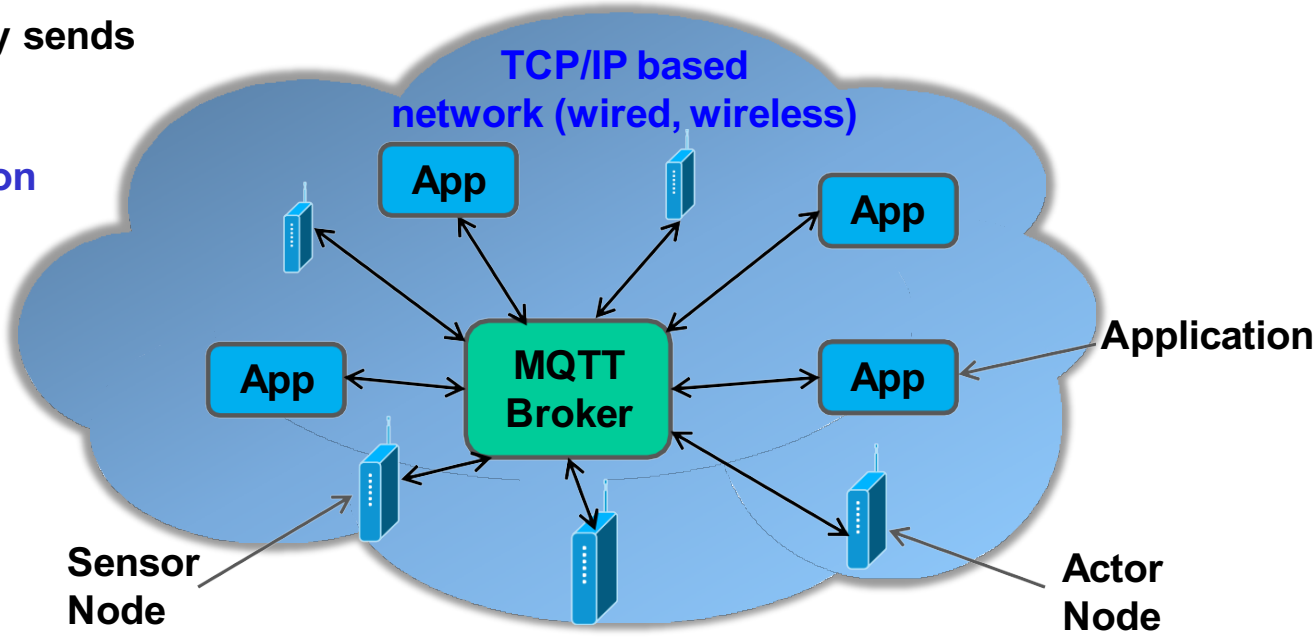
MQTT is very lightweight and thus suited for M2M (Mobile to Mobile), WSN (Wireless Sensor Networks) and ultimately IoT (Internet of Things) scenarios where sensor and actor nodes communicate with applications through the MQTT message broker.

Example:

Light sensor continuously sends sensor data to the broker.

Building control application receives sensor data from the broker and decides to activate the blinds.

Application sends a blind activation message to the blind actor node through the broker.

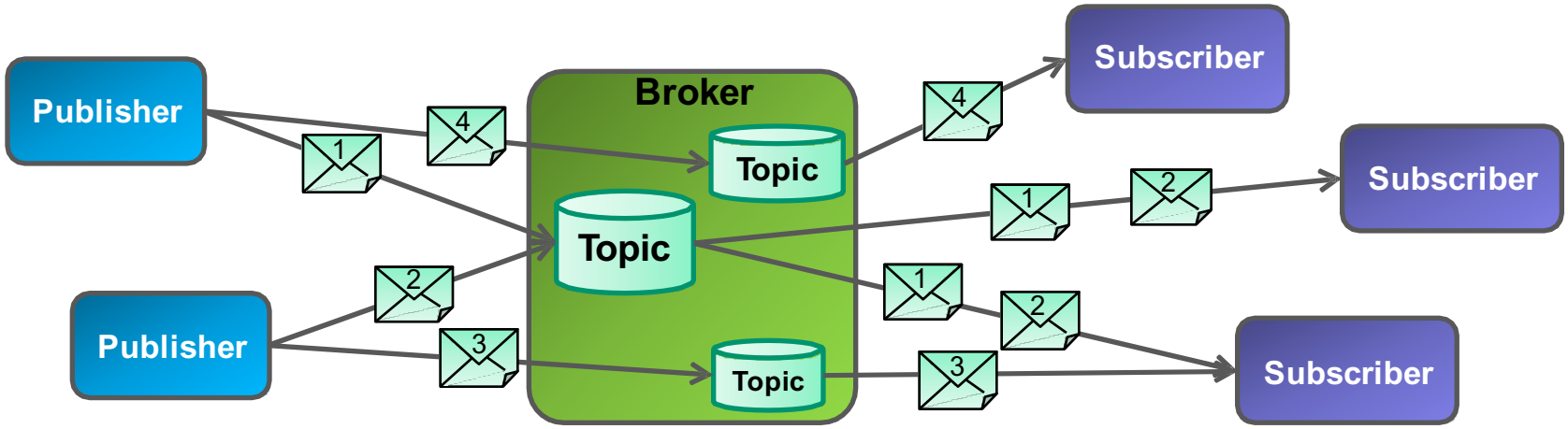


© Peter R. Egli 2015

2. MQTT characteristics

MQTT Key features:

- Lightweight message queueing and transport protocol
- Asynchronous communication model with messages (events)
- Low overhead (2 bytes header) for low network bandwidth applications
- Publish / Subscribe (PubSub) model
- Decoupling of data producer (publisher) and data consumer (subscriber) through topics (message queues)
- Simple protocol, aimed at low complexity, low power and low footprint implementations (e.g. WSN - Wireless Sensor Networks)
- Runs on connection-oriented transport (TCP). To be used in conjunction with 6LoWPAN (TCP header compression)
- MQTT caters for (wireless) network disruptions



© Peter R. Egli 2015

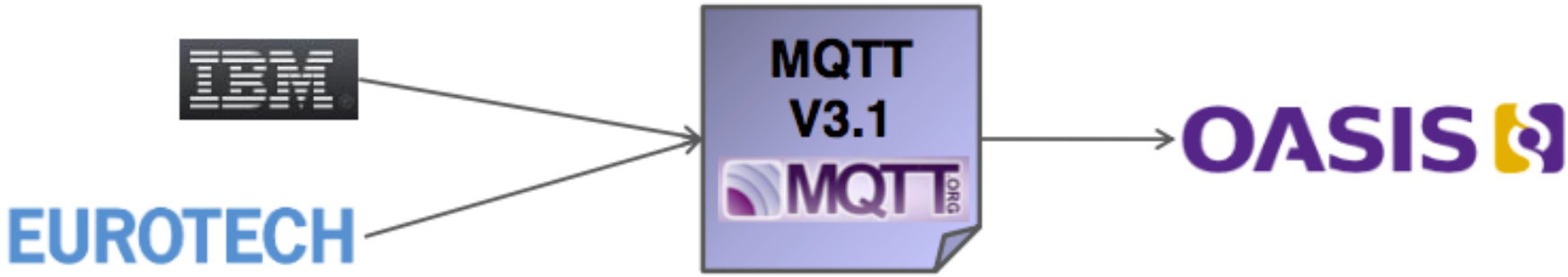
1. IoT Communications Protocols MQTT

- The past, present and future of MQTT:
  - MQTT was developed by IBM and Eurotech.
  - The current version 3.1 is available from http://mqtt.org/

Eventually, MQTT version 3.1 is to be adopted and published as an official standard by OASIS (process ongoing).
As such, OASIS becomes the new home for the development of MQTT.

The OASIS TC (Technical Committee) tasked with the further development of MQTT commits to the following:

- Backward compatibility of forthcoming OASIS MQTT standard with MQTT V3.1
- Changes restricted to the CONNECT messsage
- Clarification of existing version V3.1 (mostly editorial changes)

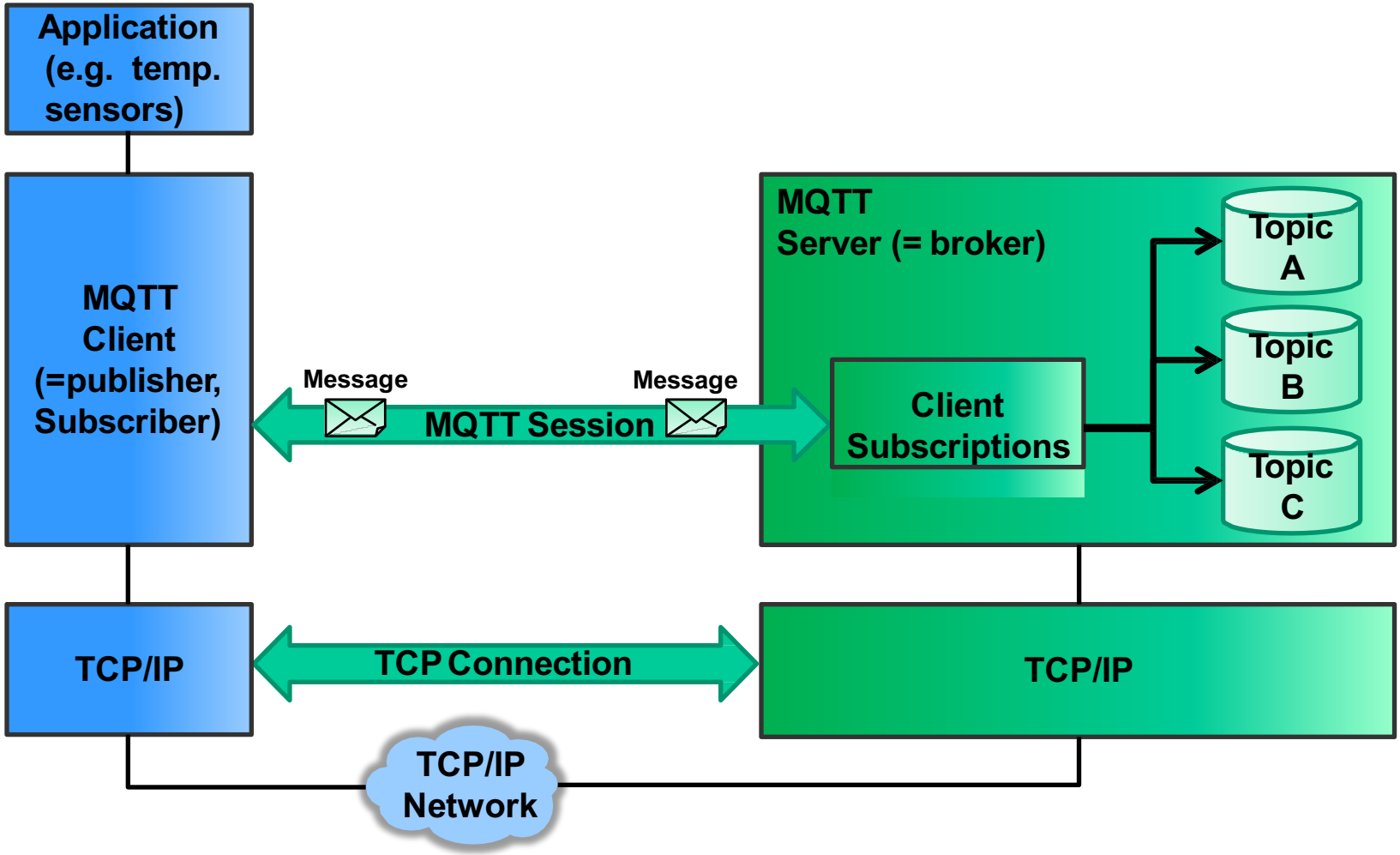


© Peter R. Egli 2015

Copyright: http://www.indigoo.com/dox/wsmw/1_Middleware/MQTT.pdf

4. MQTT model (1/3)

The core elements of MQTT are clients, servers (=brokers), sessions, subscriptions and topics.

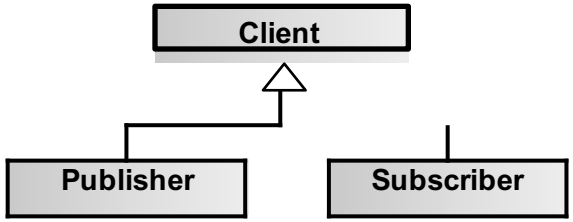


© Peter R. Egli 2015

4. MQTT model (2/3)

MQTT client (=publisher, subscriber):

Clients subscribe to topics to publish and receive messages.
Thus subscriber and publisher are special roles of a client.



MQTT server (=broker):

Servers run topics, i.e. receive subscriptions from clients on topics, receive messages from clients and forward these, based on client's subscriptions, to interested clients.

Topic:

Technically, topics are message queues. Topics support the publish/subscribe pattern for clients.
Logically, topics allow clients to exchange information with defined semantics.
Example topic: Temperature sensor data of a building.

© Peter R. Egli 2015

4. MQTT model (3/3)

Session:

A session identifies a (possibly temporary) attachment of a client to a server. All communication between client and server takes place as part of a session.

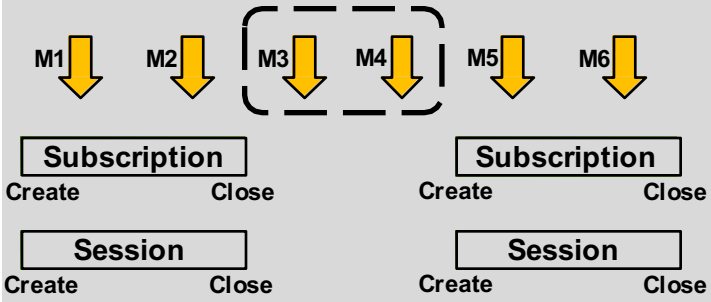
Subscription:

Unlike sessions, a subscription logically attaches a client to a topic. When subscribed to a topic, a client can exchange messages with a topic.

Subscriptions can be «transient» or «durable», depending on the clean session flag in the CONNECT message:

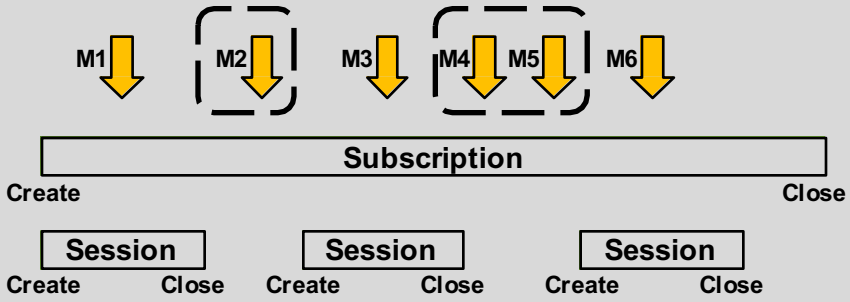
«Transient» subscription ends with session:

Messages M3 and M4 are not received by the client



«Durable» subscription:

Messages M2, M4 and M5 are not lost but will be received by the client as soon as it creates / opens a new session.



Message:

Messages are the units of data exchange between topic clients.

MQTT is agnostic to the internal structure of messages.

© Peter R. Egli 2015

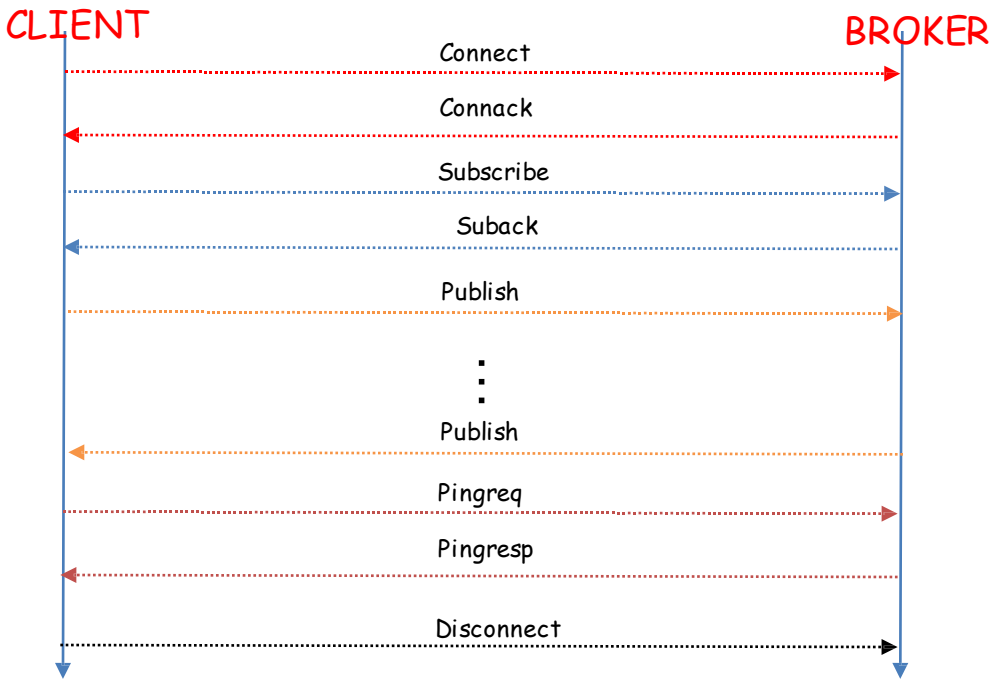
IoT Communications Protocols MQTT

MQTT – Message Type and 2 bytes Mandatory Header
Client – Broker General Messages Flow

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

Message Type

1 CONNECT	8 SUBSCRIBE
2 CONNACK	9 SUBACK
3 PUBLISH	10 UNSUBSCRIBE
4 PUBACK	11 UNSUBACK
5 PUBREC	12 PINGREQ
6 PUBREL	13 PINGRESP
7 PUBCOMP	14 DISCONNECT



Copyright: http://www.inginfpoliba.eu/index.php?action=downloadfile&filename=8-Application%20Layer%20for%20IoT.pdf&directory=Interet%20of%20Things

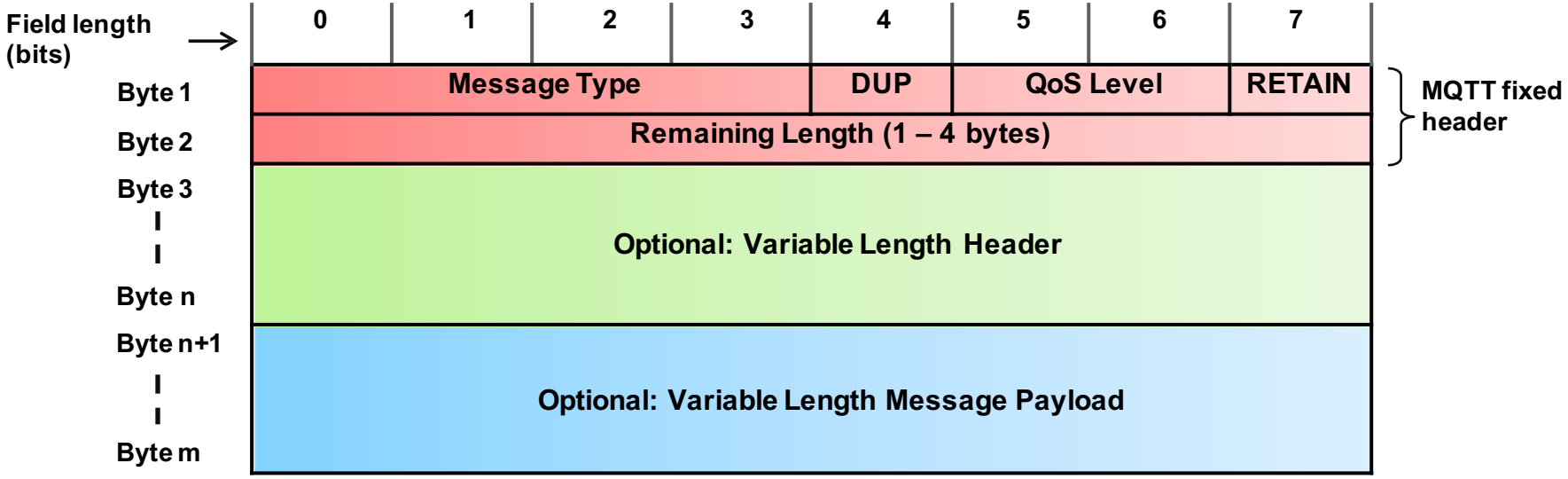
5. MQTT message format (1/14)

Message format:

MQTT messages contain a mandatory fixed-length header (2 bytes) and an optional message-specific variable length header and message payload.

Optional fields usually complicate protocol processing.
However, MQTT is optimized for bandwidth constrained and unreliable networks (typically wireless networks), so optional fields are used to reduce data transmissions as much as possible.

MQTT uses network byte and bit ordering.



© Peter R. Egli 2015

5. MQTT message format (2/14)

Overview of fixed header fields:

Message fixed header field	Description / Values	
Message Type	0: Reserved	8: SUBSCRIBE
	1: CONNECT	9: SUBACK
	2: CONNACK	10: UNSUBSCRIBE
	3: PUBLISH	11: UNSUBACK
	4: PUBACK	12: PINGREQ
	5: PUBREC	13: PINGRESP
	6: PUBREL	14: DISCONNECT
	7: PUBCOMP	15: Reserved
DUP	Duplicate message flag. Indicates to the receiver that this message may have already been received. 1: Client or server (broker) re-delivers a PUBLISH, PUBREL, SUBSCRIBE or UNSUBSCRIBE message (duplicate message).	
QoS Level	Indicates the level of delivery assurance of a PUBLISH message. 0: At-most-once delivery, no guarantees, «Fire and Forget». 1: At-least-once delivery, acknowledged delivery. 2: Exactly-once delivery. Further details see MQTT QoS.	
RETAIN	1: Instructs the server to retain the last received PUBLISH message and deliver it as a first message to new subscriptions. Further details see RETAIN (keep last message).	
Remaining Length	Indicates the number of remaining bytes in the message, i.e. the length of the (optional) variable length header and (optional) payload. Further details see Remaining length (RL).	

© Peter R. Egli 2015

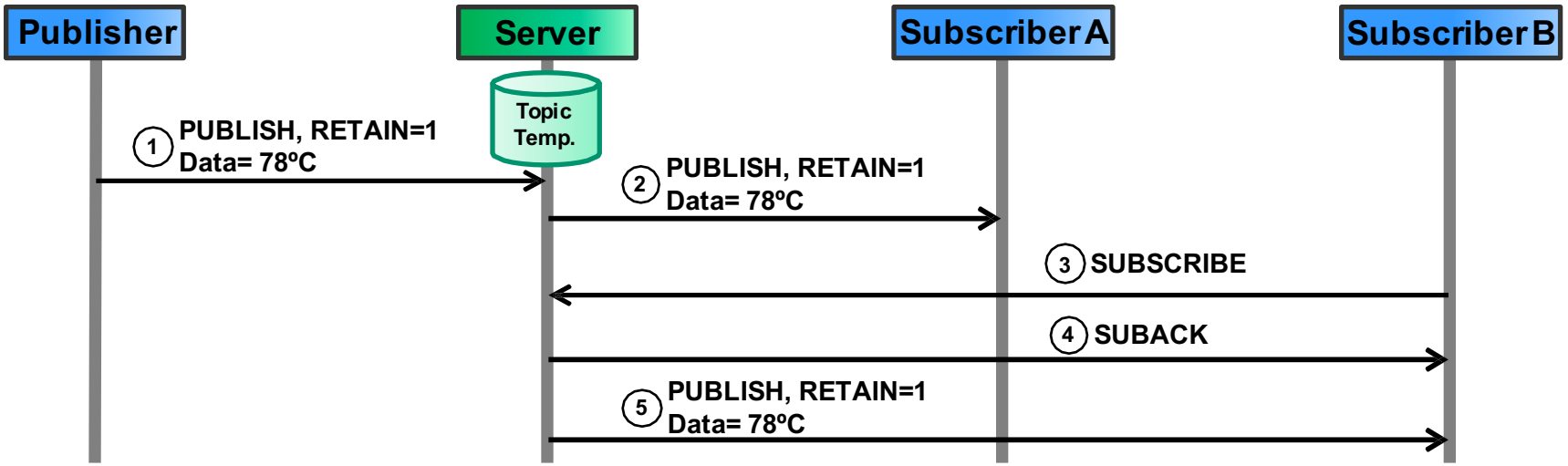
5. MQTT message format (3/14)

RETAIN (keep last message):

RETAIN=1 in a PUBLISH message instructs the server to keep the message for this topic. When a new client subscribes to the topic, the server sends the retained message.

Typical application scenarios:
Clients publish only changes in data, so subscribers receive the last known good value.

Example:
Subscribers receive last known temperature value from the temperature data topic.
RETAIN=1 indicates to subscriber B that the message may have been published some time ago.



© Peter R. Egli 2015

5. MQTT message format (4/14)

Remaining length (RL):

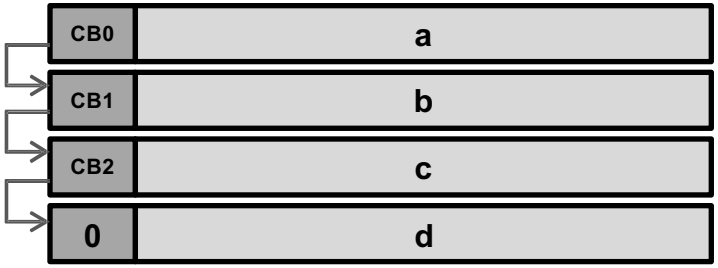
The remaining length field encodes the sum of the lengths of:

a. (Optional) variable length header

b. (Optional) payload

To save bits, remaining length is a variable length field with 1…4 bytes.
The most significant bit of a length field byte has the meaning «continuation bit» (CB). If more bytes follow, it is set to 1.

Remaining length is encoded as a * 128^0 + b * 128^1 + c * 128^2 + d * 128^3 and placed into the RL field bytes as follows:



Byte 0 = LSB (a * 128^0, CB0=1 if b > 0)

Byte 1 (b * 128^1, CB1=1 if c > 0)

Byte 2 (c * 128^2, CB2=1 if d > 0)

Byte 3 = MSB (d * 128^3)

Key:
LSB: Least Significant Byte
MSB: Most Significant Byte

Example 1: **RL = 364 = 108*128^0+2*128^1 → a=108, CB0=1, b=2, CB1=0, c=0, d=0, CB2=0**

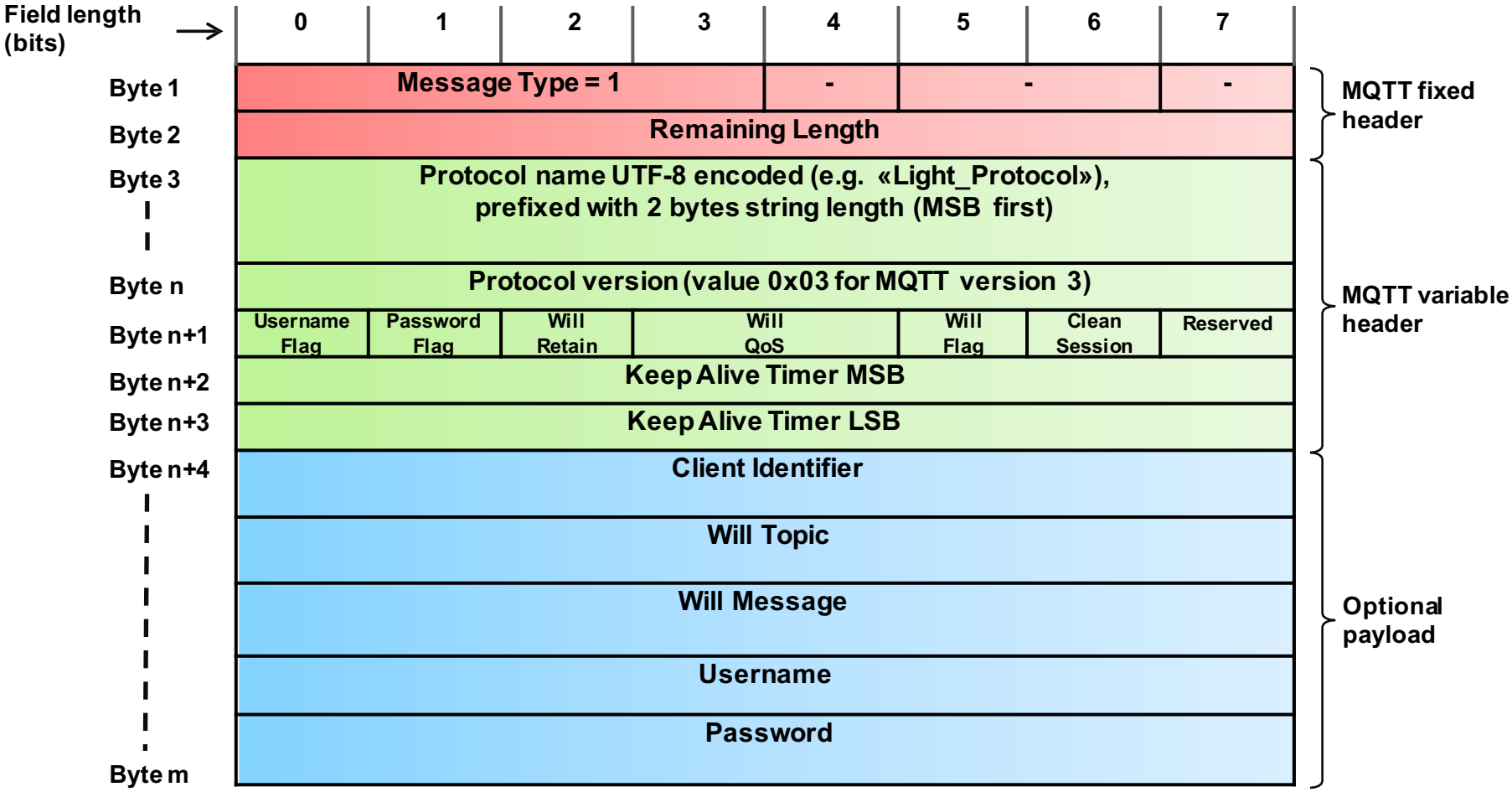
Example 2: **RL = 25'897 = 41*128^0 + 74*128^1 + 1*128^2 → a=41, CB0=1, b=74, CB1=1, c=1, CB2=0, d=0**

© Peter R. Egli 2015

5. MQTT message format (5/14)

CONNECT message format:

The CONNECT message contains many session-related information as optional header fields.



5. MQTT message format (6/14)

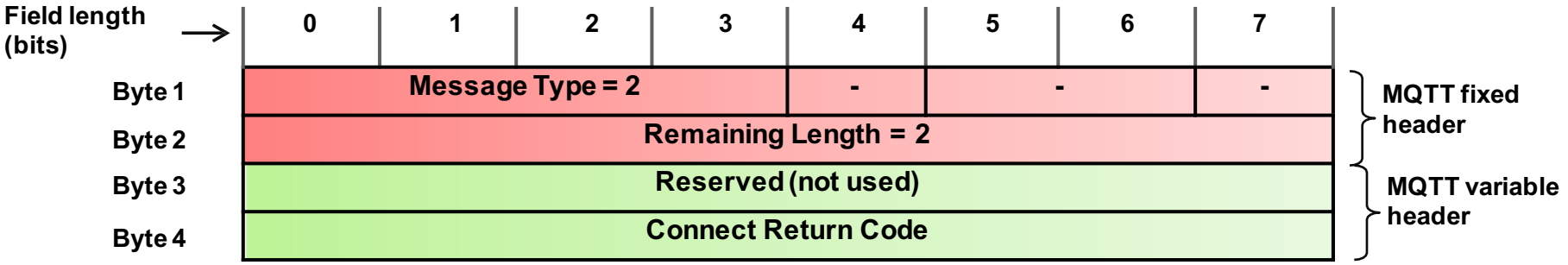
Overview CONNECT message fields:

CONNECT message field	Description / Values
Protocol Name	UTF-8 encoded protocol name string. Example: «Light_Protocol»
Protocol Version	Value 3 for MQTT V3.
Username Flag	If set to 1 indicates that payload contains a username.
Password Flag	If set to 1 indicates that payload contains a password. If username flag is set, password flag and password must be set as well.
Will Retain	If set to 1 indicates to server that it should retain a Will message for the client which is published in case the client disconnects unexpectedly.
Will QoS	Specifies the QoS level for a Will message.
Will Flag	Indicates that the message contains a Will message in the payload along with Will retain and Will QoS flags. More details see MQTT will message.
Clean Session	If set to 1, the server discards any previous information about the (re)-connecting client (clean new session). If set to 0, the server keeps the subscriptions of a disconnecting client including storing QoS level 1 and 2 messages for this client. When the client reconnects, the server publishes the stored messages to the client.
Keep Alive Timer	Used by the server to detect broken connections to the client. More details see Keepalive timer.
Client Identifier	The client identifier (between 1 and 23 characters)uniquely identifies the client to the server. The client identifier must be unique across all clients connecting to a server.
Will Topic	Will topic to which a will message is published if the will flag is set.
Will Message	Will message to be puslished if will flag is set.
Username and Password	Username and password if the corresponding flags are set.

© Peter R. Egli 2015

5. MQTT message format (7/14)

CONNACK message format:

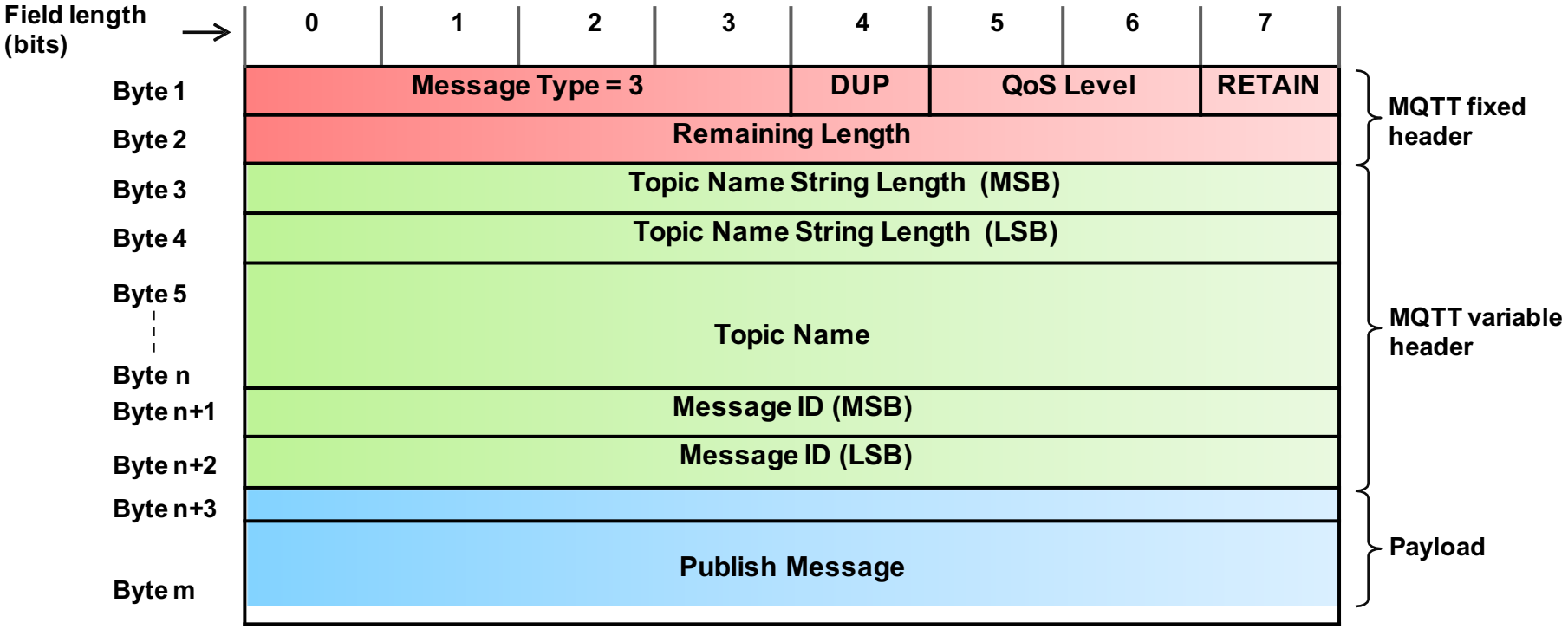


CONNACK message field	Description / Values
Reserved	Reserved field for future use.
Connect Return Code	0: Connection Accepted 1: Connection Refused, reason = unacceptable protocol version 2: Connection Refused, reason = identifier rejected 3: Connection Refused, reason = server unavailable 4: Connection Refused, reason = bad user name or password 5: Connection Refused, reason = not authorized 6-255: Reserved for future use

© Peter R. Egli 2015

5. MQTT message format (8/14)

PUBLISH message format:

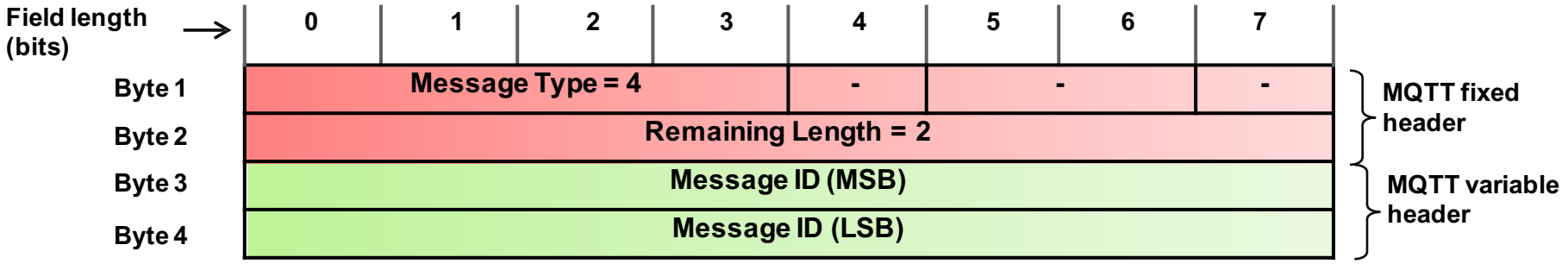


PUBLISH message field	Description / Values
Topic Name with Topic Name String Length	Name of topic to which the message is published. The first 2 bytes of the topic name field indicate the topic name string length.
Message ID	A message ID is present if QoS is 1 (At-least-once delivery, acknowledged delivery) or 2 (Exactly-once delivery).
Publish Message	Message as an array of bytes. The structure of the publish message is application-specific.

© Peter R. Egli 2015

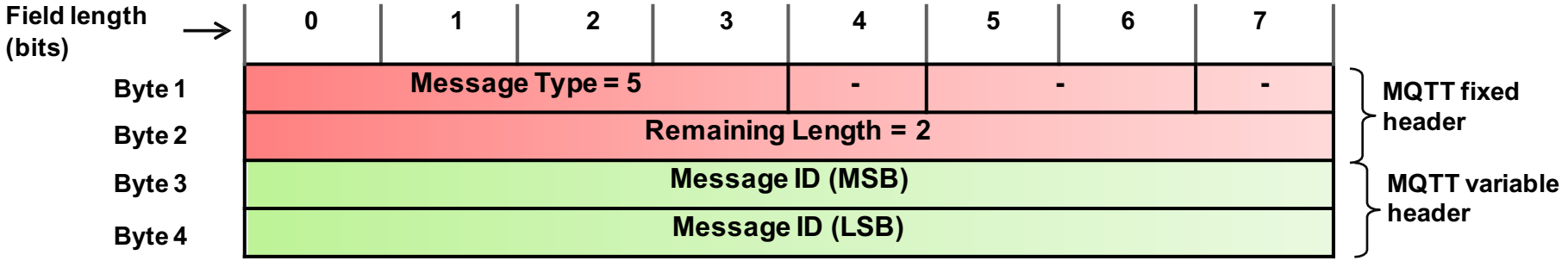
5. MQTT message format (9/14)

PUBACK message format:



PUBACK message field	Description / Values
Message ID	The message ID of the PUBLISH message to be acknowledged.

PUBREC message format:

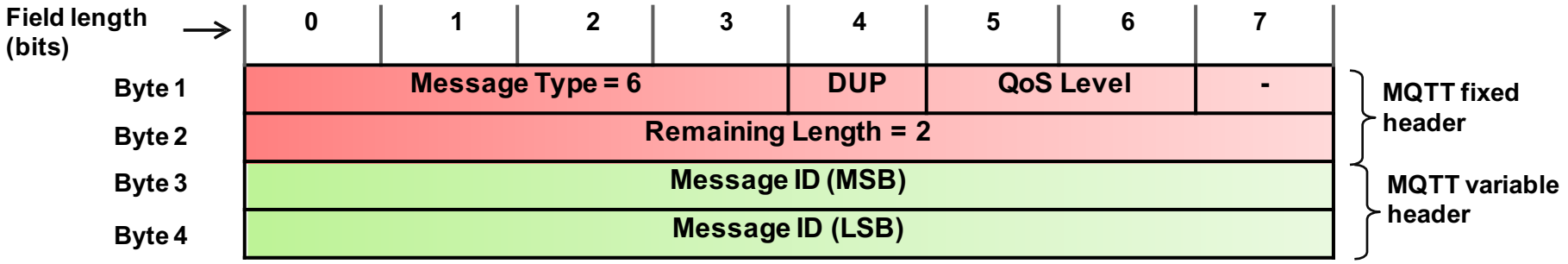


PUBREC message field	Description / Values
Message ID	The message ID of the PUBLISH message to be acknowledged.

© Peter R. Egli 2015

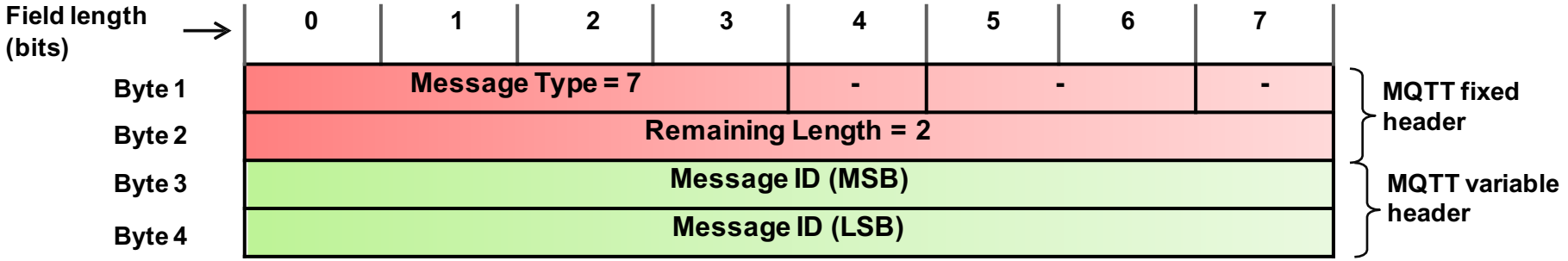
5. MQTT message format (10/14)

PUBREL message format:



PUBREL message field	Description / Values
Message ID	The message ID of the PUBLISH message to be acknowledged.

PUBCOMP message format:

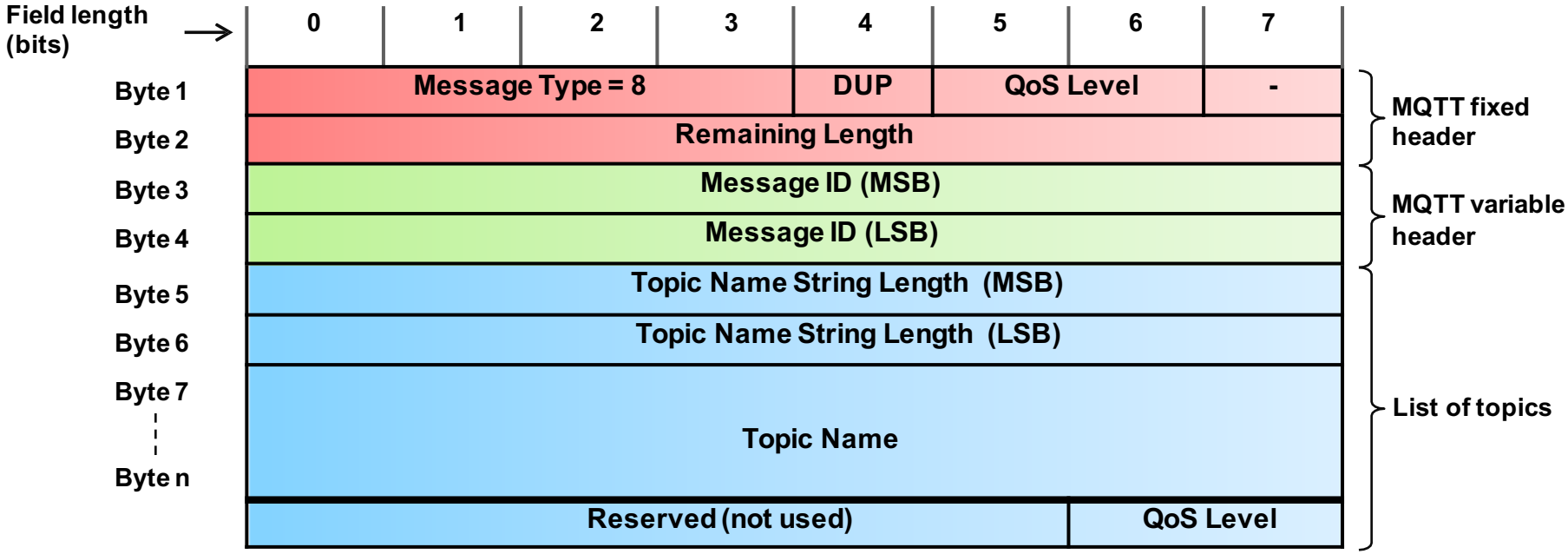


PUBCOMP message field	Description / Values
Message ID	The message ID of the PUBLISH message to be acknowledged.

© Peter R. Egli 2015

5. MQTT message format (11/14)

SUBSCRIBE message format:

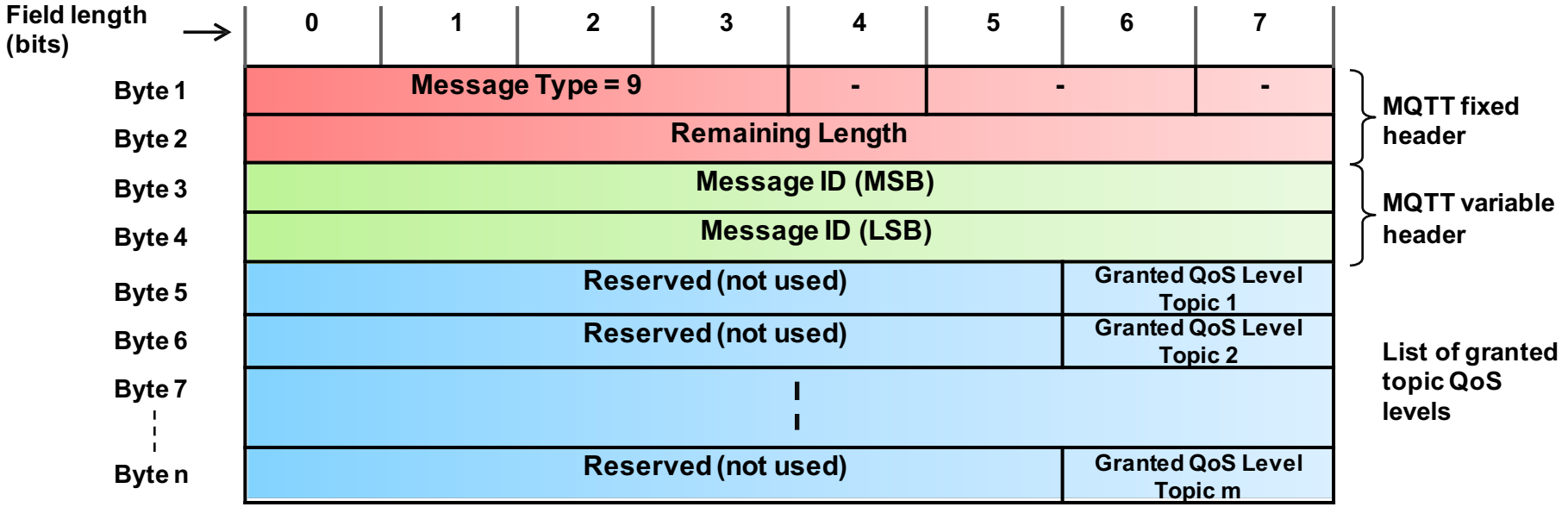


SUBSCRIBE message field	Description / Values
Message ID	The message ID field is used for acknowledgment of the SUBSCRIBE message since these have a QoS level of 1.
Topic Name with Topic Name String Length	Name of topic to which the client subscribes. The first 2 bytes of the topic name field indicate the topic name string length. Topic name strings can contain wildcard characters as explained under Topic wildcards. Multiple topic names along with their requested QoS level may appear in a SUBSCRIBE message.
QoS Level	QoS level at which the clients wants to receive messages from the given topic.

© Peter R. Egli 2015

5. MQTT message format (12/14)

SUBACK message format:

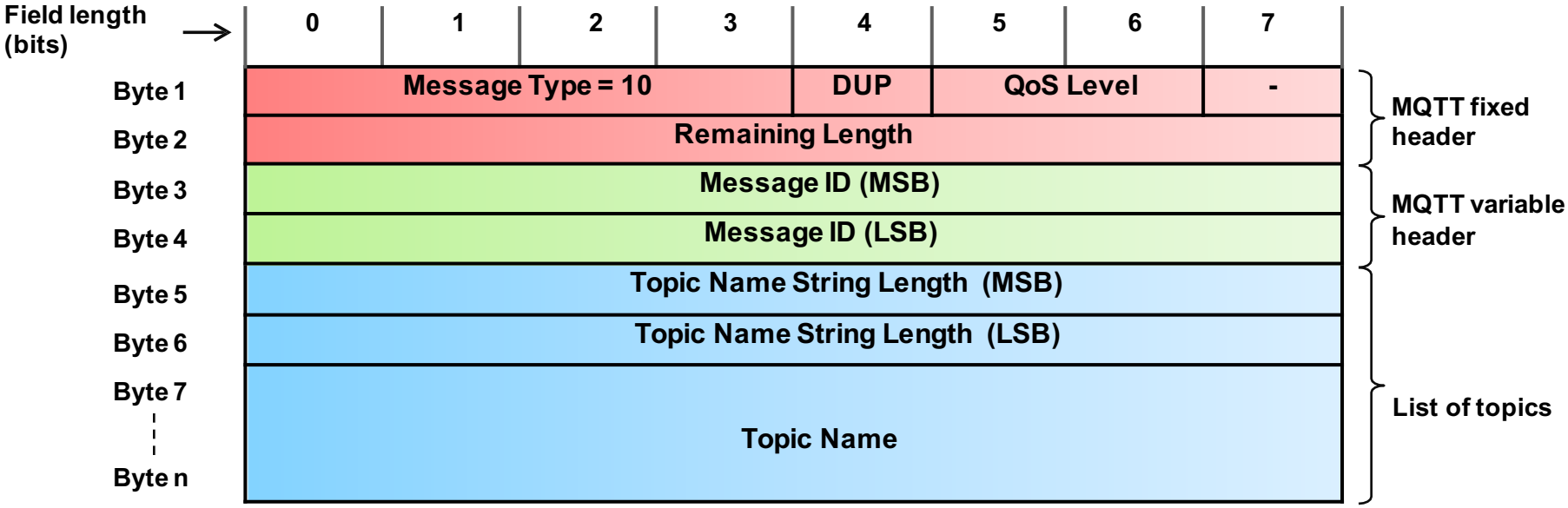


SUBACK message field	Description / Values
Message ID	Message ID of the SUBSCRIBE message to be acknowledged.
Granted QoS Level for Topic	List of granted QoS levels for the topics list from the SUBSCRIBE message.

© Peter R. Egli 2015

5. MQTT message format (13/14)

UNSUBSCRIBE message format:

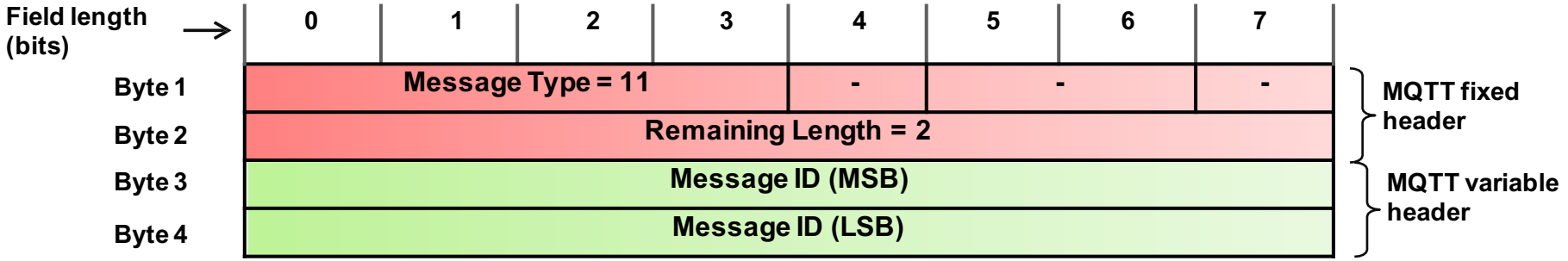


UNSUBSCRIBE message field	Description / Values
Message ID	The message ID field is used for acknowledgment of the UNSUBSCRIBE message (UNSUBSCRIBE messages have a QoS level of 1).
Topic Name with Topic Name String Length	Name of topic from which the client wants to unsubscribe. The first 2 bytes of the topic name field indicate the topic name string length. Topic name strings can contain wildcard characters as explained under Topic wildcards. Multiple topic names may appear in an UNSUBSCRIBE message.

© Peter R. Egli 2015

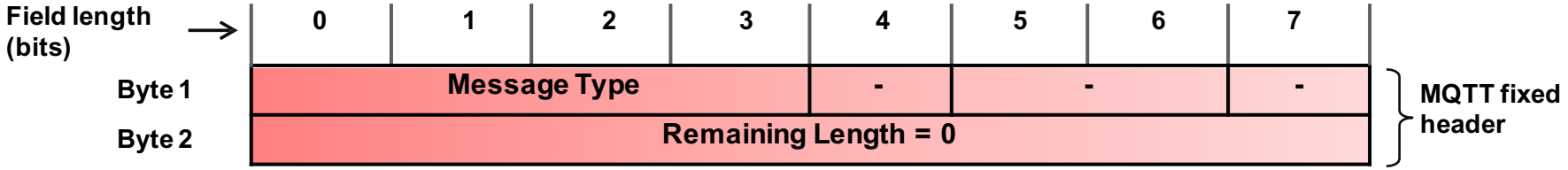
5. MQTT message format (14/14)

UNSUBACK message format:



UNSUBACK message field	Description / Values
Message ID	The message ID of the UNSUBSCRIBE message to be acknowledged.

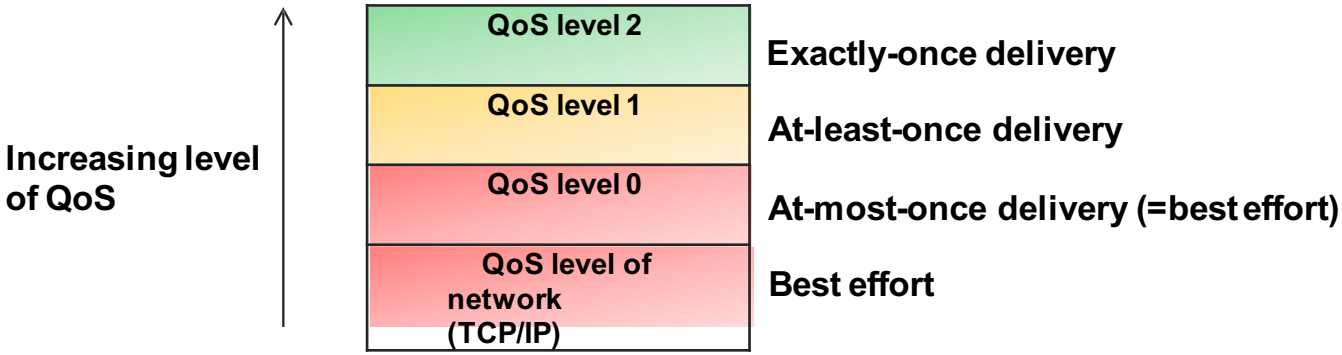
DISCONNECT, PINGREQ, PINGRESP message formats:



© Peter R. Egli 2015

6. MQTT QoS (1/Linux

MQTT provides the typical delivery quality of service (QoS) levels of message oriented middleware.
Even though TCP/IP provides guaranteed data delivery, data loss can still occur if a TCP connection breaks down and messages in transit are lost.
Therefore MQTT adds 3 quality of service levels on top of TCP.



QoS level 0:
At-most-once delivery («best effort»).
Messages are delivered according to the delivery guarantees of the underlying network (TCP/IP).
Example application: Temperature sensor data which is regularly published. Loss of an individual value is not critical since applications (consumers of the data) will anyway integrate the values over time and loss of individual samples is not relevant.

© Peter R. Egli 2015

6. MQTT QoS (2/Linux

QoS level 1:

At-lest-once delivery. Messages are guaranteed to arrive, but there may be duplicates. Example application: A door sensor senses the door state. It is important that door state changes (closed→open, open→closed) are published losslessly to subscribers (e.g. alarming function). Applications simply discard duplicate messages by evaluating the message ID field.

QoS level 2:

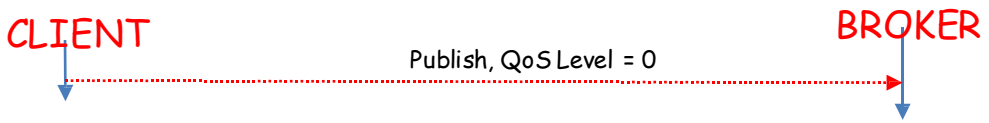
Exactly-once delivery.

This is the highest level that also incurs most overhead in terms of control messages and the need for locally storing the messages.

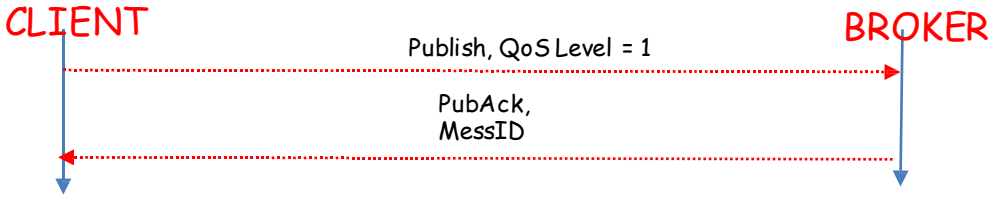
Exactly-once is a combination of at-least-once and at-most-once delivery guarantee.

Example application: Applications where duplicate events could lead to incorrect actions, e.g. sounding an alarm as a reaction to an event received by a message.

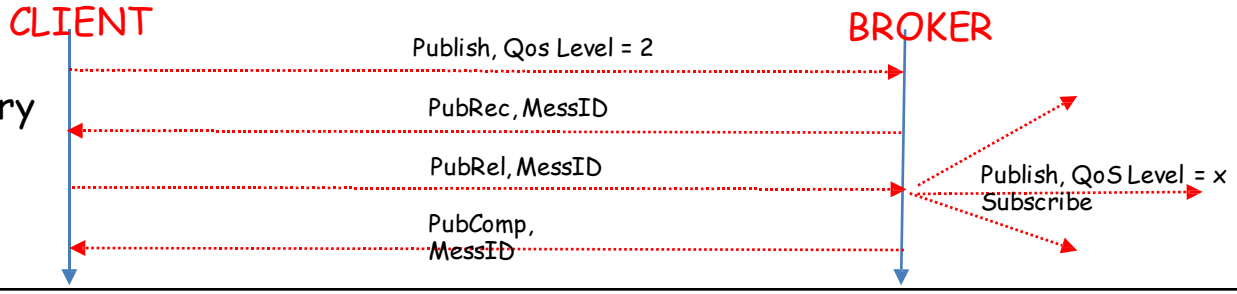
QoS Level = 0 ----- At Most Once Delivery



QoS Level = 1 ----- At Least Once Delivery



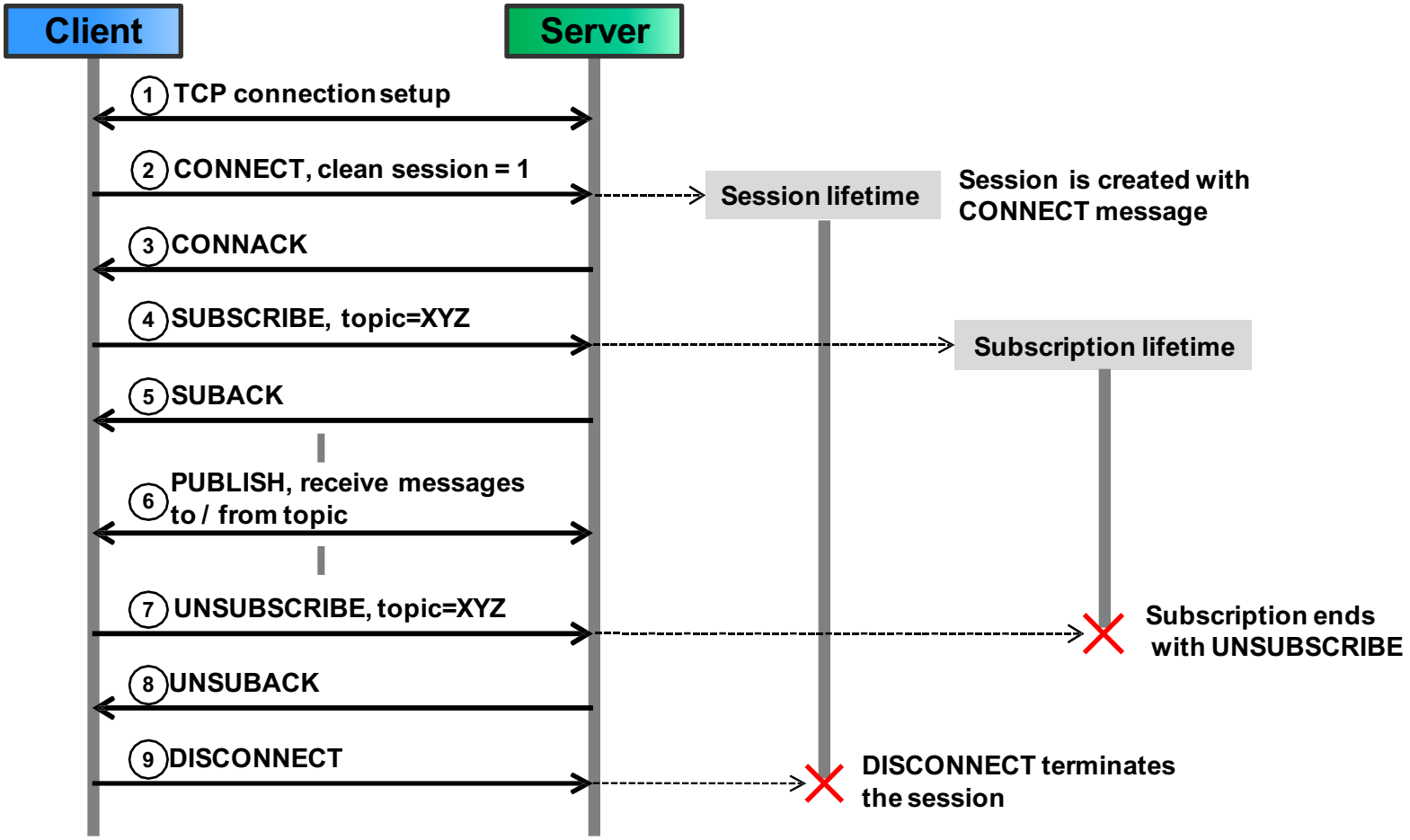
QoS Level = 2 --- Exactly Once Delivery



© Peter R. Egli 2015

7. CONNECT and SUBSCRIBE message sequence (1/Linux

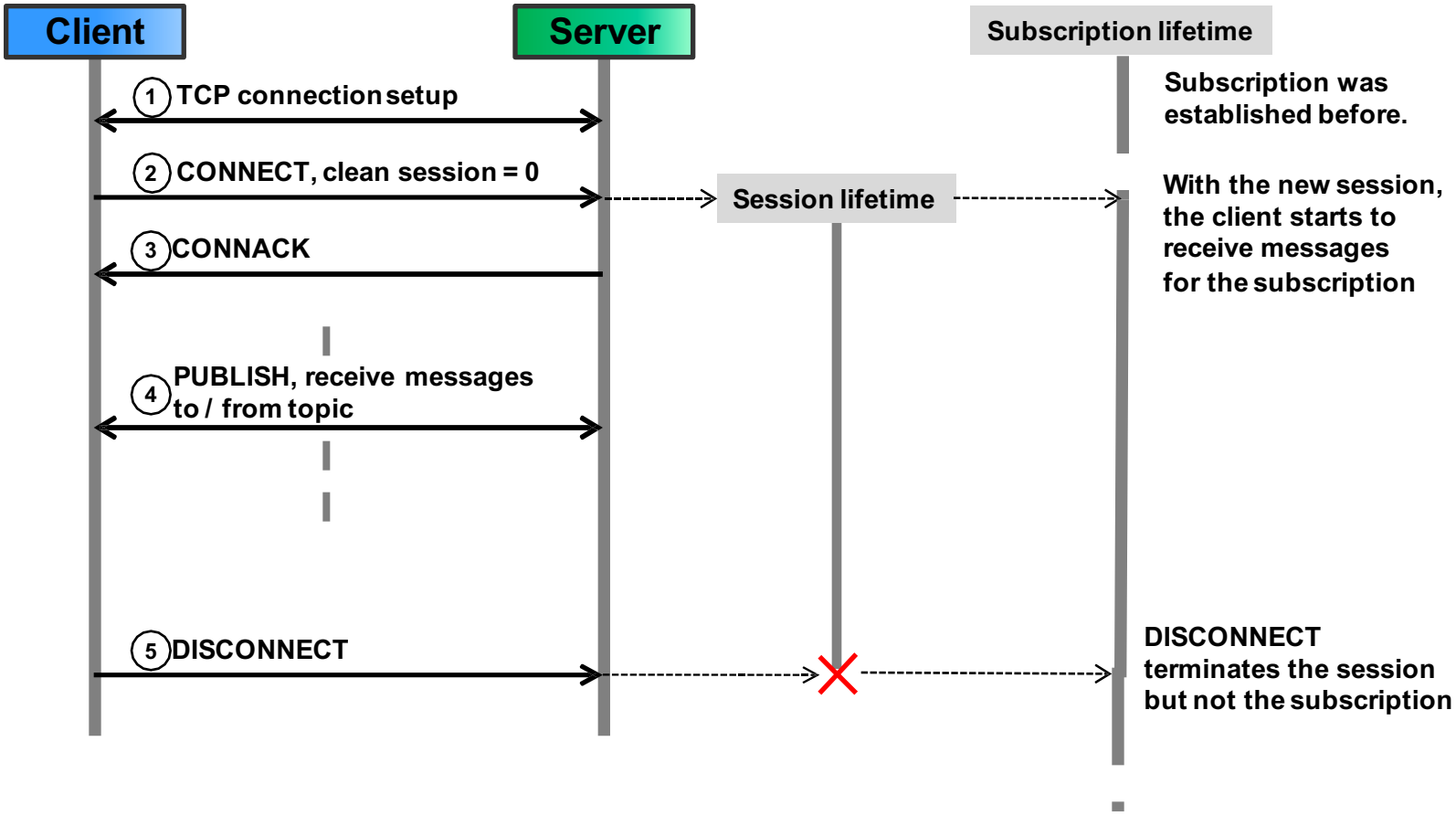
Case 1: Session and subscription setup with clean session flag = 1 («transient» subscription)



© Peter R. Egli 2015

7. CONNECT and SUBSCRIBE message sequence (2/Linux

Case 2: Session and subscription setup with clean session flag = 0 («durable» subscription)

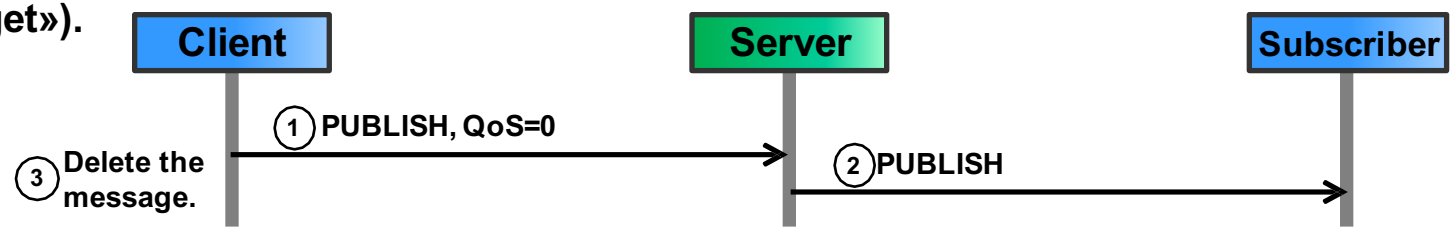


© Peter R. Egli 2015

8. PUBLISH message flows (1/Linux

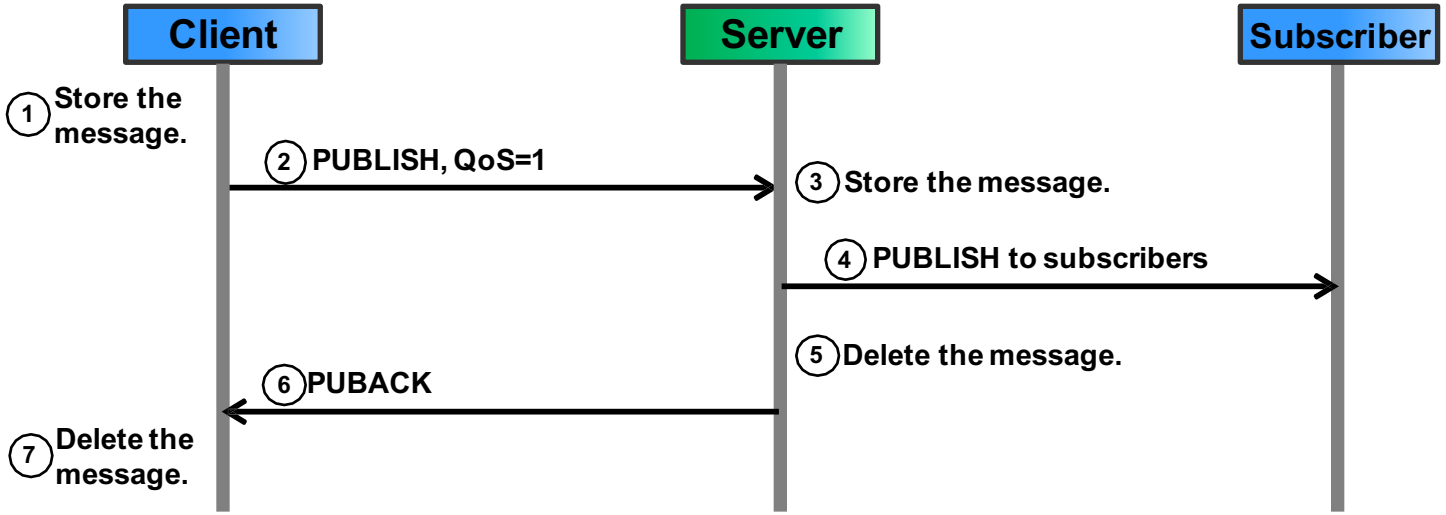
QoS level 0:

With QoS level 0, a message is delivered with at-most-once delivery semantics («fire-and-forget»).



QoS level 1:

QoS level 1 affords at-least-once delivery semantics. If the client does not receive the PUBACK in time, it re-sends the message.

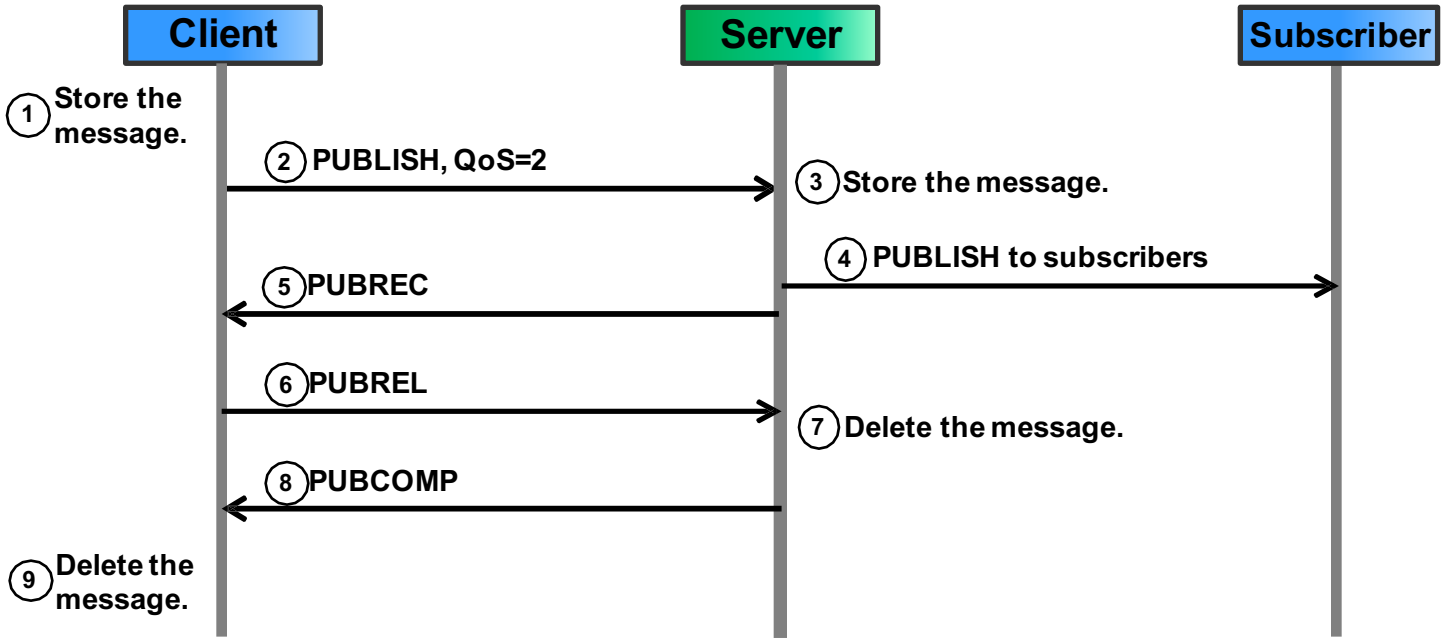


© Peter R. Egli 2015

8. PUBLISH message flows (2/Linux

QoS level 2:

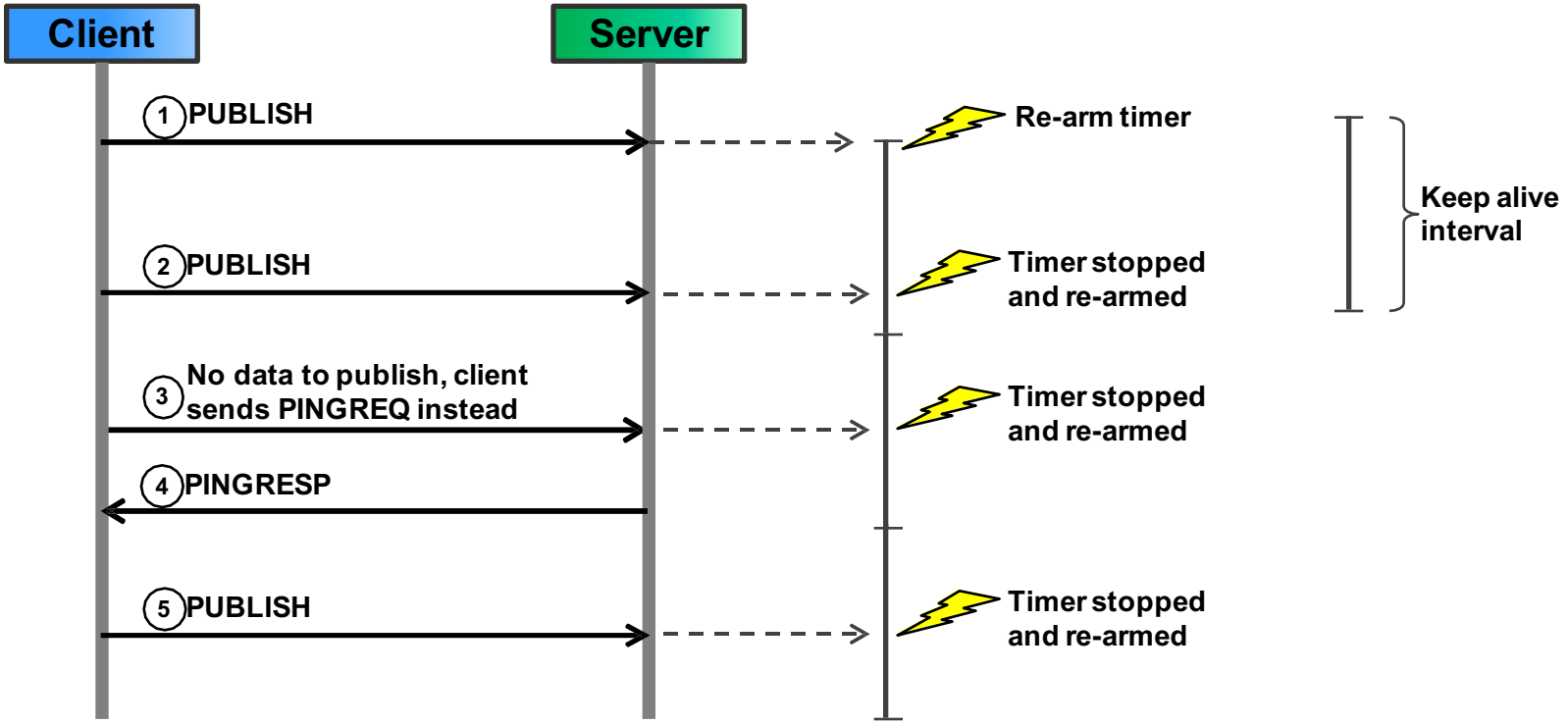
QoS level 2 affords the highest quality delivery semantics exactly-once, but comes with the cost of additional control messages.



© Peter R. Egli 2015

9. Keep alive timer, breath of live with PINGREQ

The keep alive timer defines the maximum allowable time interval between client messages.
The timer is used by the server to check client's connection status.
After 1.5 * keepalive-time is elapsed, the server disconnects the client (client is granted a grace period of an additional 0.5 keepalive-time).
In the absence of data to be sent, the client sends a PINGREQ message instead.
Typical value for keepalive timer are a couple of minutes.



© Peter R. Egli 2015

10. MQTT will message

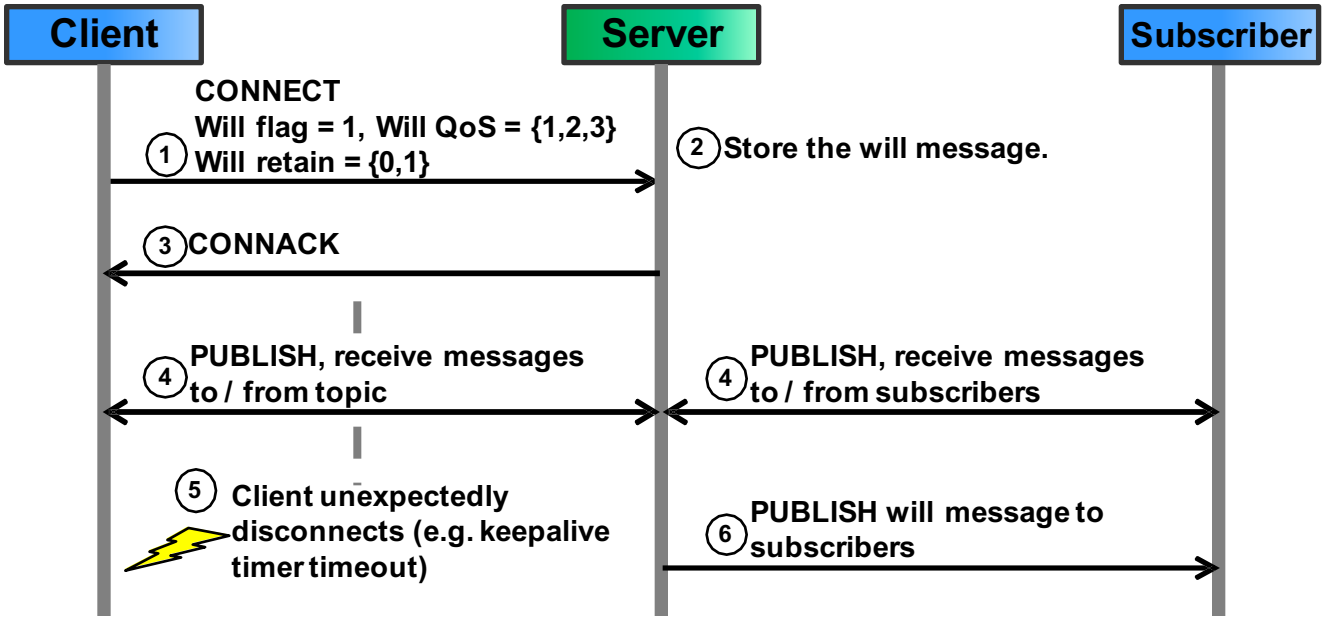
Problem:

In case of an unexpected client disconnect, depending applications (subscribers) do not receive any notification of the client's demise.

MQTT solution:

Client can specify a will message along with a will QoS and will retain flag in the CONNECT message payload.

If the client unexpectedly disconnects, the server sends the will message on behalf of the client to all subscribers («last will»).



© Peter R. Egli 2015

11. Topic wildcards

Problem:

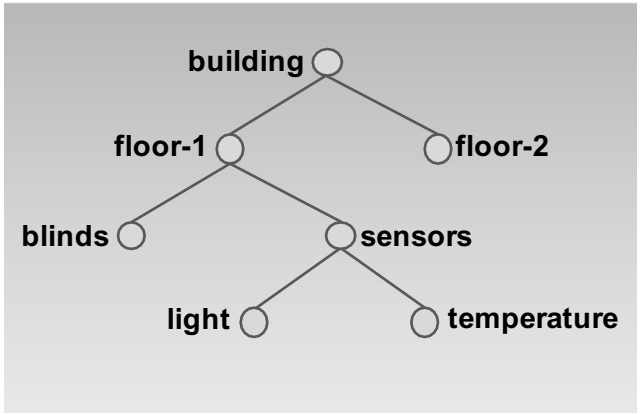
Subscribers are often interested in a great number of topics.
Individually subscribing to each named topic is time- and resource-consuming.

MQTT solution:

Topics can be hierarchically organized through wildcards with path-type topic strings and the wildcard characters '+' and '#'.
Subscribers can subscribe for an entire sub-tree of topics thus receiving messages published to any of the sub-tree's nodes.

Topic string special character	Description
/	Topic level separator. Example: *building / floor-1 / sensors / temperature*
+	Single level wildcard. Matches one topic level. Examples: *building / floor-1 / +* (matches *building / floor-1 / blinds* and *building / floor-1 / sensors*) *building / + / sensors* (matches *building / floor-1 / sensors* and *building / floor-2 / sensors*)
#	Multi level wildcard. Matches multiple topic levels. Examples: *building / floor-1 / #* (matches all nodes under *building / floor-1*) *building / # / sensors* (invalid, '#' must be last character in topic string)

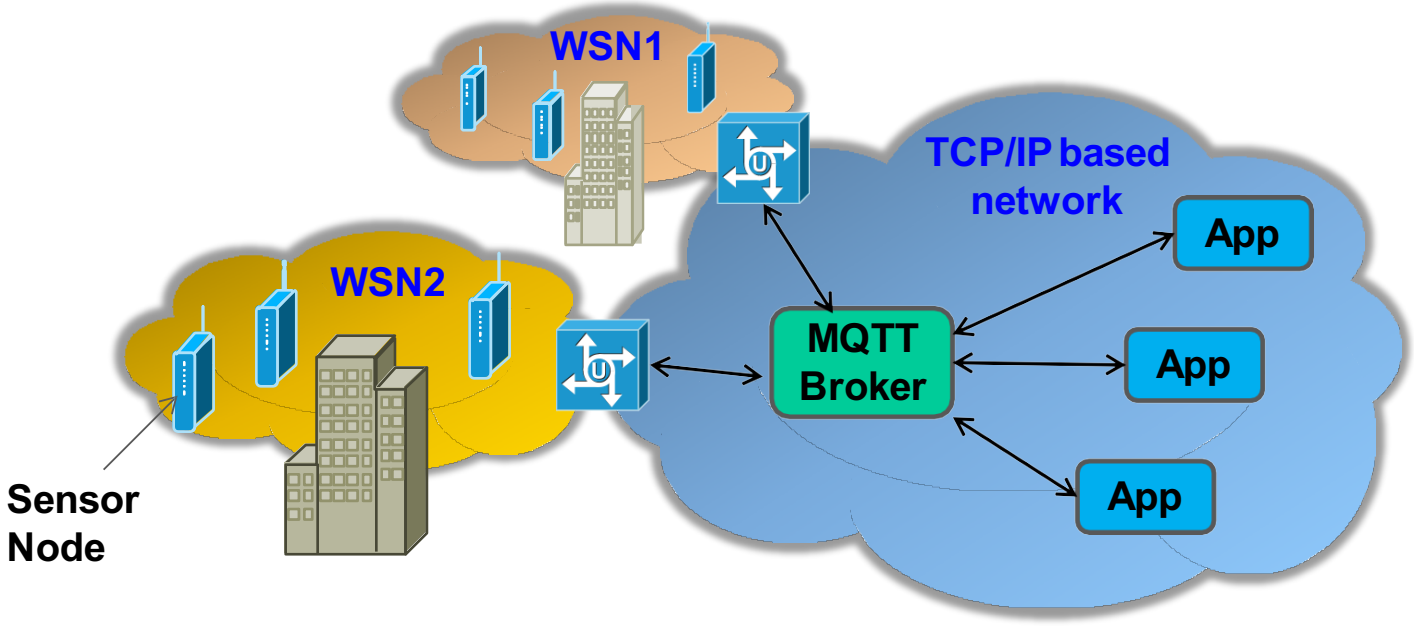
Example topic tree:



© Peter R. Egli 2015

12. MQTT-SN (1/Linux – Sensor Networks

WSNs (Wireless Sensor Networks) usually do not have TCP/IP as transport layer. They have their own protocol stack such as ZigBee on top of IEEE 802.15.4 MAC layer. Thus, MQTT which is based on TCP/IP cannot be directly run on WSNs.
WSNs are connected to traditional TCP/IP networks through gateway devices.



MQTT-SN is an extension of MQTT for WSNs.
MQTT-SN is aimed at constrained low-end devices, usually running on a batttery, such as ZigBee devices.

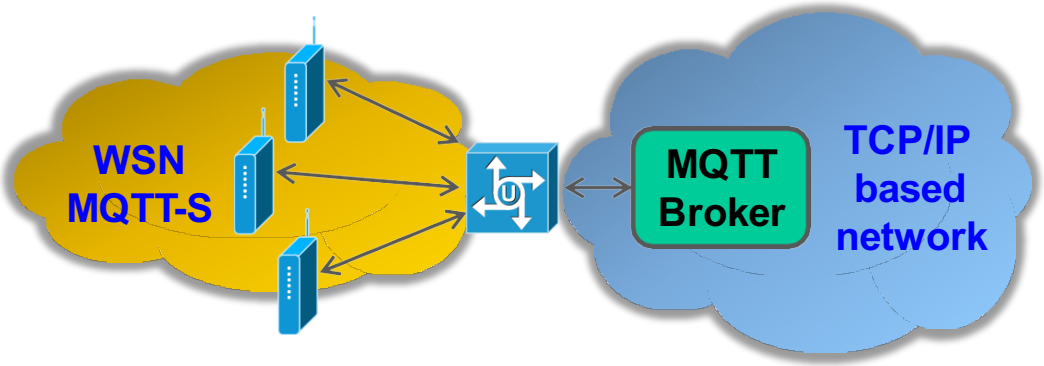
© Peter R. Egli 2015

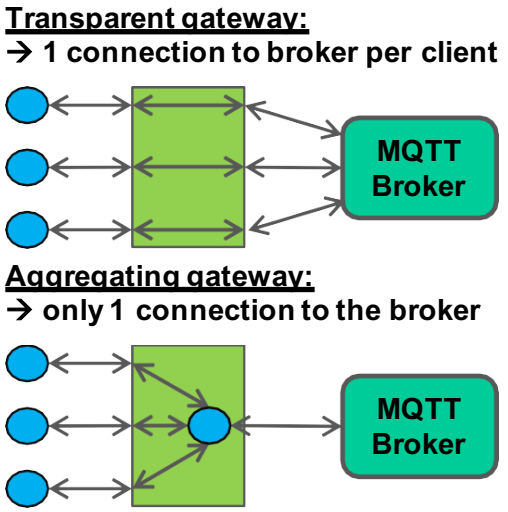
12. MQTT-SN (2/Linux – Sensor Networks

MQTT-SN is a largely based on MQTT, but implements some important optimizations for wireless networks:

- **Topic string replaced by a topic ID (fewer bytes necessary)**
- **Predefined topic IDs that do not require a registration**
- **Discovery procedure for clients to find brokers (no need to statically configure broker addresses)**
- **Persistent will message (in addition to persistent subscriptions)**
- **Off-line keepalive supporting sleeping clients (will receive buffered messages from the server once they wake up)**

MQTT-SN gateways (transparent or aggregating) connect MQTT-SN domains (WSNs) with MQTT domains (traditional TCP/IP based networks).





© Peter R. Egli 2015

IoT Communications Protocols MQTT

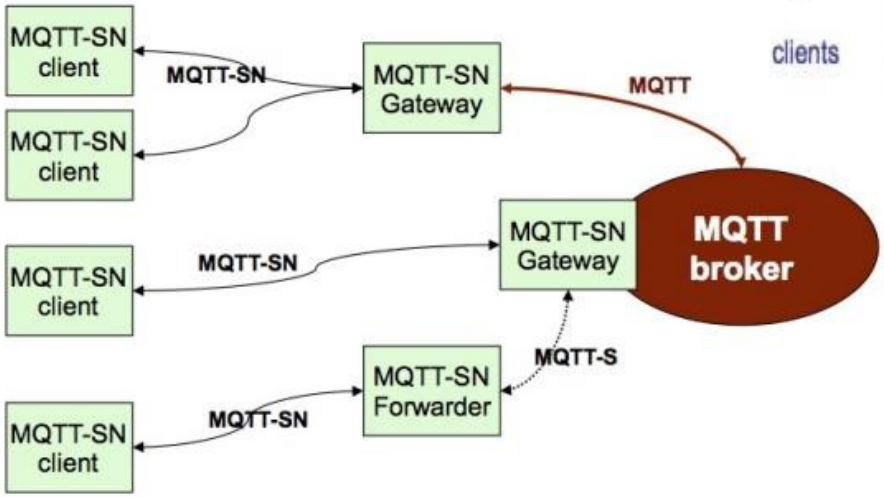
Why MQTT-SN (Sensors Network)?

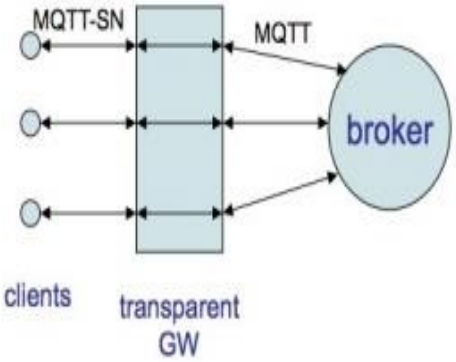
1. Very Long Packet Size for a 802.15.4 MAC layer

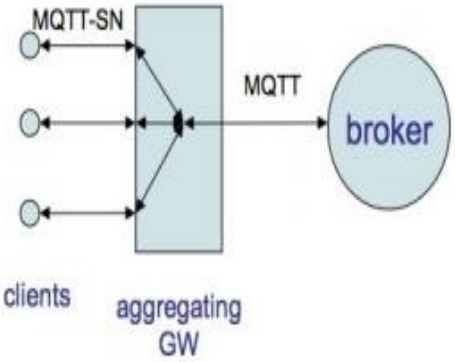
2. TCP as Transport Protocol

MQTT-SN is optimized for low-cost devices implementation, battery-supplied, and with limited computational and processing capabilities.

V1.2, November 2013.







Copyright: http://www.inginfpoliba.eu/index.php?action=downloadfile&filename=8-Application%20Layer%20for%20IoT.pdf&directory=Interet%20of%20Things

IoT Communications Protocols MQTT

MQTT vs MQTT-SN

MQTT-SN is designed to be as close as possible to MQTT, but is adapted to the peculiarities of a wireless communication environment.

1. CONNECT message, divided in three parts (Will Topic – Will Message);
2. Topic Name → Topic ID. Registration Procedure to obtain the ID for a particular Topic Name;
3. Pre-defined Topic ID and Short Topic ID (2bytes-long), for which no registration process is necessary;
4. Discovery Procedure to obtain the MQTT-SN Gateway IP Address;
5. not only client's subscriptions are persistent (RETAIN=1), but also Will topic and Will message.
6. support of sleeping clients: with this procedure, battery-operated devices can go to a sleeping state during which all messages destined to them are buffered at the server/gateway and delivered later to them when they wake up;

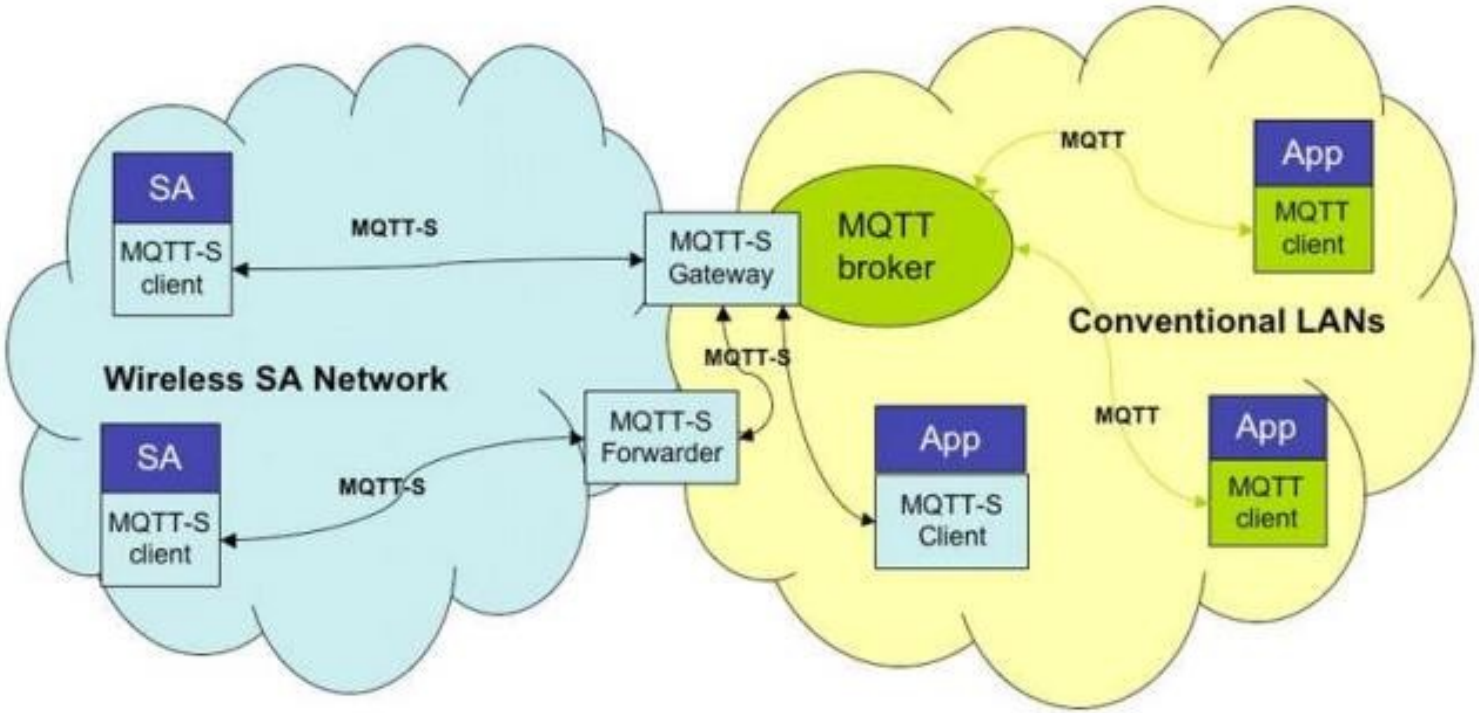
	MQTT	MQTT-S
Transport type	Reliable point to point streams	Unreliable datagrams
Communication	TCP/IP	Non-IP or UDP
Networking	Ethernet, WiFi, 3G	ZigBee, Bluetooth, RF
Min message size	2 bytes - PING	1 byte
Max message size	≤ 24MB	< 128 bytes (*)
Battery-operated		√
Sleeping clients		√
QoS: -1 "dumb client"		√
Gateway auto-discovery & fallbacks		√

ZADATA © 2013

Copyright: http://www.inginfpoliba.eu/index.php?action=downloadfile&filename=8-Application%20Layer%20for%20IoT.pdf&directory=Interet%20of%20Things

IoT Communications Protocols MQTT

MQTT ←→ MQTT-SN



Copyright: http://www.inginfpoliba.eu/index.php?action=downloadfile&filename=8-Application%20Layer%20for%20IoT.pdf&directory=Interet%20of%20Things

IoT Communications Protocols MQTT

MQTT @ OASIS

From March 2013, MQTT is being standardized at OASIS, starting from v3.1 IBM Protocol Specification.

Technical Commettee Charter: «The protocol has to support various implementations which run on embedded devices, with limited power, scarce processing and memory requirements, connected to a range of web services or enterprise middleware in constrained environments».

Targets: refinement of input specifications. Improvements:

1. Message Priority;
2. Payload Typing;
3. Request/Reply Mechanisms;
4. Subscriptions expiration.

Out of Scope:

1. Mapping of the specifications with a particular programming language or middleware;
2. No Reference Implementations for broker entities;
3. No MQTT topic namespace or conventions for topic classification or topic space;
4. No Security Mechanisms will be added, but a Transport Layer Security is assumed.

Copyright: http://www.inginfpoliba.eu/index.php?action=downloadfile&filename=8-Application%20Layer%20for%20IoT.pdf&directory=Interet%20of%20Things

MQTT: Clients & Brokers

MQTT Client Implementations:

- WebSphere MQ Telemetry Client (C,Java)
- Eclipse Paho (C, Java, Python, Lua)

MQTT Server Implementations:

- WebSphere MQ Broker (C, Java);
- Really Small Message Broker, RSMB (C);
- Mosquitto (JMS);

Utility for MQTT:

- Eclipse Paho (Eclipse);
- WMQTT (Java application);

Related Technology Proposals

Moquette MQTT: creation of a simple and small self-contained Java Implementation of a client broker;

Projects using MQTT

Say It, Sign It (SiSi): helps deaf people by converting speech into British Sign Language, rendered via an MQTT-attached Java avatar. The System uses MQTT and a micro-broker as its messaging infrastructure.

Location Aware Messaging for Accessibility (LAMA): it is a system for making information available in a way that is relevant to their interests and location. The system uses smartphones, MQTT and Websphere Message Broker and some rather clever application software.

Smart Lab: ideated at the University of Southampton, it was a project for monitoring lab experiments in the Chemistry department, and displaying a live dashboard on a Java-enabled cellphone, all using MQTT and the IBM broker technology.

FloodNet: the projects centres upon the development of providing a pervasive, continuous, embedded monitoring presence, by processing and synthesizing collected information over a river and functional floodplain.

Copyright: http://www.inginfpoliba.eu/index.php?action=downloadfile&filename=8-Application%20Layer%20for%20IoT.pdf&directory=Interet%20of%20Things

IoT Communications Protocols MQTT Libs for iOS

https://github.com/ckrey/MQTT-Client-Framework

For iOS MQTT Library there are available several 3rd party libraries.

Using C language libraries or wrapper libraries usually means that there are used POSIX networking calls at some point.

Apple forbids the use of third party networking libraries from using the mobile internet antenna. Thus if one uses C can-only use MQTT, when is connected to a Wi-Fi network.

Therefore, taking into consideration the observations from above and the security constraints - to use native iOS keychain mechanisms instead OpenSSL, there is only one library which is compliant with the requirements - please see the table:

IoT Communications Protocols MQTT Libs for iOS

https://github.com/ckrey/MQTT-Client-Framework

Name	Type	Programming Language	Code	Obs/Usage
~~Paho~~	~~Original~~	~~C~~	~~Open-Source. Eclipse project~~	~~Potential no access to GSM because of POSIX calls~~
~~IBM~~	~~Original~~	~~C~~	~~Close Source. IBM SDK~~	~~Potential no access to GSM because of POSIX calls~~
~~Mosquitto~~	~~Original~~	~~C~~	~~Open-Source. Eclipse project~~	~~Potential no access to GSM because of POSIX calls~~
~~MQTTKit~~	~~Wrapper (Mosquitto)~~	~~Objective-C~~	~~Open-Source. Github~~	~~No longer maintained~~
~~Marquette~~	~~Wrapper (Mosquitto)~~	~~Objective-C~~	~~Open-Source. Github~~	~~Only for JS | iOS not provided~~
~~Moscapsule~~	~~Wrapper (Mosquitto)~~	~~Swift~~	~~Open-Source. Github~~	~~Security using openSSL; not recommended by Apple !~~
~~Musqueteer~~	~~Wrapper (Mosquitto)~~	~~Objective-C~~	-	~~No repo available~~
MQTT-Client-Framework	Native	Objective-C	Open-Source. Github	It is used by an iOS App from Apple store – OwnTracks (https://github.com/owntracks/ios | https://itunes.apple.com/us/app/mqttitude/id692424691?mt=8) and has keychain and native security for SSL + no 3rd party usage
~~MQTTSDK~~	~~Native~~	~~Objective-C~~	-	~~No repo available~~
~~CocoaMQTT~~	~~Native~~	~~Swift~~	~~Open-Source. Github~~	~~No security~~

IoT Communications Protocols MQTT

MQTT vs CoAP

MQTT	CoAP
Many-to-Many Communication Protocol	One-to-One Communication Protocol
Decoupling producers and consumers	Transferring State Information between client and server
Data - Centric.	Document - Centric
It does best as a communication bus for live data	Best-suited to a state transfer model, not purely event-based
Clients make s long-lived outgoing TCP connection to a broker	Clients and servers both send and receive UDP packets.
No problem behind NAT	Tunnelling or Port Forwarding can be used to allow CoAP in NAT environments (IPv4). With IPv6 no problems.
No support for labelling messages. All clients must know the message format up-front to allow the communication.	Provides inbuilt support for content negotiation (ACCEPT) and discovery (list), allowing devices to find a way of exchanging data.
3 QoS Levels.	Reliability mechanisms is based on NON/CON messages.

Copyright: http://www.inginfpoliba.eu/index.php?action=downloadfile&filename=8-Application%20Layer%20for%20IoT.pdf&directory=Interet%20of%20Things

1. IoT Communications Protocols CoAP vs. MQTT

MQTT, CoAP, DDS and XMPP are the main competitors for IoT messaging at the Application Layer.

Each one of these has however some weaknesses:

- MQTT appears weak in security;
- DDS has problems in terms of scalability and various version dependence;
- XMPP is excessively heavy;
- CoAP not suitable for sending large sums of data and not reliable.

The choice among these is related to the desired QoS Level, the addressing capabilities and the particular application.

QoS is handled by TCP in MQTT, DDS and XMPP, but the mechanism defined there can be heavy in M2M communications. Because it targets device-to-device communications, DDS differs markedly from the other protocols in QoS control, but it is not ideal for device-to-server communications.

In that context MQTT and XMPP are the best-suited, for their discovery procedures.

"The Internet of Things is a big place, with room for many protocols. Choose the one for your application carefully and without prejudice of what you know."

Copyright: http://www.inginfpoliba.eu/index.php?action=downloadfile&filename=8-Application%20Layer%20for%20IoT.pdf&directory=Interet%20of%20Things

IoT Device Dev Boards and Platforms

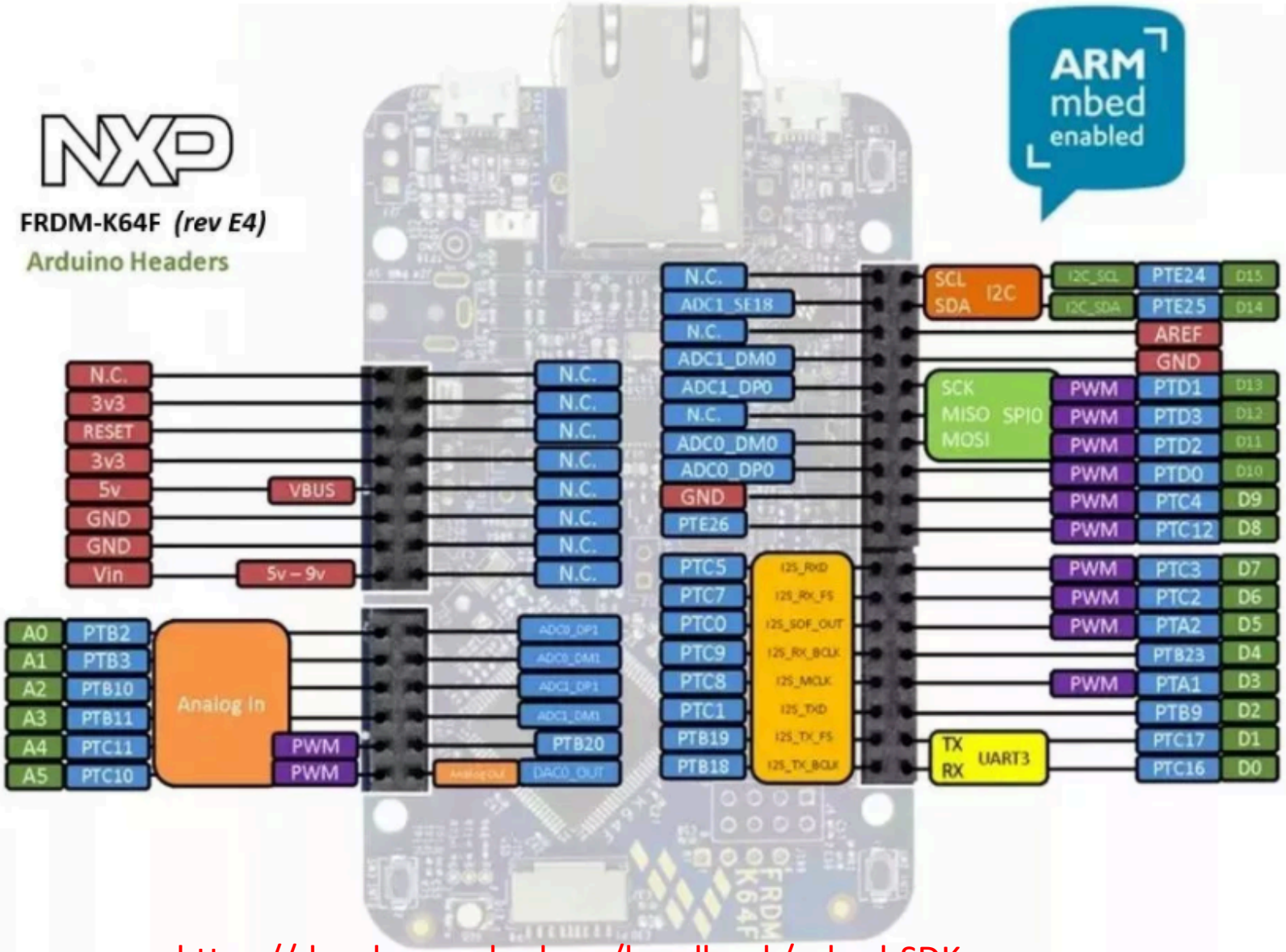
IoT Nodes Development Platforms:

- C* (e.g. C-mbed, C-POSIX, C-Arduino)
- Assembly / Firmware (e.g. GNU ARM)
- Lua/Node.js (e.g. NodeMCU vs. Espruino firmware for ESP8266)

IoT Gateways Development Platforms:

- C* (e.g. C-mbed, C-POSIX, C++, C-Arduino)
- Python 2.x and 3.x
- Java SE-e and Java ME-e
- Swift for ARM
- Node.js – Node-RED (JavaScript)
- C# .Net
- Mobile: iOS-Swift, Android-Java, Windows Mobile/IoT – C# .Net
- Lua
- Ruby/Perl (very rare)
- … mainly programming languages for Linux Embedded OS (on ARM)

IoT Node Device Dev Boards and Platforms



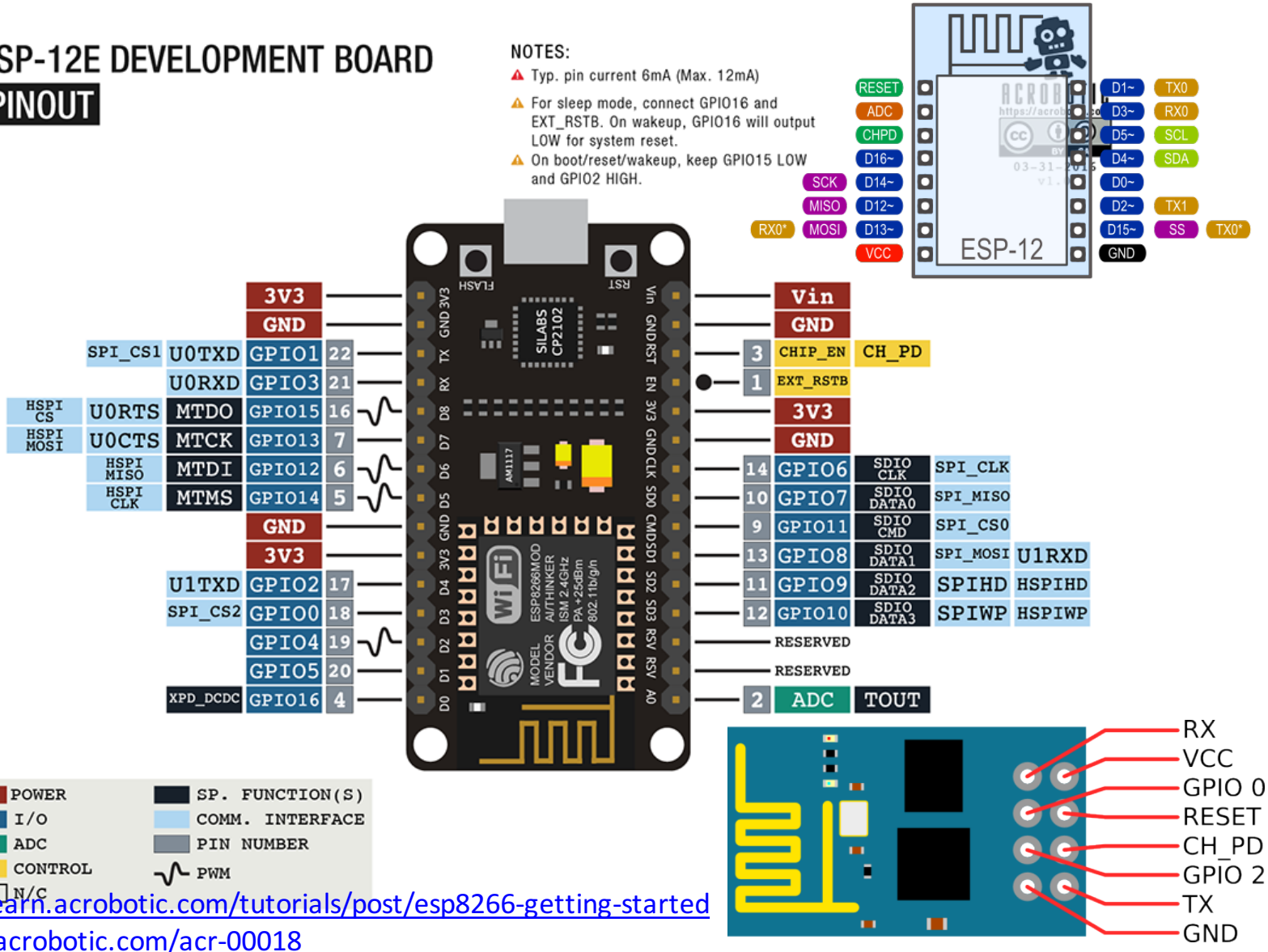
https://developer.mbed.org/handbook/mbed-SDK

IoT Device Nodes and Gateways Development

IoT Node Device Dev Boards and Platforms

ESP8266 (AT Commands / Lua – NodeMCU / JS-Node.js – Espruino)

IoT Device Nodes and Gateways Development

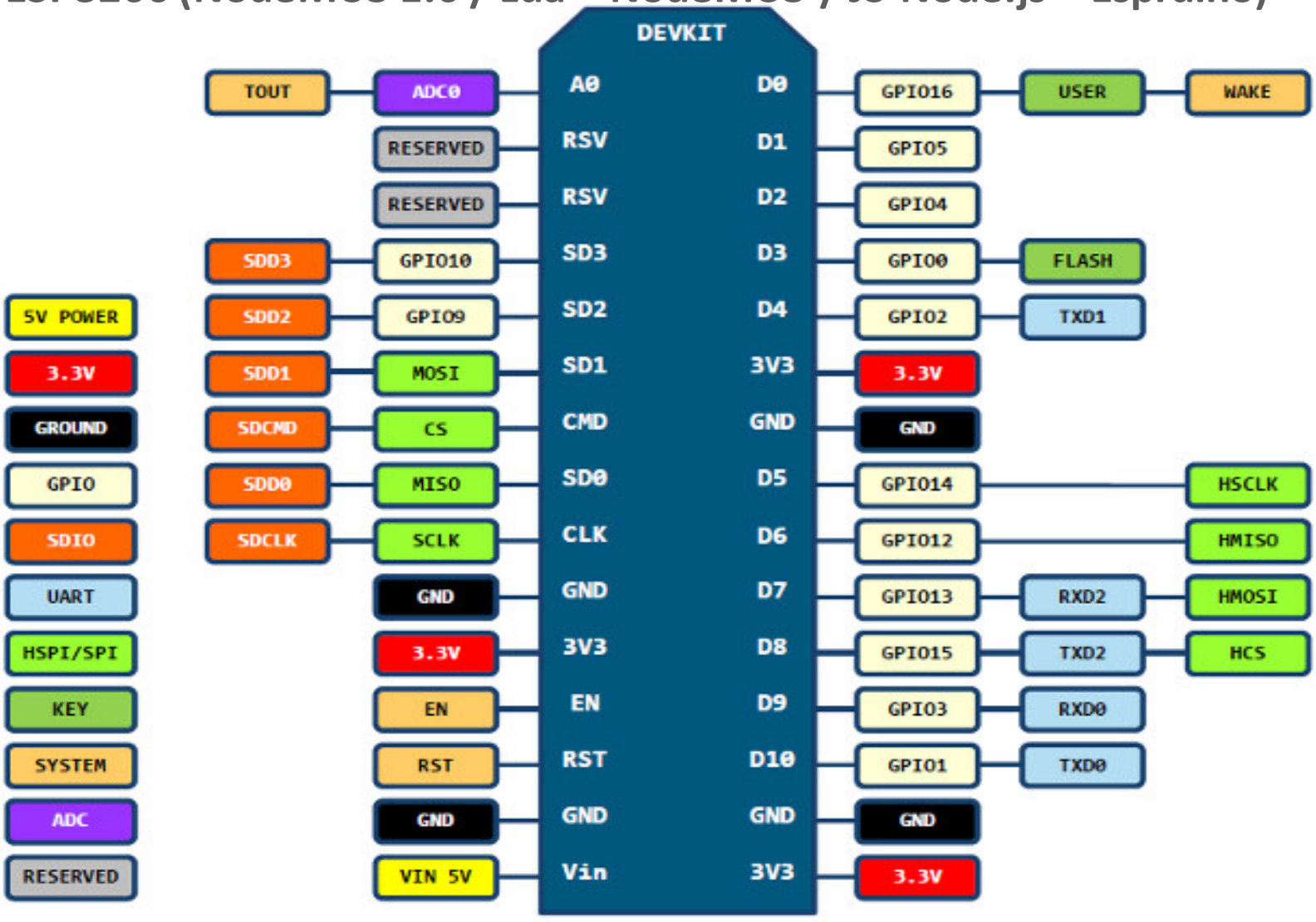


http://learn.acrobotic.com/tutorials/post/esp8266-getting-started

https://acrobotic.com/acr-00018

IoT Node Device Dev Boards and Platforms

ESP8266 (NodeMCU 1.0 / Lua – NodeMCU / JS-Node.js – Espruino)



D0(GPIO16) can only be used as gpio read/write, no interrupt supported, no pwm/i2c/ow supported.

Copyright: http://www.cnx-software.com/2015/10/29/getting-started-with-nodemcu-board-powered-by-esp8266-wisoc/

IoT Node Device Dev Boards and Platforms

ESP8266 – NodeMCU – Lua: http://thomaslauer.com/download/luarefv51.pdf | https://www.tutorialspoint.com/lua/

1. Download the latest firmware on Linux/Raspberry Pi:
@ https://github.com/nodemcu/nodemcu-firmware/releases | https://github.com/nodemcu/nodemcu-flasher/tree/master/Resources/Binaries | https://github.com/nodemcu/nodemcu-firmware/releases/download/0.9.6-dev_20150704/nodemcu_float_0.9.6-dev_20150704.bin

2. Install esptool from Github on Linux/Raspberry Pi:
git clone https://github.com/themadinventor/esptool.git
or https://github.com/espressif/esptool

3. Erase firmware & Flash the NodeMCU firmware from Linux/Raspberry Pi to ESP8266, after USB connection between boards (also have driver USB to UART – no need in Rpi 3 - http://www.silabs.com/products/mcu/pages/usbtouartbridgevcpdrivers.aspx):

Rpi: sudo esptool.py --port /dev/ttyUSB0 erase_flash

Mac: sudo esptool.py --port /dev/tty.wchusbserial14310 erase_flash

sudo python ./esptool.py --port /dev/ttyUSB0 write_flash 0x00000 ../nodemcu_integer_0.9.6-dev_20150704.bin

sudo python ./esptool.py --port /dev/ttyUSB0 write_flash 0x00000 ../nodemcu_float_0.9.6-dev_20150704.bin

Copyright: http://www.cnx-software.com/2015/10/29/getting-started-with-nodemcu-board-powered-by-esp8266-wisoc/ | http://www.esp8266.com/viewtopic.php?p=58281 | http://www.instructables.com/id/Develop-for-the-ESP8266-on-the-Raspberry-Pi/?ALLSTEPS

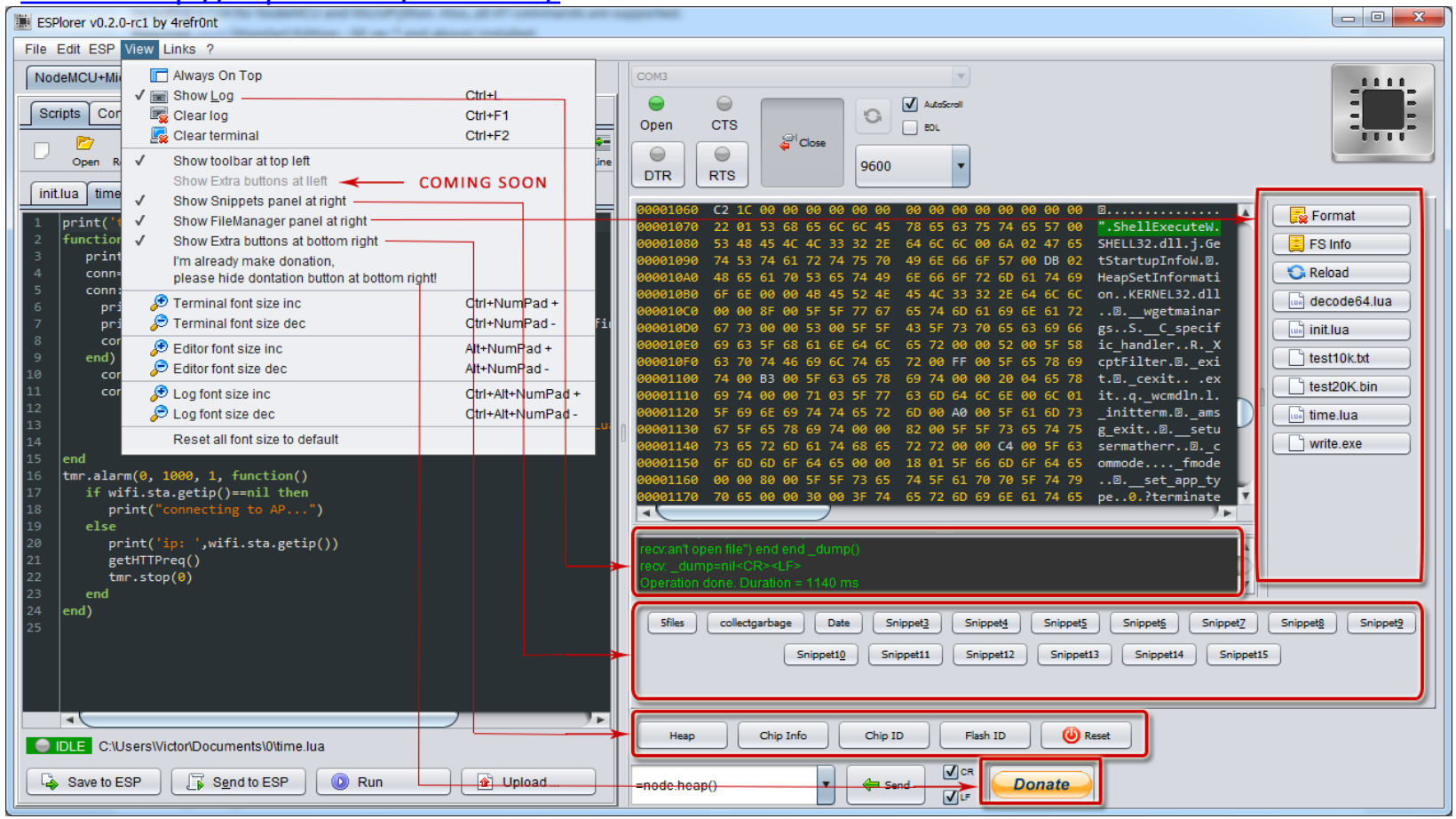
IoT Node Device Dev Boards and Platforms

ESP8266 - Lua – NodeMCU

4. Download Java ESPlorer tool:

https://github.com/4refr0nt/ESPlorer | https://github.com/hmic/ESPlorer

Home: http://esp8266.ru/ESPlorer/



Keep boudrate 9600 for uploading Lua scripts in ESPlorer IDE on Rpi:

Copyright: http://www.esp8266.com/viewtopic.php?f=22&t=882

IoT Node Device Dev Boards and Platforms

ESP8266 - Lua – NodeMCU: Write Code (http://www.cnx-software.com/2015/10/29/getting-started-with-nodemcu-board-powered-by-esp8266-wisoc/)

Connect to the wireless network

```
print(wifi.sta.getip()) --nil
wifi.setmode(wifi.STATION)
wifi.sta.config("SSID","password")
print(wifi.sta.getip()) --192.168.18.110
```

Arduino like IO access

```
pin = 1
gpio.mode(pin,gpio.OUTPUT)
gpio.write(pin,gpio.HIGH)
gpio.mode(pin,gpio.INPUT)
print(gpio.read(pin))
```

HTTP Client

```
-- A simple http client
conn=net.createConnection(net.TCP, false)
conn:on("receive", function(conn, pl) print(pl) end)
conn:connect(80,"121.41.33.127")
conn:send("GET / HTTP/1.1\r\nHost:
www.nodemcu.com\r\n" .."Connection: keep-
alive\r\nAccept: */*\r\n\r\n")
```

HTTP Server

```
-- a simple http server
srv=net.createServer(net.TCP)
srv:listen(80,function(conn)
conn:on("receive",function(conn,payload)
print(payload)
conn:send("<h1> Hello, NodeMcu.</h1>") end) end)
```

Copyright: http://www.nodemcu.com/index_en.html |
https://github.com/javieryanez/nodemcu-modules/tree/master/dht22

IoT Node Device Dev Boards and Platforms

ESP8266 – Espruino – JS-Node.js:

https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm

1. Download the latest firmware on Linux/Raspberry Pi:
@ http://www.espruino.com/Download |
http://www.espruino.com/files/espruino_1v89.zip

2. Install esptool from Github on Linux/Raspberry Pi:
git clone https://github.com/themadinventor/esptool.git
or https://github.com/espressif/esptool

3. Erase firmware & Flash the NodeMCU firmware from Linux/Raspberry Pi to ESP8266, after USB connection between boards (also have driver USB to UART – no need in Rpi 3 - http://www.silabs.com/products/mcu/pages/usbtouartbridgevcpdrivers.aspx):
RPi: sudo esptool.py --port /dev/ttyUSB0 erase_flash
Mac: sudo esptool.py --port /dev/tty.wchusbserial14310 erase_flash

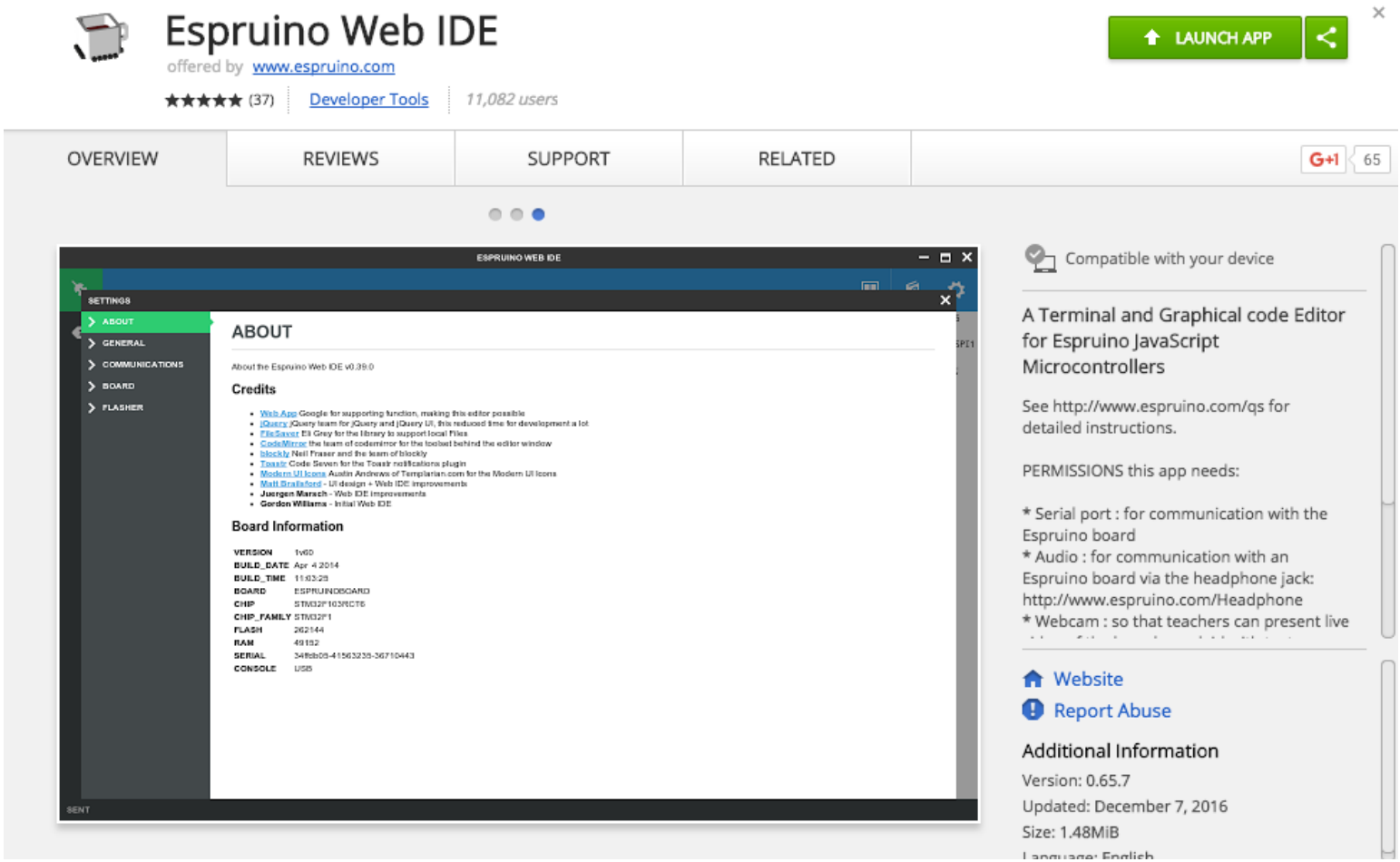
RPi: sudo python esptool.py --port /dev/ttyUSB0 -b 115200 write_flash -ff 80m -fm qio -fs 32m 0x0000 "boot_v1.4(b1).bin" 0x1000 espruino_esp8266_user1.bin 0x37E000 blank.bin
Mac: sudo python esptool.py --port /dev/tty.wchusbserial14310 -b 115200 write_flash -ff 80m -fm qio -fs 32m 0x0000 "boot_v1.4(b1).bin" 0x1000 espruino_esp8266_user1.bin 0x37E000 blank.bin

Copyright: http://justjibba.net/flashing-nodemcu-with-espruino/ | http://www.espruino.com/ | http://www.espruino.com/Other+Boards | http://www.espruino.com/EspruinoESP8266

IoT Node Device Dev Boards and Platforms

ESP8266 - Lua – NodeMCU

4. Download Espruino IDE tool as Google Chromium plugin in RPi:
https://chrome.google.com/webstore/detail/espruino-web-ide/bleoifhkdalbjfbobjackfdifdneehpo?hl=en



Keep boudrate 115200 for uploading JS-Node.js scripts (Settings-> Communications) from Rpi in ESP8266 | http://forum.espruino.com/conversations/281522/ | https://odd-one-out.serek.eu/esp8266-nodemcu-dht22-mqtt-deep-sleep/ | http://forum.espruino.com/conversations/281507/

IoT Node Device Dev Boards and Platforms

ESP8266 – JS-Node.js: Write Code

Arduino like IO access

```
var led = new Pin(2);
var toggle=1;
setInterval(
 function() {
  toggle=!toggle;
  digitalWrite(led, toggle);
 }, 500);
```

Arduino like IO access

```
var led = NodeMCU.D1;
var toggle=1;
setInterval(
 function() {
  toggle=!toggle;
  digitalWrite(led, toggle);
 }, 500);
```

Wi-Fi access

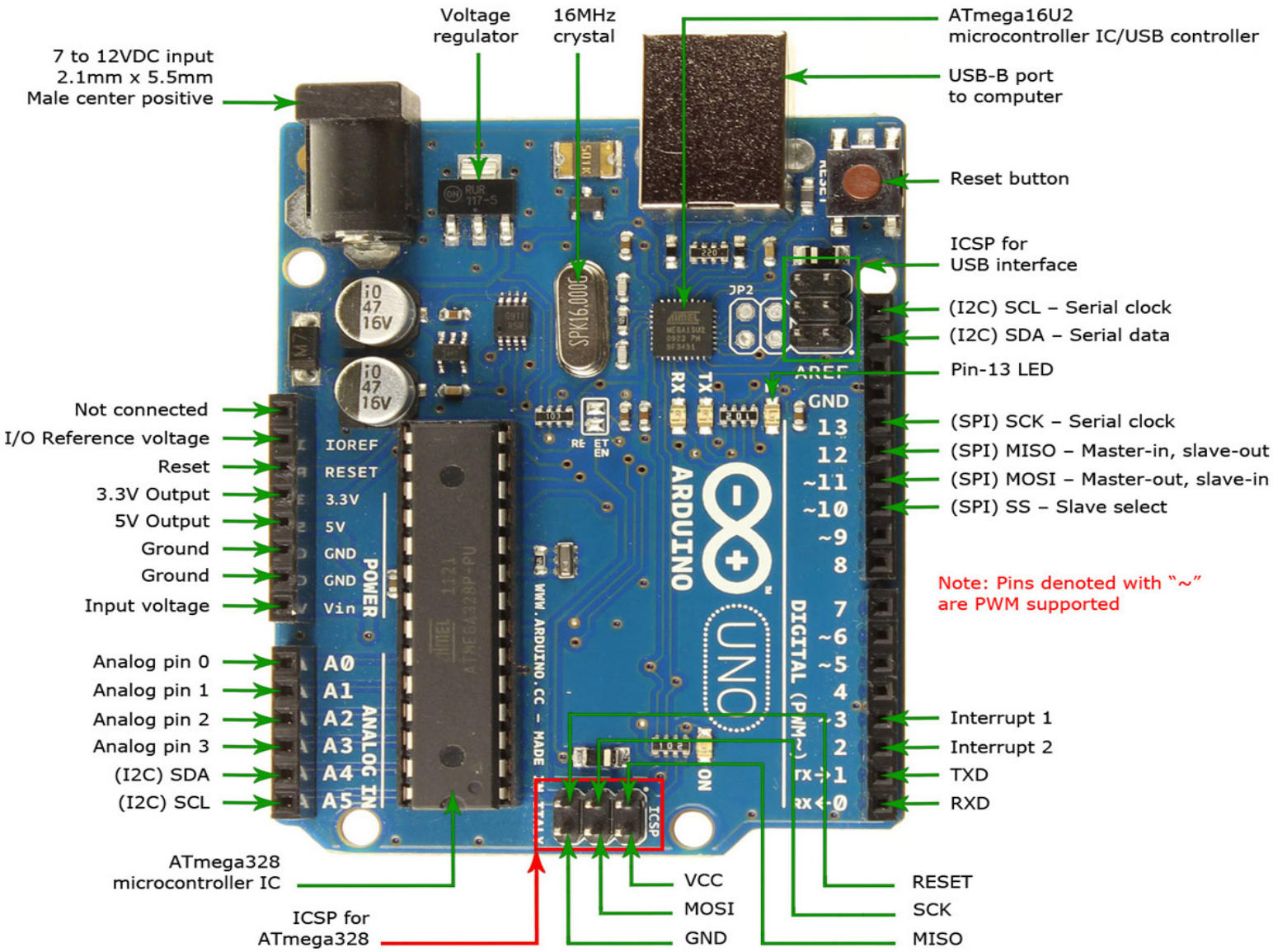
```
var wifi = require("Wifi");
wifi.connect("SSID", {password:”wpa2pass"}, function(err) {
  console.log("connected? err=", err, "info=", wifi.getIP() );
});
wifi.save();
wifi.stopAP();
```

http://www.espruino.com/Reference#NodeMCU

Copyright: http://justjibba.net/flashing-nodemcu-with-espruino/ | http://www.espruino.com/Tutorials

IoT Node Device Dev Boards and Platforms

Arduino (C-Arduino on ATmega16)



IoT Node Device Dev Boards and Platforms

Arduino UNO (C-Arduino on ATmega16)

Structure

Arduino programs can be divided in three main parts: **Structure, Values** (variables and constants), and **Functions**. In this tutorial, we will learn about the Arduino software program, step by step, and how we can write the program without any syntax or compilation error.

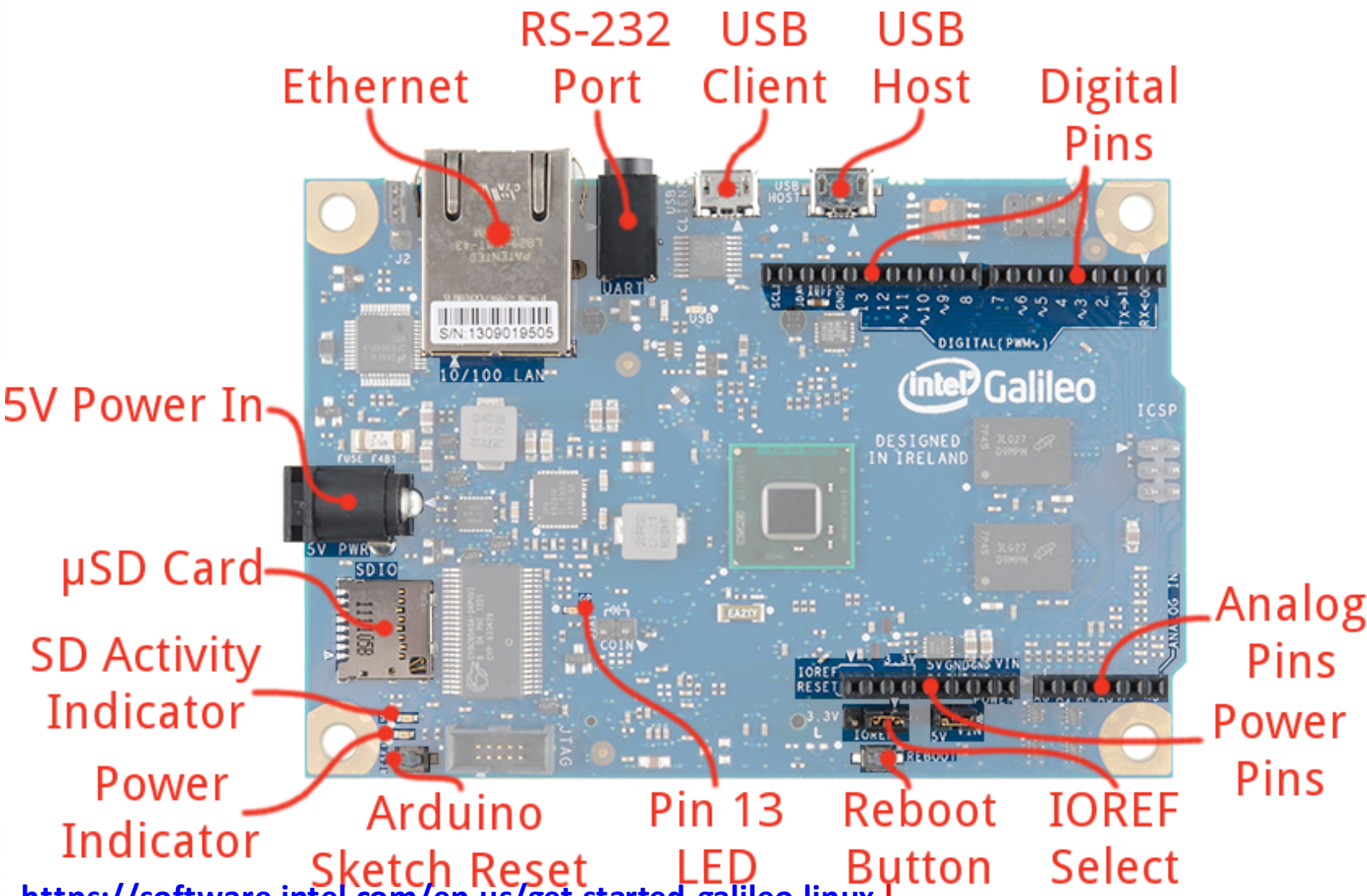
Let us start with the **Structure**. Software structure consist of two main functions –

- Setup( ) function
- Loop( ) function

http://www.tutorialspoint.com/arduino/ | https://www.arduino.cc/en/Tutorial/HomePage https://www.arduino.cc/en/Guide/Libraries | https://github.com/RobTillaart/Arduino/tree/master/libraries/DHTlib | http://playground.arduino.cc/Main/DHTLib

IoT Node Device Dev Boards and Platforms

Intel Galileo Gen 1&2 **(C-Arduino in SoC | C-Posix / Node.js /… in Yocto Linux on Intel Quark SoC)**



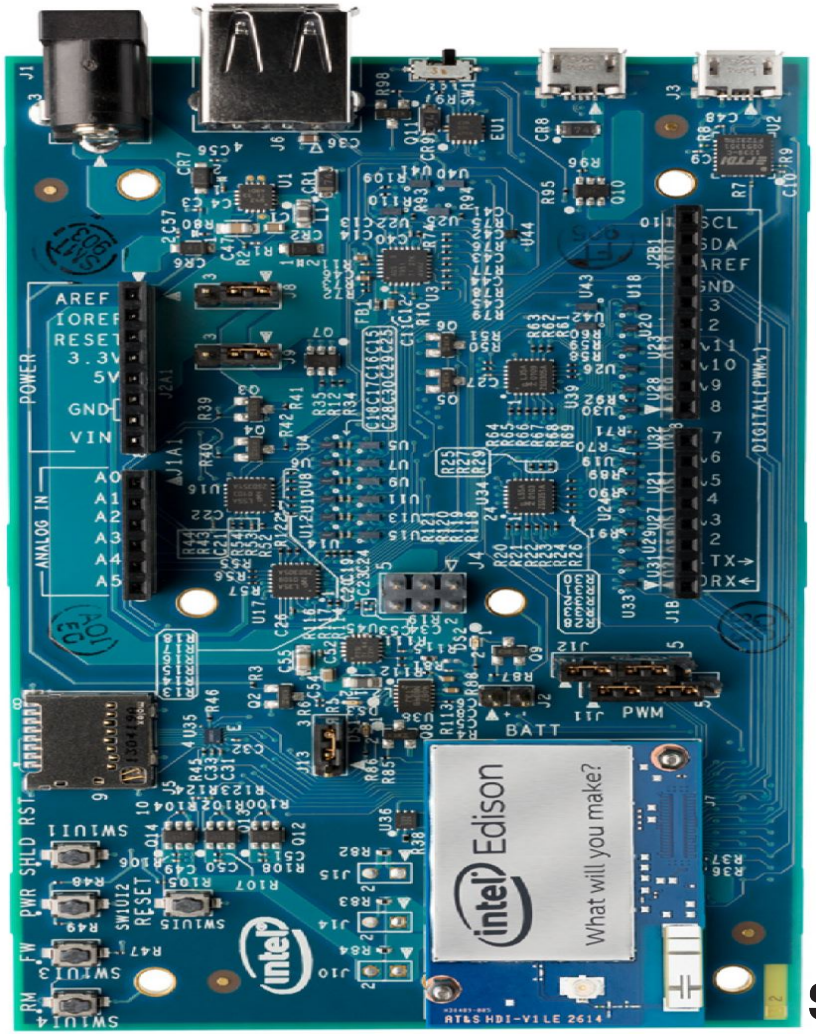
https://software.intel.com/en-us/get-started-galileo-linux |
https://www.arduino.cc/en/Guide/IntelGalileoGen2

IoT Node & GW Device Dev Boards and Platforms

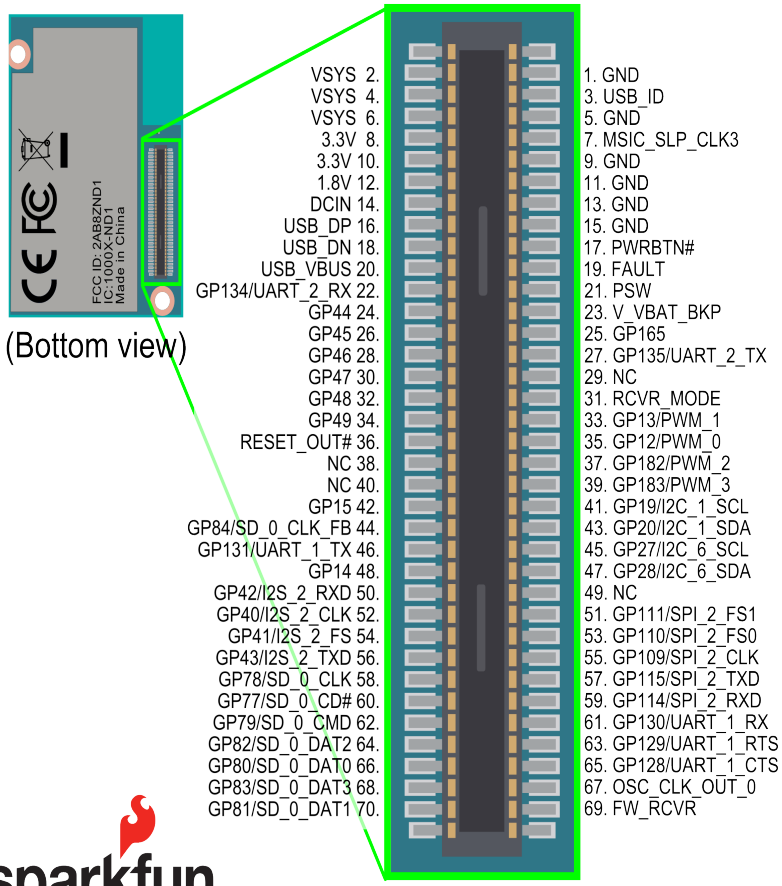
IoT Device Nodes and Gateways Development

Intel Edison (C-Arduino in SoC | C-Posix / Node.js /… in Yocto Linux on 2 CPU ATOM cores + 1 SoC Quark)

Real Platforms: Intel NUC Gateway



Intel© Edison Pinout



(Bottom view)

sparkfun ELECTRONICS

https://software.intel.com/en-us/get-started-edison-osx | https://software.intel.com/en-us/get-started-edison-linux | https://software.intel.com/en-us/iot/tools-ide/ide

IoT Node/GW Device Dev Boards and Platforms

Intel Edison vs. Galileo

Galileo
- CPU: Intel Quark X1000 400 MHz
- RAM: 256 MB
- Storage: Micro SD Card

Edison
- CPU: A dual core, dual threaded Intel ATOM x86 CPU running at 500 MHz and a 32-bit Intel Quark Micro-controller running at 100 MHz.
- RAM: 1 GB
- Storage: 4 GB ROM + (micro SD card on Arduino board)
- Communication: Wi-Fi and Bluetooth LE.

Summary
- Edison is way more powerful in terms of CPU (ATOM vs Quark) and RAM.

Stackoverflow.com:

Intel(R) Edison is a product-ready, general-purpose compute platform optimized to enable rapid innovation and product development. Intel Edison is ideal for small form factor devices that require a powerful computing system. Some good use cases are robots and quadcopters, 3D fabrication machines, remote asset monitoring, and audio processing.

Intel(R) Galileo is an open source, Arduino-compatible platform that enables educators, students, and makers of all skill levels to quickly and easily engage in projects. It combines the simplicity of the Arduino development environment with the performance of Intel technology and the advanced capabilities of a full Linux software stack.
A really great place to learn more about both platforms is our online community at **maker.intel.com**.

IoT Node/GW Device Dev Boards and Platforms: Intel

https://software.intel.com/en-us/iot/tools-ide/ide

https://software.intel.com/en-us/intel-xdk

https://software.intel.com/en-us/get-started-arduino

https://software.intel.com/en-us/iot/tools-ide/ide/iss-iot-edition

https://software.intel.com/en-us/intel-system-studio-microcontrollers

https://github.com/intel-iot-devkit/upm/tree/master/examples/python

WindRiver: https://software.intel.com/en-us/iot/hardware/gateways

The Intel® IoT Developer Kit is programmable using Arduino*, C/C++, JavaScript*, Node.js*, and Python*. Explore the list below to find the best solution for you.

Intel® XDK

Create web interfaces, add sensors to your project, and work with the cloud. This developer kit includes companion templates to get your project up and running quickly.

Arduino*

With readily available code from a variety of manufacturers, quickly add sensors to your project with this intuitive interface.

Intel® System Studio IoT Edition

This Eclipse*-based IDE comes with the built-in capability to easily integrate sensors via UPM and MRAA libraries, which you can develop in C/C++ or Java.

Intel® System Studio for Microcontrollers

Develop for Intel® Quark™ microcontrollers using this Eclipse*-based software suite. Effective debug capabilities, powerful library extensions, and code portability enable innovation for low-power connected devices.

Python*

Even though Intel does not provide an IDE for Python*, the Python programming language comes preinstalled on your board, plus you can find Python support in the sensor library.

Wind River* Intelligent Device Platform XT

Harness the connectivity, security, and manageability features of the Wind River* Intelligent Device Platform XT.

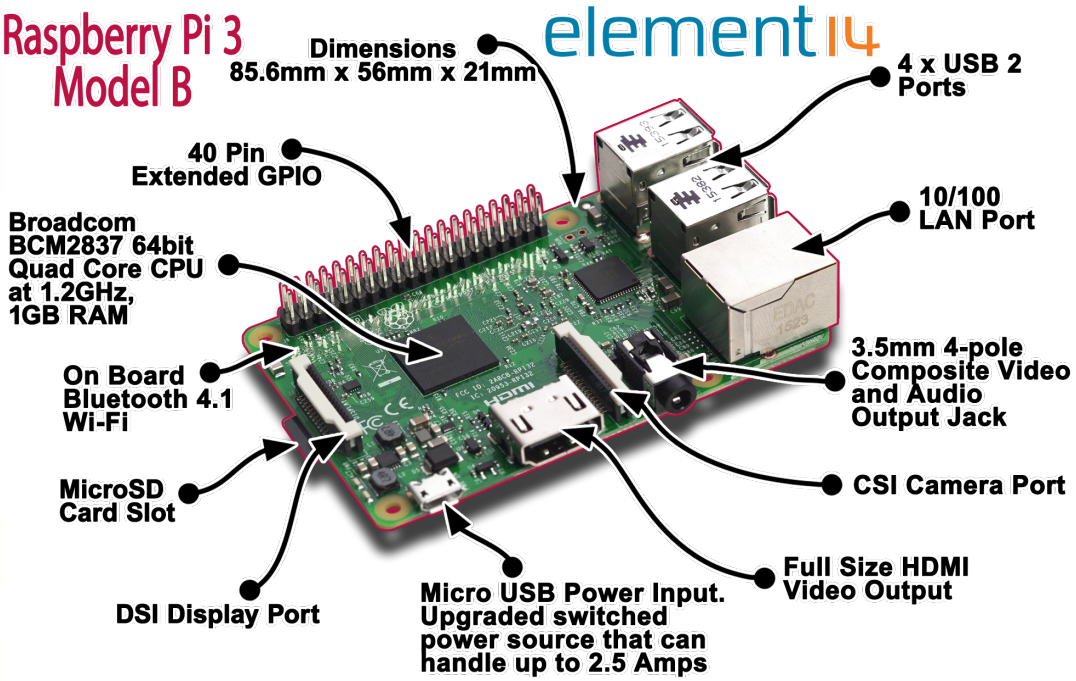
IoT Device Nodes and Gateways Development

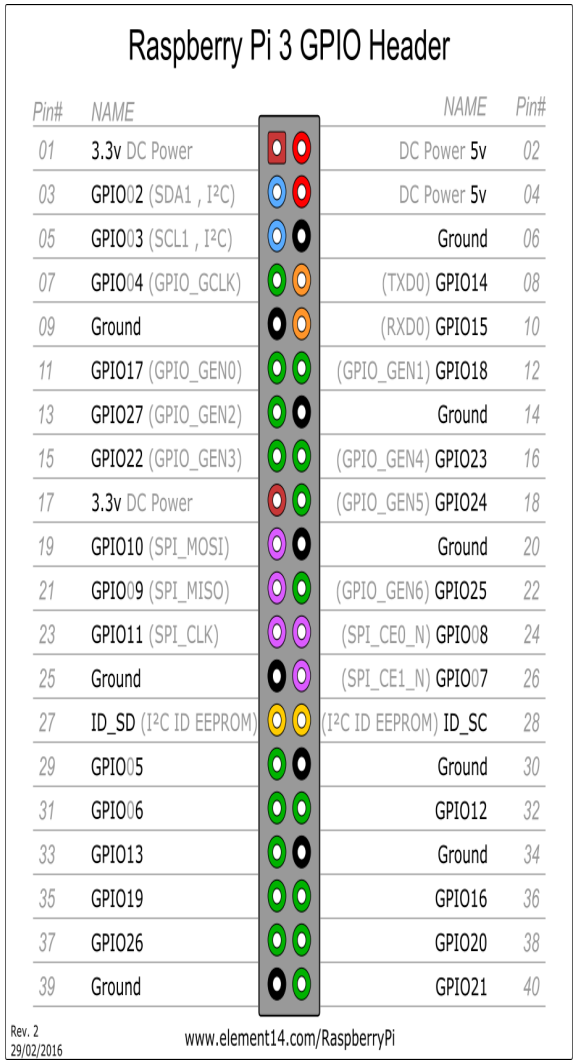
IoT Gateway Device Dev Boards and Platforms

Raspberry Pi Model 1 / 2 / 3 with A / B / B+ layout

C POSIX/C++, Java SE-e / Java ME, Python, Node.js/Node-RED, Swift

(http://dev.iachieved.it/iachievedit/swift-3-0-on-raspberry-pi-2-and-3/)





IoT Gateway Device Dev Boards and Platforms

Raspberry Pi Model 1 / 2 / 3 with A / B / B+ layout

C POSIX/C++, Java SE-e / Java ME, Python, Node.js/Node-RED, Swift, ARM - http://elinux.org/RPi_GPIO_Code_Samples

C on ARM: http://www.valvers.com/open-software/raspberry-pi/ | http://www.valvers.com/open-software/raspberry-pi/creating-a-bootable-sd-card/ | http://www.valvers.com/open-software/raspberry-pi/step01-bare-metal-programming-in-cpt1/ | http://www.valvers.com/open-software/raspberry-pi/step02-bare-metal-programming-in-c-pt2/ … http://www.valvers.com/open-software/raspberry-pi/step02-bare-metal-programming-in-c-pt5/

Java SE-e & OpenJDK DIO: http://openjdk.java.net/projects/dio/ | https://wiki.openjdk.java.net/display/dio/Getting+Started | https://www.tutorialspoint.com/java/ | https://docs.oracle.com/javase/tutorial/ | http://docs.oracle.com/javame/8.0/api/dio/api/index.html

Java ME & OpenJDK DIO: http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/RaspberryPi_Setup/RaspberryPi_Setup.html | http://www.oracle.com/technetwork/java/embedded/javame/embed-me/downloads/java-embedded-java-me-download-2162242.html | http://docs.oracle.com/javame/8.1/get-started-rpi/toc.htm | http://www.oracle.com/technetwork/articles/java/cruz-gpio-2295970.html | http://docs.oracle.com/javame/8.0/api/dio/api/index.html

IoT Gateway Device Dev Boards and Platforms

Raspberry Pi Model 1 / 2 / 3 with A / B / B+ layout

C POSIX/C++, Java SE-e / Java ME, Python, Node.js/Node-RED, Swift, ARM - http://elinux.org/RPi_GPIO_Code_Samples

ASM ARM: http://thinkingeek.com/arm-assembler-raspberry-pi/ |
https://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/

Python : https://www.tutorialspoint.com/python/ |
https://www.tutorialspoint.com/python3/ |
https://www.python.org/about/gettingstarted/ |
https://docs.python.org/3/tutorial/ |
https://docs.python.org/2/tutorial/index.html

JS-Node.js: http://www.tutorialspoint.com/nodejs/ |
https://www.airpair.com/javascript/node-js-tutorial |
https://nodejs.org/en/docs/ | http://eloquentjavascript.net/

Node-RED: http://noderedguide.com/ |
http://nodered.org/docs/hardware/raspberrypi | http://noderedguide.com/nr-lecture-1/ …

Swift: **http://dev.iachieved.it/iachievedit/swift-3-0-on-raspberry-pi-2-and-3/** |
http://dev.iachieved.it/iachievedit/more-swift-on-linux/ |
http://www.tutorialspoint.com/swift/

Introduction to the Java Device I/O (DIO) APIs

Jen Dority
Senior Member of Technical Staff
October 1, 2014

* Parts of the slides for Java DIO - copyright from Thierry Violleau and Cristian Toma (Oracle)

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

DIO Agenda

1. Overview of The Device I/O OpenJDK Project
2. Building the Device I/O libraries
3. Using the Device I/O APIs
4. A closer look at working with GPIO, SPI, I2C and UART
5. More info

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The Device I/O Project

The Device I/O Project is an OpenJDK to provide a Java-level API for accessing generic device peripherals on embedded devices.

- Follows the JavaME / Java SE-e Device I/O API
- Targets Linux/ARM SBCs
  - Raspberry Pi
  - SABRE Lite
- Supports an initial set of four peripheral device APIs
  - GPIO | SPI | I2C | UART
- Provides a consistent method for accessing low level peripherals on embedded devices
- Is extendable with service providers
- Helps developers manage multiple hardware configurations by providing the ability to assign logical names to devices

JavaOne ORACLE

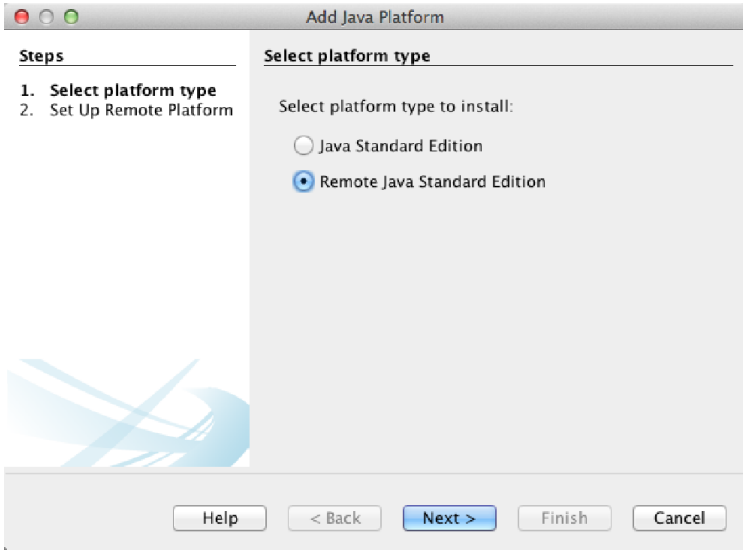
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Building The Device I/O Libraries

- Supports building on a Linux host with ARM cross-compiler
- Requires JDK7 or JDK8, Linux/ARM cross-compiler and GNU Make
- Sample code may also use the ANT build tool
- Define required environment variables
  - `export JAVA_HOME=<path to JDK>`
  - `export PI_TOOLS=<path to Linux/ARM cross-compiler>`
- Get the source
  - `hg clone http://hg.openjdk.java.net/dio/dev`
- Build
  - `cd dev`
  - `make`
    - Completed library files will be in build directory
      - `<top-level>/build/jar/dio.jar`
      - `<top-level>/build/so/libdio.so`

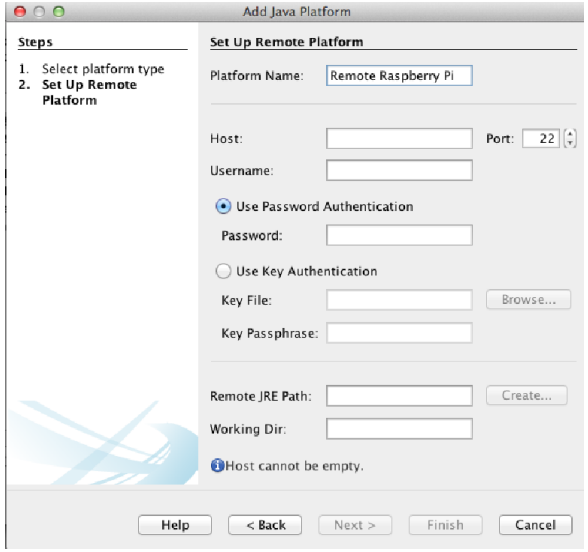
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Working With DIO APIs in Netbeans (in MacOS)



Tools → Java Platforms → Add Platform . . .

Select “Remote Java Standard Edition” then click next



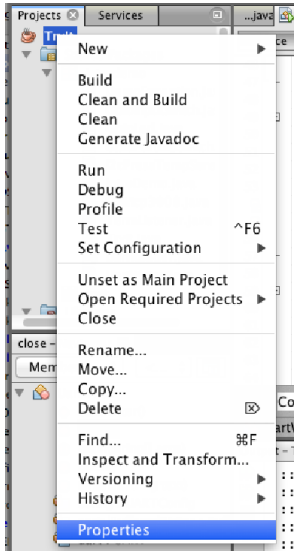
Fill in required fields then click “Finish”

Note: may need to use “root” credentials to run DIO apps from netbeans

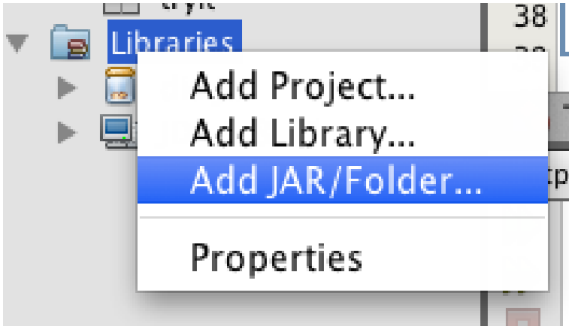
JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

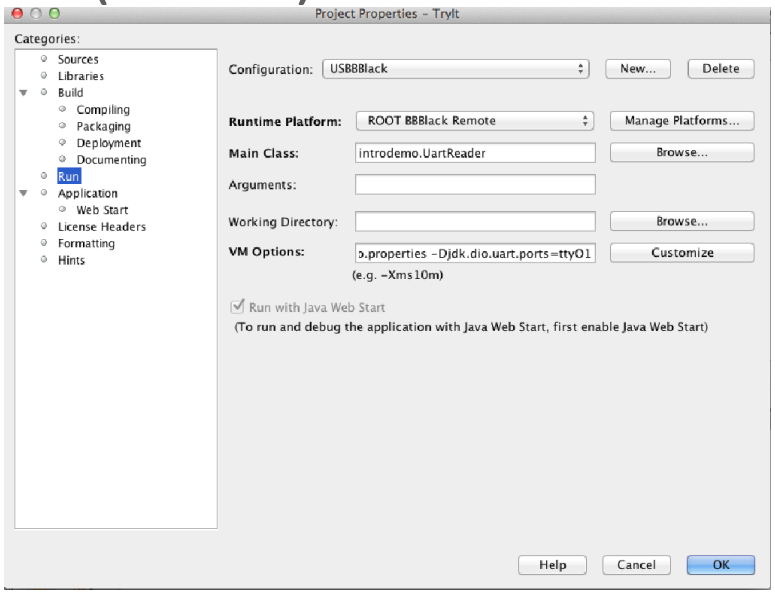
Working With DIO APIs in Netbeans (cont'd)



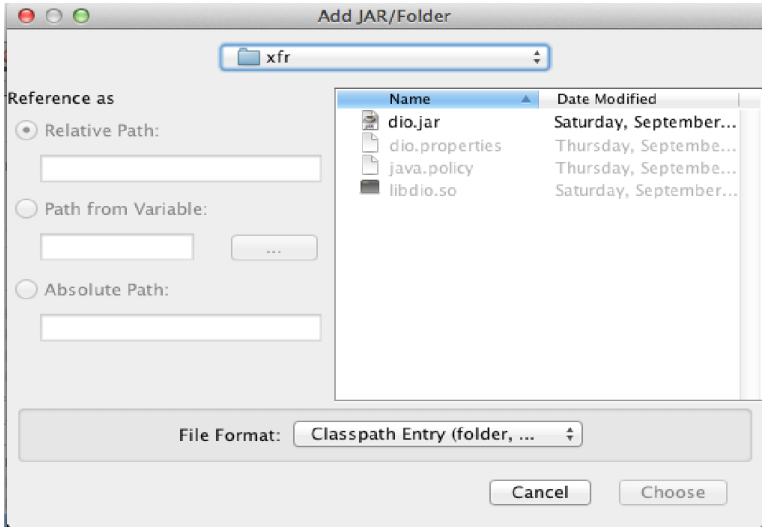
Right click on your project and select "Properties"



Right click on "libraries" in your project tree and select "Add JAR/Folder…"



Create a new configuration with your new remote platform



Choose the dio.jar file

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Using the Device I/O APIs

- Copy libido.so to your native java library path on the target device (see also $LD_LIBRARY_PATH)
  - Or, specify its location with –Djava.library.path in VM options
- Specify -Djdk.dio.registry in VM options (or in the `java` command line) to use a `.properties` file to preload a set of device configurations which you can refer to by a numeric ID
- Use DeviceManager.list() to get a list of all preloaded and user-registered devices in the system
- Get a device instance by using DeviceManager.open() methods
- When done with a device, be sure to call its close() method
- Access to devices depends on appropriate OS level access and new Java permissions

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

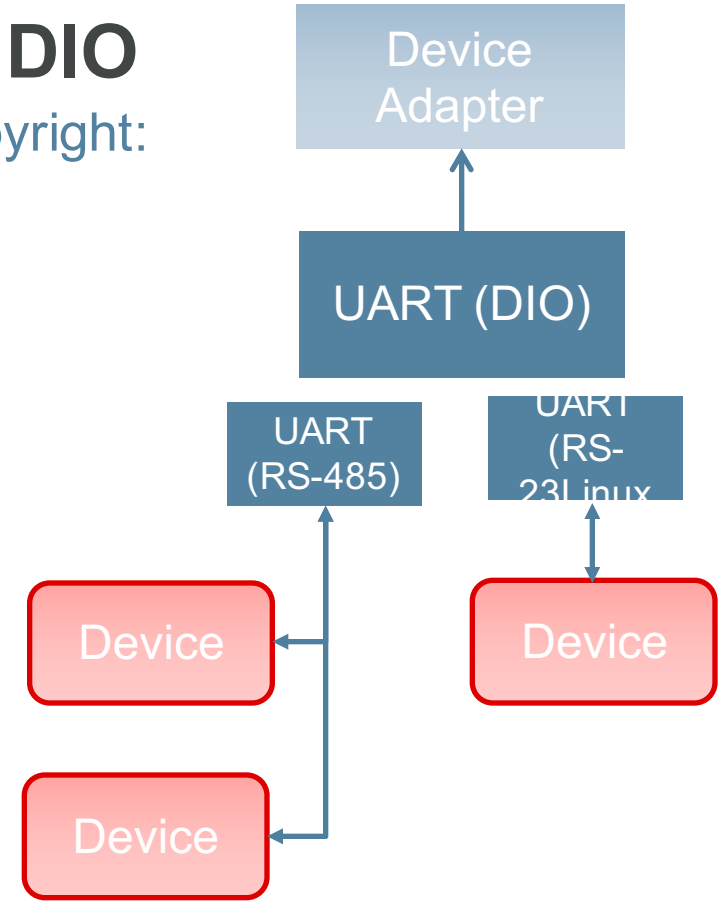
Oracle IoT GW 1.0 uses DIO

…Thierry Violleau Presentation Copyright:

- Enable the development of Device Adapters for devices connected over RS-485 and RS-232 serial line communications

- Support RS-485 and RS-232 through DIO UART API

- Target platforms:
  - Rpi
  - Nitrogen6 SabreLite

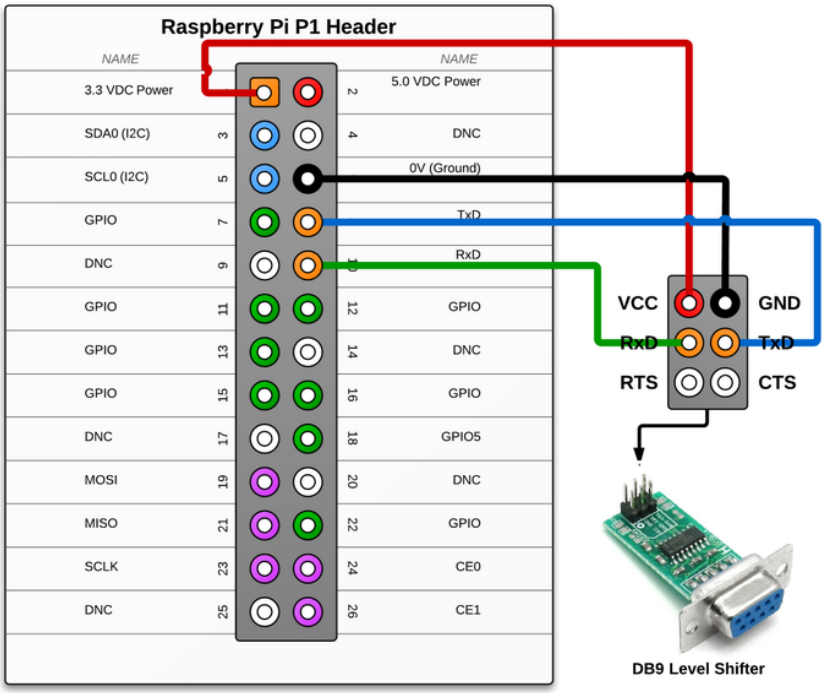


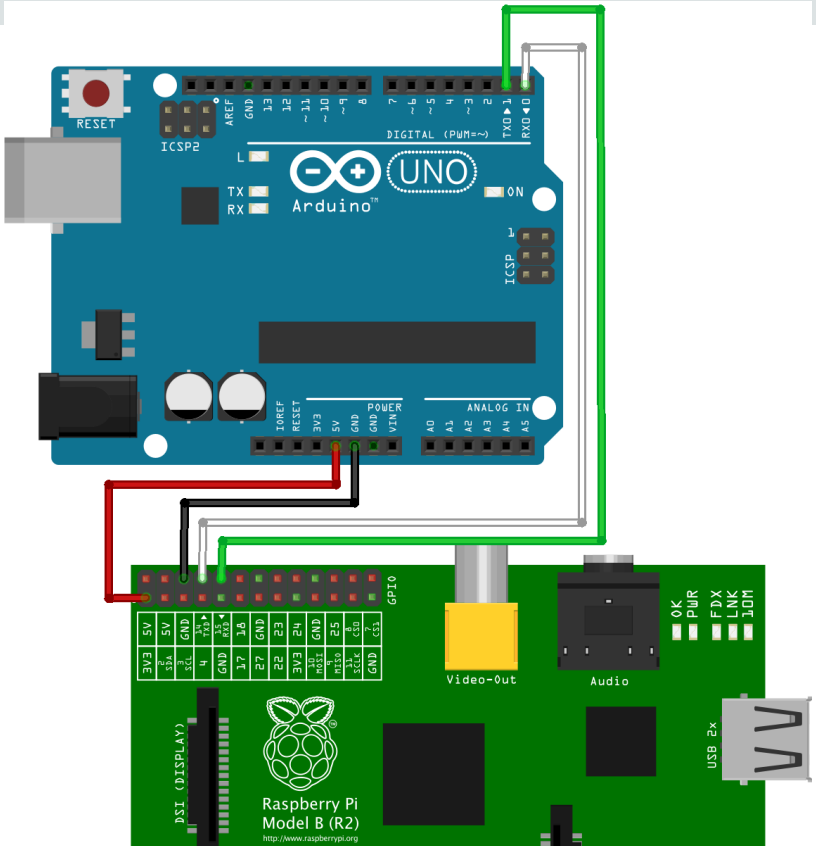
A Closer Look . . .

UART

Universal Asynchronous Receiver/Transmitter

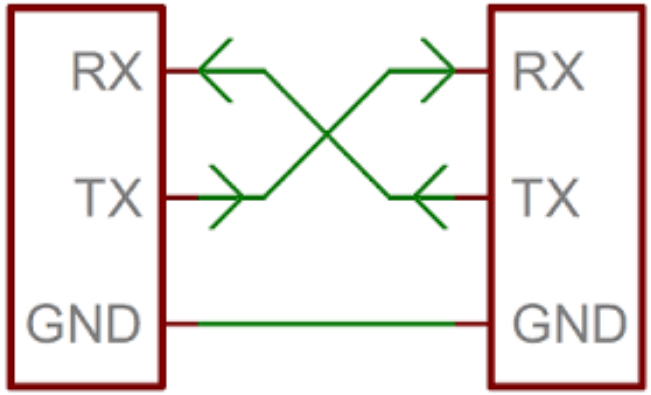
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

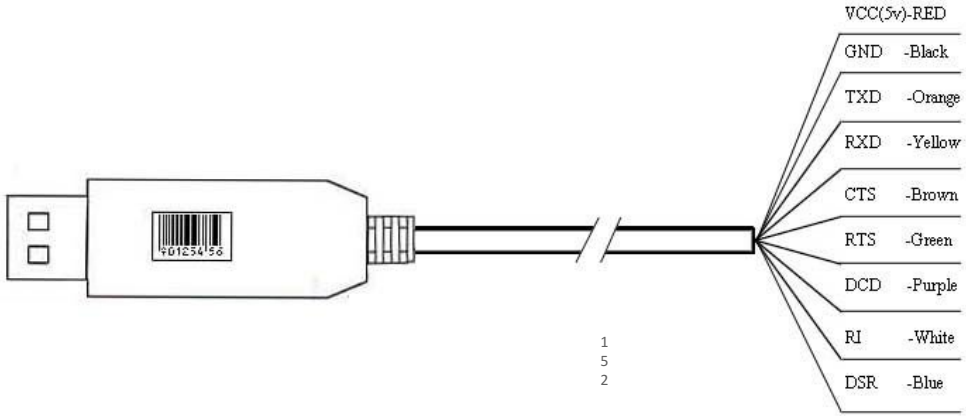




Samples. . .

UART





JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

UART

- Universal Asynchronous Receiver/Transmitter
- UARTs are commonly used in conjunction with communication standards such as TIA (formerly EIA) RS-232, RS-422 or RS-485.
- It is an asynchronous protocol because of the protocol and the 4 wires:
  - 2 wires for Data: RX (Receive) and TX (Transmit)
  - 2 wires for VCC (Voltage) and GND (Ground)

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

jdk.dio.uart.UART

Key configuration details

- Controller name or number
- Baud rate
- Parity
- Stop bits
- Flow control

- Allows for control and access of a UART device
- Provides methods to for synchronous and asynchronous reads and writes
- Implements the java.nio.channels interfaces ReadableByteChannel and WriteableByteChannel
- Uses java.nio.ByteBuffer in API calls

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

jdk.dio.uart.UART

```
UARTConfig config  = new UARTConfig(“ttyAMA0”,           // device name
                                    DeviceConfig.DEFAULT, // channel
                                    9600,                 // baud rate
                                    UARTConfig.DATABITS_7,
                                    UARTConfig.PARITY_NONE,
                                    UARTConfig.FLOWCONTROL_NONE);
        . . .
UART uart = DeviceManager.open(config);
OutputStream os = Channels.newOutputStream(uart);
BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(os));

writer.print(“Hello”);

        . . .
```

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

A Closer Look . . .

GPIO

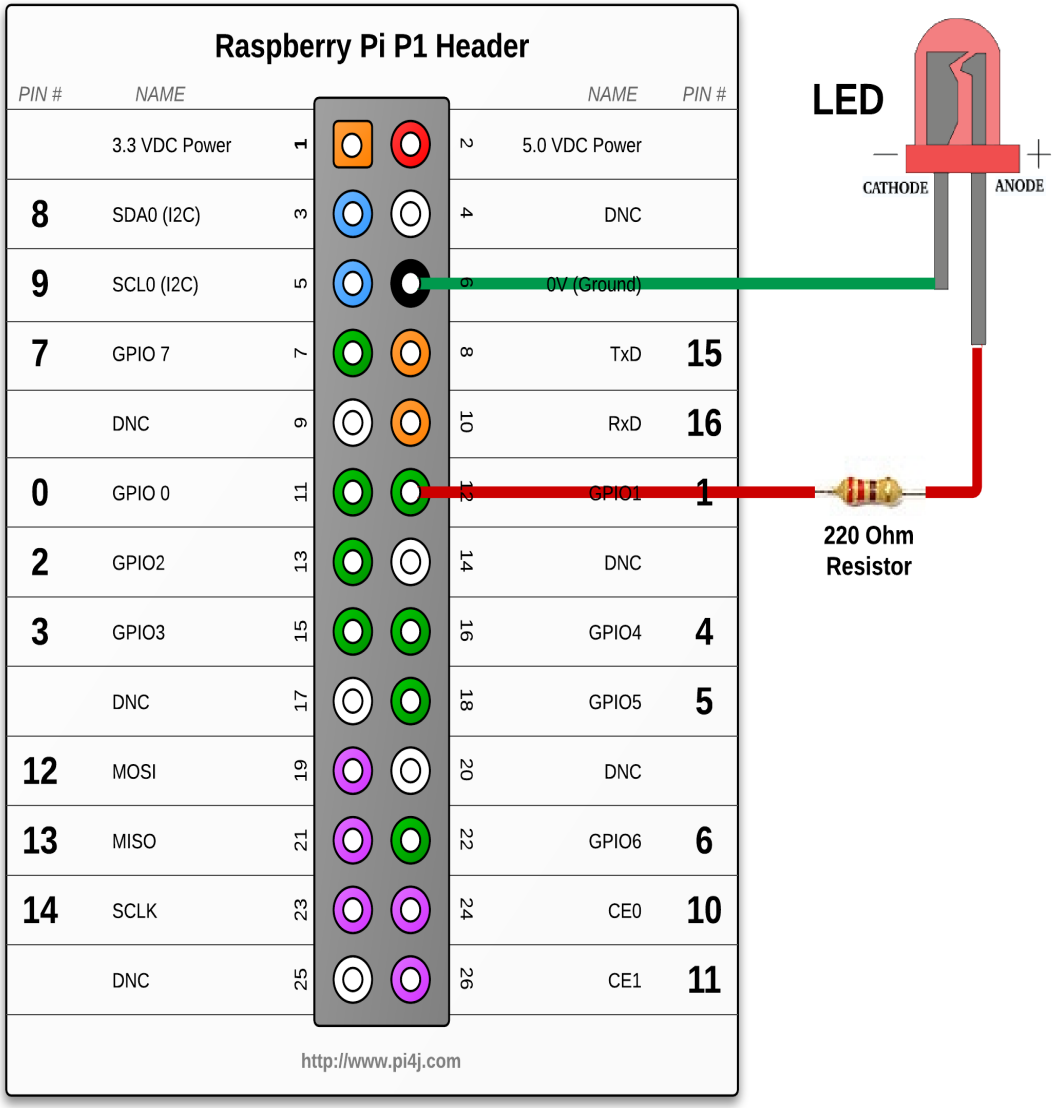
General Purpose Input Output

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Sample. . .

GPIO

General Purpose Input Output



Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

GPIO

- General Purpose Input/Output
- Logical 1 or 0 controlled by software
- Two wires (one for data, one for ground)
- Dedicated to a single purpose
  - Drive a single LED
  - Status flag
  - “bit-banging”

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

jdk.dio.gpio.GPIOPin

Key configuration details

- Pin number
- Direction
  - Input
  - Output
- Trigger
  - Rising
  - Falling
- Mode – Not software configurable for Linux/ARM port
- Represents a single GPIO pin
- Can be configured as input or output
  - Detect a button press
  - Drive a single LED
- Can register listeners to handle “value changed” events

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

jdk.dio.gpio.GPIOPin

```
GPIOPinConfig config =
              new GPIOPinConfig(DeviceConfig.DEFAULT,     // controller number
                                18,                       // pin number
                                GPIOPinConfig.DIR_OUTPUT_ONLY,
                                GPIOPinConfig.DEFAULT,    // mode (ignored)
                                GPIOPinConfig.TRIGGER_NONE,
                                false);                   // initial value

. . .
       GPIOPin outputPin = DeviceManager.open(config);
. . .
       outputPin.setValue(true);
```

```
GPIOPinConfig config =
              new GPIOPinConfig(DeviceConfig.DEFAULT,           // controller number
                                23,                             // pin number
                                GPIOPinConfig.DIR_INPUT_ONLY,
                                GPIOPinConfig.DEFAULT,
                                GPIOPinConfig.TRIGGER_RISING_EDGE |
                                       GPIOPinConfig.TRIGGER_FALLING_EDGE,
                                false);                         // initial value
. . .
       GPIOPin inputPin = DeviceManager.open(config);
. . .
       boolean pinValue = inputPin.getValue();

                 inputPin.setInputListener(new PinListener() {
                        public void valueChanged(PinEvent event) {
                               System.out.println("Pin value is now " + event.getValue());
                        }
                 });
```

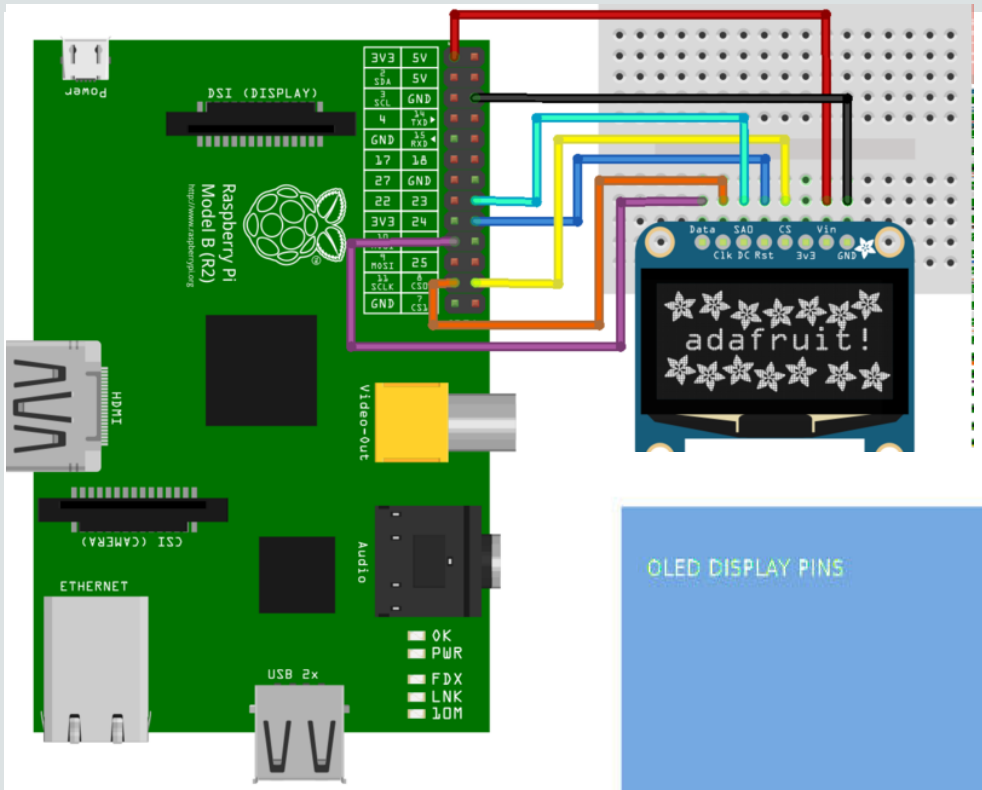
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

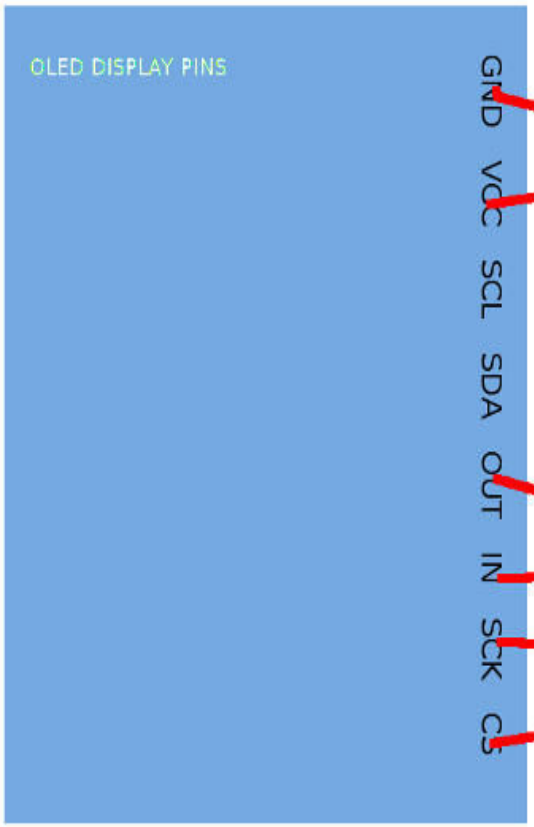
A Closer Look . . .

SPI

Serial Peripheral Interface

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.





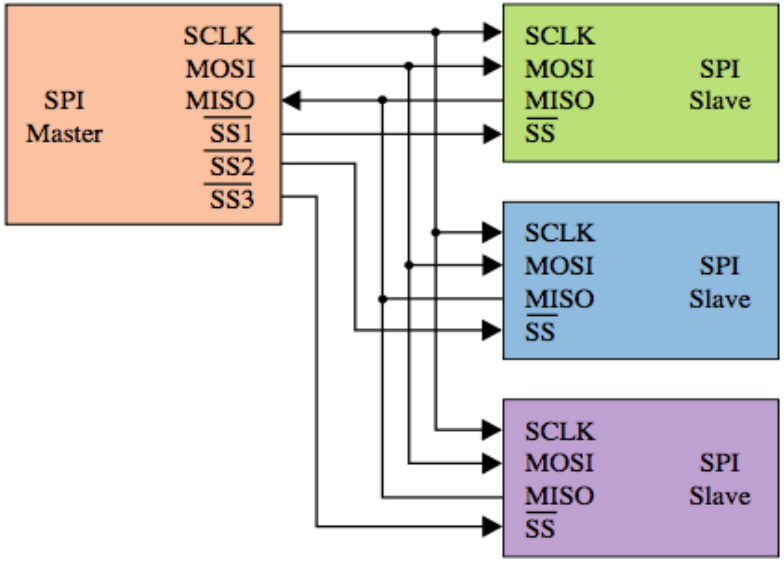
Samples. . .

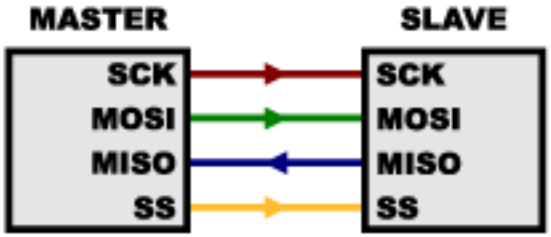
SPI

Serial Peripheral Interface

JavaOne ORACLE

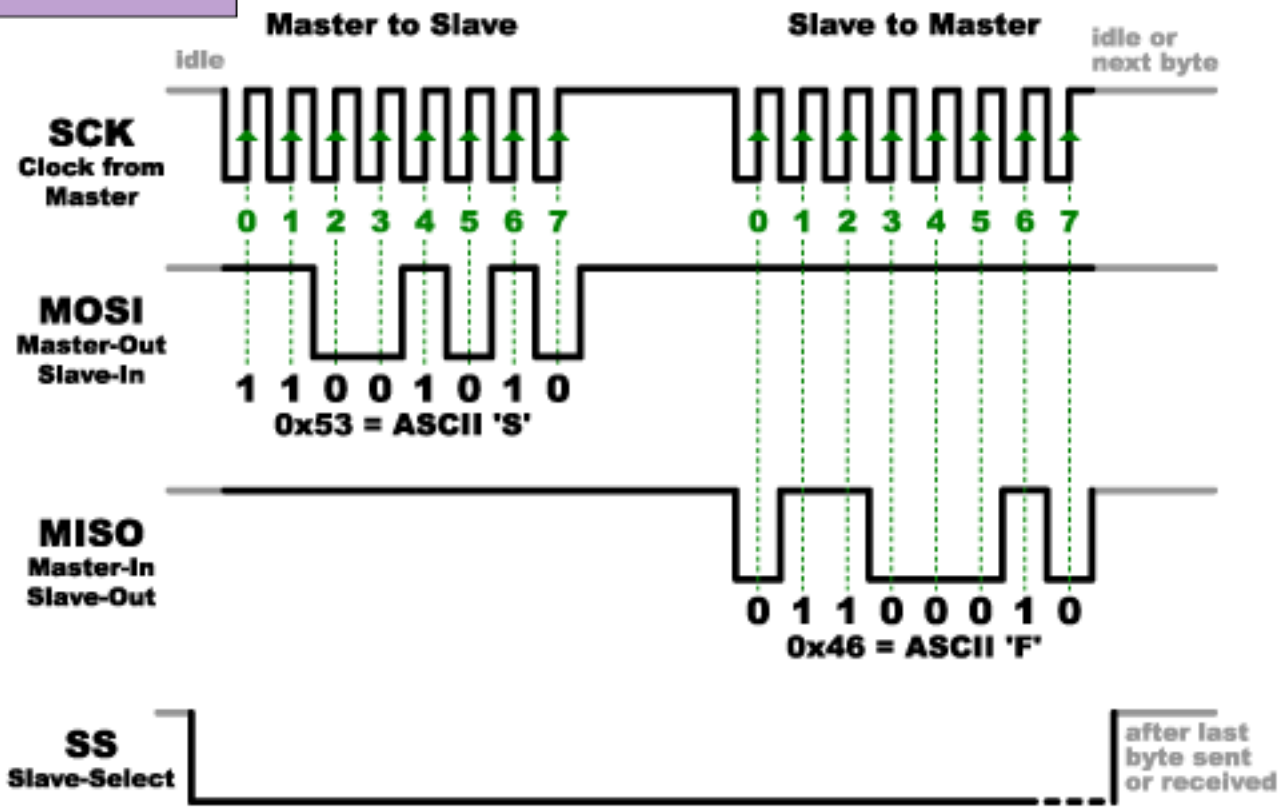
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.


SPI
Master
SCLK
MOSI
MISO
SS1
SS2
SS3
SCLK
MOSI
MISO
SS
SPI
Slave
SCLK
MOSI
MISO
SS
SPI
Slave
SCLK
MOSI
MISO
SS
SPI
Slave


MASTER
SLAVE
SCK
MOSI
MISO
SS
SCK
MOSI
MISO
SS

Samples. . .

SPI


Master to Slave
Slave to Master
idle
idle or
next byte
SCK
Clock from
Master
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7
MOSI
Master-Out
Slave-In
1 1 0 0 1 0 1 0
0x53 = ASCII 'S'
MISO
Master-In
Slave-Out
0 1 1 0 0 0 1 0
0x46 = ASCII 'F'
SS
Slave-Select
after last
byte sent
or received

SPI

- Serial Peripheral Interface
- Single master/multiple slaves connected to a single bus
- Serial, full-duplex
- Bits shift in on MISO (Master In Slave Out) as they shift out on MOSI (Master Out Slave In)

  Synchronous full duplex protocol because of the protocol and the 6 wires:
  - 4 (MISO – Master In Slave Out, MOSI – Master Out Slave In, SCK/SCLK – Clock, CS/SS/CE0 – Slave Select)
  - 2 wires for VCC (Voltage) and GND (Ground)

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

jdk.dio.spibus.SPIDevice

Key configuration details

- Device number
- Chip select address (device address)
- Chip select active level
  - High, low, not controlled
- Clock mode – see javadocs for explanation
- Word length
- Bit ordering

- Represents an SPI slave device
- Provides methods to write, read and writeAndRead to/from the slave device
  - write(); read(); != writeAndRead();
- Allows you to surround a series of writes and reads with begin(), end() to keep slave select line active
- Uses java.nio.ByteBuffer in API calls

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

jdk.dio.spibus.SPIDevice

```
SPIDeviceConfig config =
              new SPIDeviceConfig(DeviceConfig.DEFAULT,          // Device Number
                                                        0,          // SS connected to CE0
                                                        500000,     // clock frequency
                                                        SPIDeviceConfig.CS_ACTIVE_LOW,
                                                        8,          // 8-bit words
                                                        Device.BIG_ENDIAN);
 . . .
SPIDevice spiDevice = DeviceManager.open(config);
 . . .
```

MCP3008 Example

```
public int readChannel(int c) {
        ByteBuffer out = ByteBuffer.allocate(3);
        ByteBuffer in = ByteBuffer.allocate(3);
        out.put((byte)0x01);                                  // start bit
        out.put((byte)(((c | 0x08) & 0x0f) << 4));            // single-ended, channel c
        out.put((byte)0);                                     // padding
        out.flip();     // important!!! reset or flip buffer to start sending from
                        // the beginning
        . . .
            spiDevice.writeAndRead(out, in);
        . . .
        int high = (int)(0x0003 & in.get(1));    // first byte is padding, 10-bit result is
        int low = (int)(0x00ff &                 // contained in bit 1-0 of second byte and
        in.get(Linux);                           // all eight bits of third byte
        return (high << 8) + low;
    }
```

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

A Closer Look . . .

I^2C

Inter-Integrated Circuit

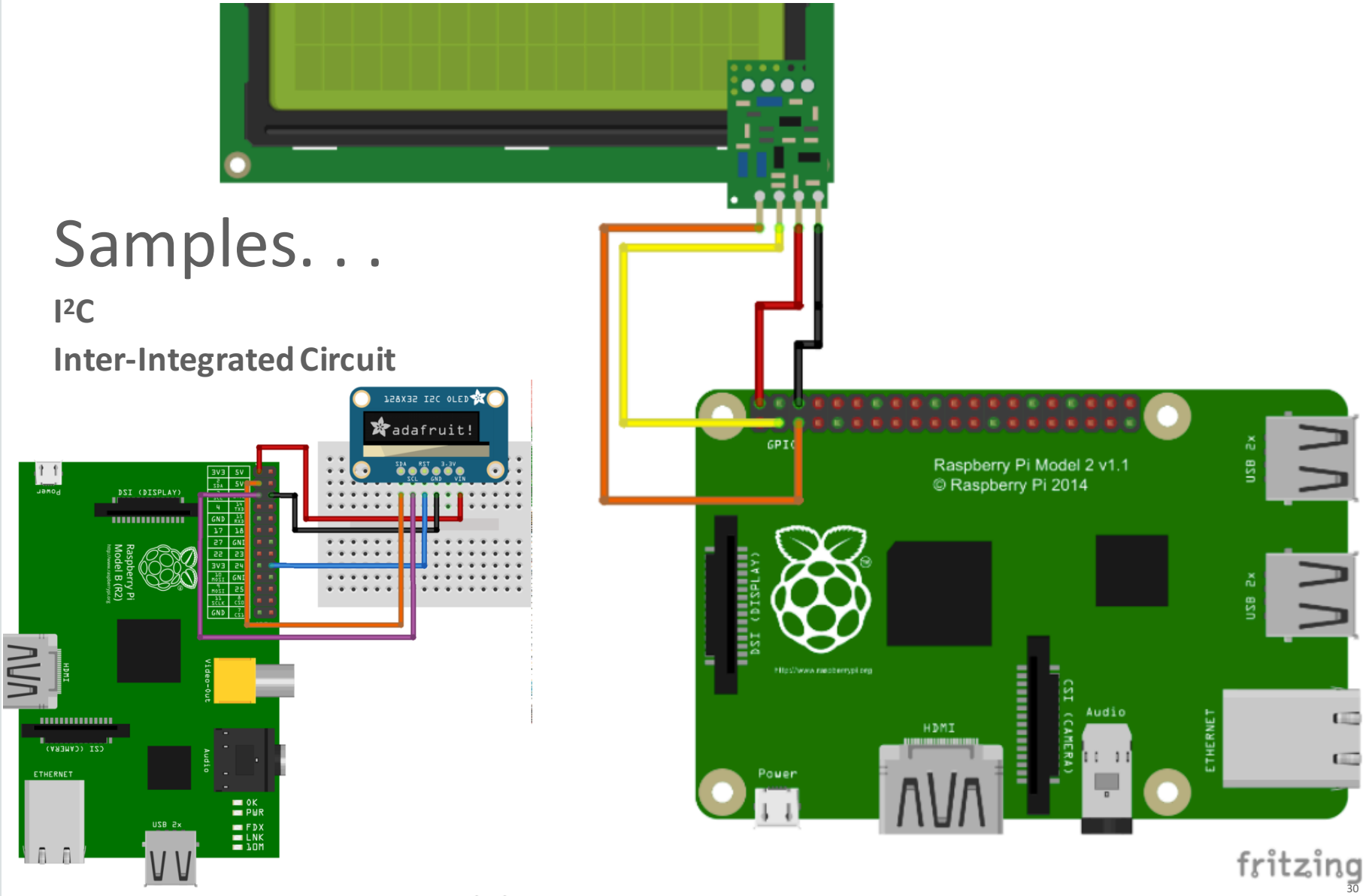
JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Samples. . .

I²C

Inter-Integrated Circuit



From left to right on the LCD are SCL (I2C Clock), SDA (I2C Data), VCC (+5v) and GND.

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

I²C

- Inter-Integrated Circuit
- Multi-master/multi-slave bus
  - Device I/O supports only slave devices
  - One master is assumed
- Serial, half-duplex because of the protocol and 4 or 5 wires:
    - 2 wires (SLC – I2C Clock, SDA – I2C Data)
    - 2 wires for VCC (Voltage) and GND (Ground)
    - 1 optional – RESET wire
- One line for data, one for clock, no separate address lines

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

jdk.dio.i2cbus.I2CDevice

Key configuration details

- Controller number
- Slave address
- Address size
- Clock frequency

- Represents a I2C slave device
- Provides methods to read, write from/to slave device
- Allows you to surround a series of related writes and reads with begin(), end()
- Uses java.nio.ByteBuffer in API calls

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

jdk.dio.i2cbus.I2CDevice

BMP160 Example

```
I2CDeviceConfig config =
                new I2CDeviceConfig(1,              // i2c bus number (raspberry pi)
                                    0x77,     // i2c slave address (BMP180 press/temp sensor)
                                    7,        // address size in bits
                                    3400000   // 3.4MHz clock frequency
                                    );
        . . .
I2CDevice i2cSlave = DeviceManager.open(config);
        . . .
// read calibration data
ByteBuffer dst = ByteBuffer.allocate(2Linux;// 22 = size (bytes) of calibration
data
int bytesRead = i2cSlave.read(0xAA,      // EEPROM start address
                              1,         // size (bytes) of subaddress
                              dst);
```


JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

More Info DIO

- Device I/O OpenJDK Project page
  - http://openjdk.java.net/projects/dio/
- Device I/O mailing list
  - http://mail.openjdk.java.net/mailman/listinfo/dio-dev
- Device I/O Wiki
  - https://wiki.openjdk.java.net/display/dio/Main
- Device I/O mercurial repo
  - http://hg.openjdk.java.net/dio/dev

JavaOne ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Overview of technical issues

Too much information?

This was **Section 1 – Internet of Things**

- IoT Clouds
  - AMAZON
  - ORACLE
  - IBM
  - MICROSOFT
- IoT Communications Protocols
  - REST-HTTP
  - MQTT
  - CoAP
- IoT Gateway Programming: Java Device Input Output (DIO) | http://elinux.org/RPi_GPIO_Code_Samples
  - UART
  - SPI
  - I2C
  - GPIO
  - Wireless: ZigBee/Zwave
- IoT Nodes Programming: Arduino C Lang for Arduino or Intel Galileo/ESP Lua
  - Analogic/Digital Serial Connectivity

There are a lot of Embedded, Gateways, MOBILE devices, technologies, concepts and APIs/SDKs.

+ CRYPTO SECURITY

Overview of technical issues

Too much information?

This was **Section 1 – Internet of Things**

- IoT Clouds
  - AMAZON
  - ORACLE
  - IBM
  - MICROSOFT
- IoT Communications Protocols
  - REST-HTTP
  - MQTT
  - CoAP
- IoT Gateway Programming: Java Device Input Output (DIO) | http://elinux.org/RPi_GPIO_Code_Samples
  - UART
  - SPI
  - I2C
  - GPIO
  - Wireless: ZigBee/Zwave
- IoT Nodes Programming: Arduino C Lang for Arduino or Intel Galileo/ESP Lua
  - Analogic/Digital Serial Connectivity

There are a lot of Embedded, Gateways, MOBILE devices, technologies, concepts and APIs/SDKs.

+ CRYPTO SECURITY

Hardware, OS Boot-loader, Kernel, Drivers, System Calls, File-systems, IPC – Inter-Process Communication, Applications, User & Kernel Space, Assembly

Hands-on: NASM – x86 16 bits (Mike OS), IPC, ARM Assembly Intro & GNU ARM ASM GCC for OS Dev for Raspberry Pi(Cambridge ARM RPi OS)

Embedded OS

It’s not just about the ideas, but technologies, architectures & security

Internet of Things using Embedded Devices

What about the eMbedded Devices

Requirements & Features?

They are not new.

ICT | Cyber Security Master

OS Security

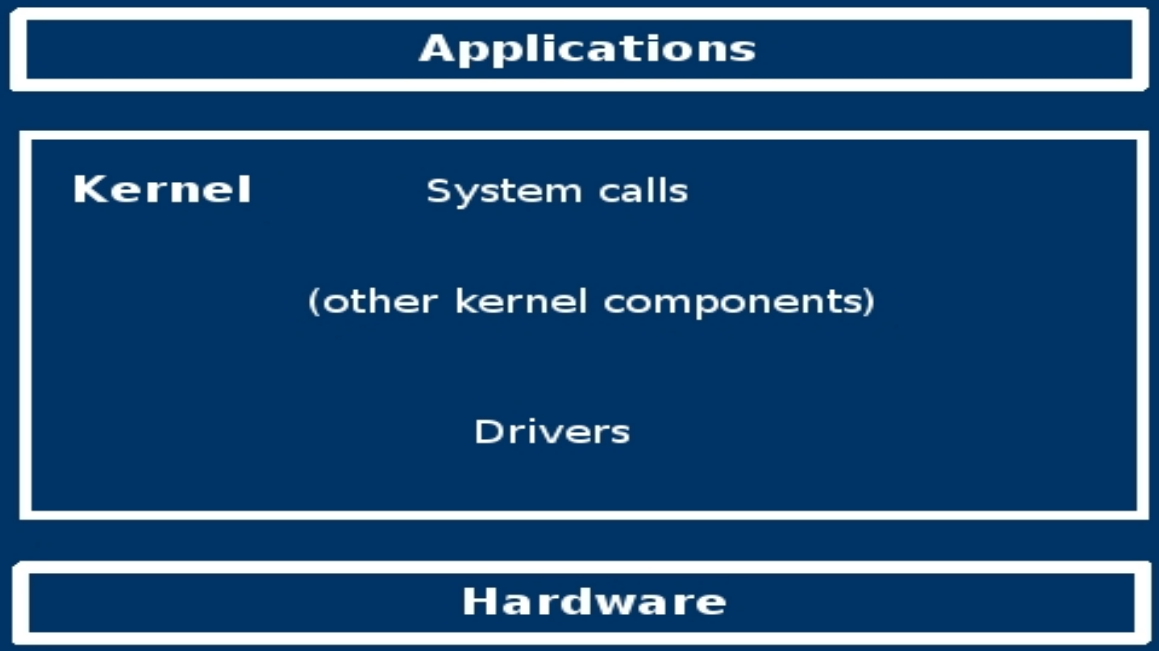
George Iosif
giosif@gmail.com

Embedded OS Details

- Hardware
- OS Boot-loader (Hands-on: Assembly NASM x86 16 bits for MikeOS)
- OS Kernel (Hands-on: Assembly NASM ARM 32 bits for Cambridge RPi OS)
- Drivers
- System calls
- Applications
- IPC – Inter-Process Communication (Hands-on)

Embedded OS Details

OS architecture overview



Embedded OS Details

- Hardware
  - physical electronic components
  - dedicated function
  - mainly driven by software running in the OS (there are some exceptions)
  - they communicate with other (specific) components through (electric) signals
  - examples: CPU, motherboard, memory, I/O devices, storage devices (hard disk, CD, DVD,...), network adapters, etc.

Embedded OS Details

- Kernel
  - the first "program" loaded by the BIOS/boot loader
  - its functionality (the way it works) is very closely related to the CPU architecture
  - represents the "glue layer" which provides the environment necessary for the applications to run on the given hardware
  - responsible for the management (initialisation, utilisation << incl. protection >>, deallocation) of all computer resources (examples of resources: CPU, memory, storage devices, etc.)
  - responsible for the management of all processes

Embedded OS Details

- Kernel (cont.)
  - there are two main types of kernel architectures:
    - monolithic – all the parts of the kernel execute in the same (kernel) address space
      - its capability is extensible through modules (drivers). Once these modules (drivers) are loaded, they become part of the running kernel (they run in the kernel address space).
      - advantages: speed
      - disadvantages: stability and security
    - microkernel – only a few essential parts of the kernel execute in the kernel address space, the rest are running in user space as programs called “servers”
      - the communication between different parts of the kernel happens through IPC (Inter-Process Communication) mechanisms
      - advantages: stability, maintainability and security
      - disadvantages: speed, ease of implementation

Embedded OS Details

- Driver(s)
  - part of the kernel, mainly responsible for the management of hardware
  - contains device specific code
  - communicates with the device through IRQs, I/O ports and DMA channels.
  - main source of system instability

Embedded OS Details

- System calls
  - kernel code which permits user-space applications to use kernel-space functions/services in a legitimate way
  - examples: file read/write, memory allocation, etc.
  - it generally interfaces with applications through system libraries
  - some of them are “intercepted” by antivirus software so it (the antivirus) is able to perform “on-access” scanning

Embedded OS Details

- Applications
  - programs that run outside the kernel and directly or indirectly provide services to the user (be it an administrator, end-user, hacker)
  - when running, their entire “life” is strictly managed by the kernel
  - examples: IoT Clients SW, IoT GW SW, HTTP server, shell, IM client, most viruses, etc.

Linux OS security levels

- The boot process
- The kernel
- Processes & memory
- User system
- The filesystem
- Networking
- General (DAC vs MAC)

Linux OS security aspects

- The boot process
  - represents the process of loading the kernel into memory and passing control to it
  - for Linux on the IBM PC/IA 32 architecture, it can be broken into six logical stages*:
    1. BIOS selects the boot device
    2. BIOS loads the bootsector/bootloader from the boot device
    3. Bootsector/bootloader loads setup, decompression routines and compressed kernel image
    4. The kernel is uncompressed in protected mode
    5. Low-level initialisation (asm code)
    6. High-level initialisation (C code)

* http://www.moses.uklinux.net/patches/lki.html
http://www-128.ibm.com/developerworks/linux/library/l-linuxboot/index.html
http://www-128.ibm.com/developerworks/linux/library/l-bootload.html

Linux OS security aspects

- The boot process
  - security considerations:
    - boot viruses (ancient)
    - boot device override
    - OS override
    - kernel parameters override

Linux OS security aspects

- The kernel
  - presentation

  command: *'$ dmesg | less'*
  - security considerations:
    - due to the key role of the kernel, security compromises at this level have the greatest impact and are the hardest to detect (although rare and somewhat harder to exploit)
    - examples of security vulnerabilities: buggy kernel code, bad drivers, kernel architecture (modules ⟺ rootkits)

Linux OS security aspects

- Processes & memory
  - Process = program instance loaded in memory and running with its own dedicated address space and state information
  - presentation

  command: *'$ ps -e -o pid,ppid,uid,gid,ni,stat,time,%cpu,%mem,command | less'*
  - main characteristics:
    - PID = Process ID
    - PPID = Parent Process ID
    - UID = User ID of the process
    - GID = Group ID of the process
    - ... (next page)

Linux OS security aspects

- Processes & memory
  - main characteristics (cont.):
    - NI = process' NIce value
    - STAT = process STATe
    - TIME = cumulative CPU TIME used by this process
    - %CPU = CPU time used / the time this process has been running
    - %MEM = % of memory used by this process
    - COMMAND = the command line used to start the process
  - process execution control – signals[2]
    - Signal = a limited form of interprocess communication used to (asynchronously) notify a specific process that an event had occurred

Linux OS security aspects

- Processes & memory
  - process execution control – signals (cont.)
    - sending signals:
      - keyboard input

      command: *'$ top'*
      keyboard input: CTRL+z
      command: *'$ ps aux'* (observe the STAT value)
      command: *'$ fg'*
      keyboard input: CTRL+c
      - kill command

      command (initial session): *'$ kill -l'* (observe the list of signals)
      command (initial session): *'$ top'*
      command (from a 2nd session): *'$ ps aux'* (take note of *top*'s PID)
      command (from the 2nd session): *'$ kill -19 <<PID_of_top>>'*
      (observe the process' behaviour on the initial session)
      command (initial session): *'$ fg'*
      command (from the 2nd session): *'$ kill -2 <<PID_of_top>>'*
      (observe the process' behaviour on the initial session)

Linux OS security aspects

- Processes & memory
  - process execution control – signals (cont.)
    - sending signals (cont.):
      - exceptions – examples: division by zero, segmentation violation
      - ***kill*** system call – used between different processes
      - kernel
    - handling signals:
      - through signal handlers – code which deals with the situation that generated the signal
      - if no custom handler is present for a specific signal, a default handler is used
      - if no custom handler is present, for some signals, a process can ignore the signal or use the default handler
      - there are two signals which cannot be intercepted and handled: SIGKILL and SIGSTOP

Linux OS security aspects

- Processes & memory
  - process capabilities[4]
    - for the purpose of performing permission checks, traditional Unix implementations distinguish two categories of processes:
      - privileged processes – their UID is 0 and bypass all kernel permission checks
      - unprivileged processes – their UID is different than 0 and they are subject to full permission checking based on the process' credentials (UID, GID and supplementary group list)
    - in order to provide a more granular set of permissions, the privileges traditionally associated with the superuser were divided into distinct units, known as capabilities, which can be independently enabled and disabled.

Linux OS security aspects

- Processes & memory
  - process capabilities (cont.)
    - capabilities are a per-thread attribute
      Thread = code which runs independently, but which was created by a parent (they are both part of the same program) with which it shares a common address space and state information
    - generally, a thread can only drop capabilities and, once dropped, it cannot regain them
    - the effective capabilities of a process are determined by performing a logical AND between its designed capabilities and a system-wide value called *capability bounding set* (/proc/sys/kernel/cap-bound).
      Only the **init** process may set bits in the capability bounding set and the superuser may only clear bits in this set.

Linux OS security aspects

- Processes & memory
  - process capabilities (cont.)
    - a full implementation of capabilities requires:
      - that for all privileged operations, the kernel checks whether the process has the required capability in its effective set
      - that the kernel provides system calls allowing a process' capability sets to be changed and retrieved
      - file system support for attaching capabilities to an executable file, so that a process gains those capabilities when the file is executed

      Currently, only the first two of these requirements are met, the third being in the process of implementation.
    - examples: CAP_DAC_OVERRIDE, CAP_FSETID, CAP_NET_ADMIN, etc.
    - Q: Use ?

      A: A process/thread should drop all capabilities which are not necessary for its designated use, thus limiting the impact of any possible abuse of the given program. (for more see DAC vs MAC)

Linux OS security aspects

- Processes & memory
  - memory area (stack / heap) access / execution
    - a way to change the normal execution path of a given process by making it to illegally access memory areas
    - it achieves this mainly by exploiting programming mistakes
    - examples: buffer overflow exploits, viruses
    - solutions:
      - non-executable user stack area

      "Most buffer overflow exploits are based on overwriting a function's return address on the stack to point to some arbitrary code, which is also put onto the stack. If the stack area is non-executable, buffer overflow vulnerabilities become harder to exploit."*

* http://www.openwall.com

Linux OS security aspects

- Processes & memory
  - memory area (stack / heap) access / execution (cont.)
    - solutions (cont.):
      - executable space protection

      "In computer security, executable space protection is the marking of memory regions as non-executable, such that an attempt to execute machine code in these regions will cause an exception. It often makes use of hardware features such as the NX/XD bit. Implementations for Linux include PaX, Exec Shield, and Openwall."[2]
      - address space layout randomization (ASLR)

      "The generic idea behind this approach is based on the observation that in practice most attacks require advance knowledge of various addresses in the attacked task. If we can introduce entropy into such addresses each time a task is created then we will force the attacker to guess or brute force it which in turn will make the attack attempts quite 'noisy' because any failed attempt will likely crash the target. It will be easy then to watch for and react on such events."*

      * http://pax.grsecurity.net

Linux OS security aspects

- User system
  - necessary in a multiuser environment in order to be able to set individual characteristics (related to the specified environment) and to uniquely identify each person
  - consists of a database containing user and group information (along with credentials and other data) and a set of tools to manage that database
  - examples (Linux):
    - user and group information is stored in the following files: /etc/passwd, /etc/shadow, /etc/group, /etc/gshadow
    - tools:
      useradd, userdel, passwd, groupadd, gpasswd, vipw, etc.

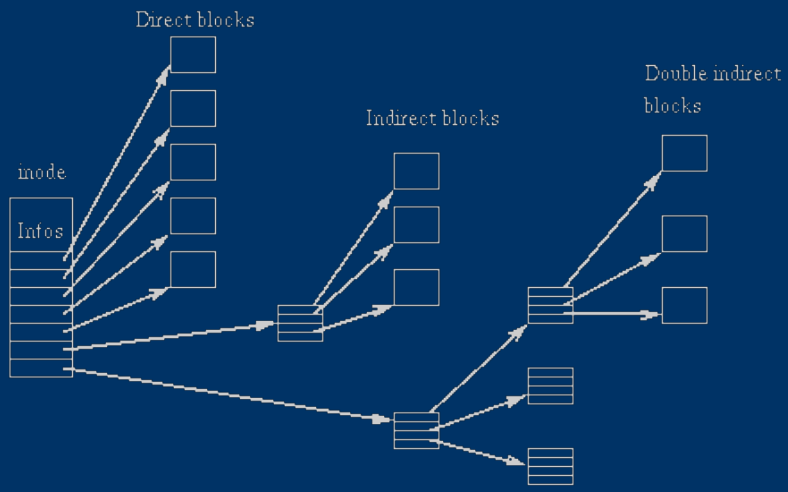
Linux OS security aspects

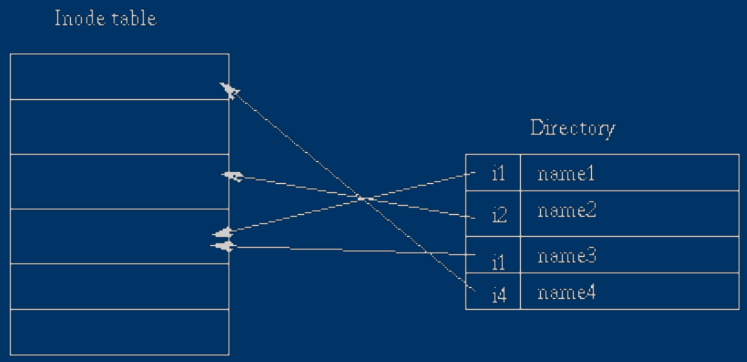
- The filesystem[5][6][7]
  - represents a method for organizing data in logical elements and providing ways to manage that data
  - main concepts (for ExtLinux:
    - superblock - contains information about the filesystem as a whole, such as its size, the number of free blocks, free inodes, logical block size, the number of times the volume has been mounted, and other accounting information about the filesystem
    - inode - contains all information about a file, except its name

    The inode contains the numbers of several data blocks, which are used to store the data in the file. There is space only for a few data block numbers in the inode, however, and if more are needed, more space for pointers to the data blocks is allocated dynamically
    - direct data block - data block that is accessed directly with the information from the inode

Linux OS security aspects

- The filesystem
  - main concepts (cont.):
    - indirect data block - data block that is accessed with the information residing on another data block
    - directory - a special file which contains directory entries
    - directory entry - contains the name of the file and the inode number which represents that file



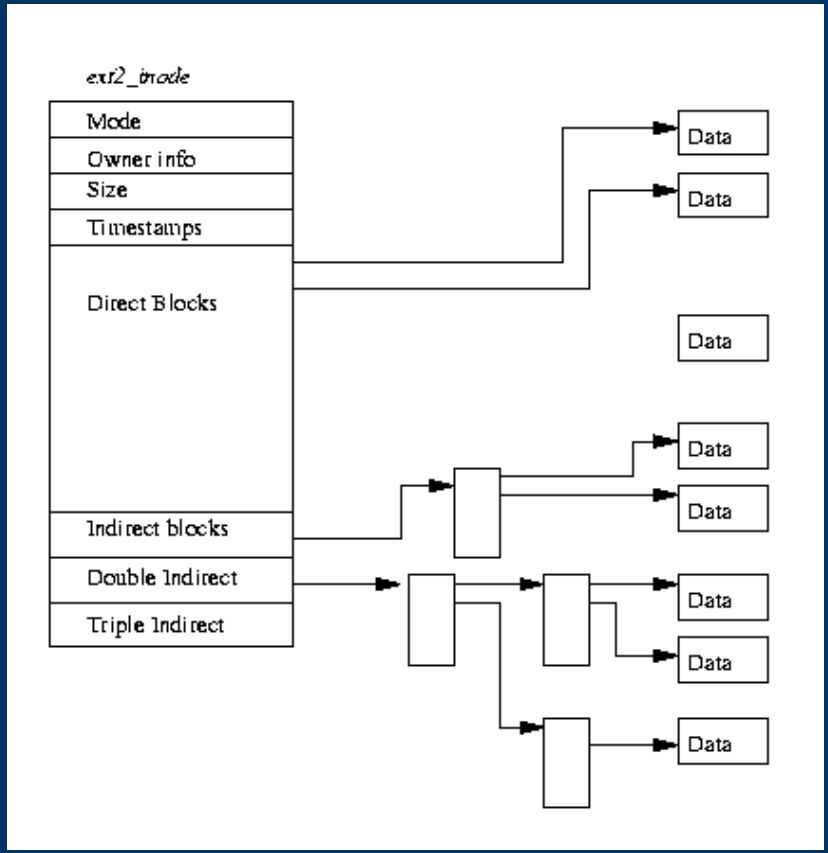


Linux OS security aspects

- The filesystem
  - main concepts (cont.):
    - detailed view of an Ext2 inode
  - examples:

command: *'# dumpe2fs /dev/sda1 | less'*

command: *'$ ls -li'*



Linux OS security aspects

- The filesystem
  - file types:
    - regular file
    - directory
    - device file (block and character)
    - symbolic link
    - socket
    - Examples
  - access rights:
    - traditional access - composed of rights for the file user owner, group owner and others
      - rights:
        - **r** - for a file, it means the right to read the contents of that file; for a directory, it means the right to read the contents (files / subdirectories) of that directory

Linux OS security aspects

- The filesystem
  - access rights (cont.):
    - traditional access (cont.)
      - rights (cont.):
        - **w** - for a file, it means the right to write and modify the content of that file; for a directory, given that the user also has **x** rights on that directory, it means the right to create, rename and delete files / subdirectories in that directory
        - **x** - for a file, it means that the file can be executed; for a directory, it means that the user can enter that directory
        - **SUID** - for an executable file, if run, it means that the new process has the UID of the owner of the file, no matter who ran the file; for a directory, it has no effect
        - **SGID** – for an executable file, if run, it means that the new process has the GID of the group owner of the file, no matter who ran the file; for a directory, it means that all new files / subdirectories created in and below it will have the group owner of that directory, no matter who creates the new files / subdirectories (given it has the right permissions to do so)

Linux OS security aspects

- The filesystem
  - access rights (cont.)
    - traditional access (cont.)
      - rights (cont.):
        - **t (sticky bit)** - for a file, it has no effect; for a directory, all files / subdirectories created in or below it can be deleted or renamed only by the owner of those files / subdirectories or by the superuser (root)
      - observation - in setting access rights, two notations can be used to specify them:
        - text - presented above (examples: r,w,x,etc.)
        - decimal numbers (powers of Linux - r=4,w=2,x=1,SUID=4,SGID=2,t=1
      - examples:

command: *'$ touch some_file'*
command: *'$ ls -l'*
(observe the output of the last command)
'-rw-r--r-- 1 giosif users 0 2006-10-24 15:28 some_file'

Linux OS security aspects

- The filesystem
  - access rights (cont.)
    - traditional access (cont.)
      - examples (cont.):

command: *'$ chmod 674 some_file'*
command: *'$ ls -l'*
(observe the output of the last command)
'-rw-rwxr-- 1 giosif users 0 2006-10-24 15:28 some_file'
command: *'$ chmod g-w some_file'*
command: *'$ ls -l'*
(observe the output of the last command)
'-rw-r-xr-- 1 giosif users 0 2006-10-24 15:28 some_file'
command: *'$ chmod 4754 some_file'*
command: *'$ ls -l'*
(observe the output of the last command)
'-rwsr-xr-- 1 giosif users 0 2006-10-24 15:28 some_file'
command: *'$ ls -l /usr/bin/passwd'*
(note the fact that **passwd** has SUID bit set)

Linux OS security aspects

- The filesystem
  - access rights (cont.)
    - ACLs (Access Control List)
      - allow setting permissions at a more fine-grained level than the traditional permissions system
      - each file / directory has associated a list with users and groups along with their permissions on that file / directory
      - is backward compatible with the traditional permissions system
      - examples:

      command: *'$ getfacl some_file'*
      (observe the output of the last command)
      command: *'$ setfacl -m u:root:rx some_file'*
      command: *'$ getfacl some_file'*
      (observe the output of the last command)
    - security considerations

Linux OS security aspects

- Networking
  - a network represents two or more devices, together with a method of communication between them, allowing each to share and access resources with the other
  - the entire communication is based on the concept of *protocol* = a set of rules to which both communicating parties agree in order to understand each other
  - also, to reduce complexity and provide flexibility, a network is organized in layers, one (layer) sitting on top of another, each using the services provided by the underneath layer and providing specific services to the layer above

Linux OS security aspects

- Networking
  - between two communicating parties, the communication takes place between the layers found at the same level (example: the 3rd layer from one host communicates with 3rd layer from the other host)
  - the protocol concept mentioned above is used per layer (in order for two parties to communicate successfully, the corresponding layers found at the same level have to use the same protocol = use the same set of rules when communicating)
  - the group of protocols determined by all layers, each with its protocol, forms a protocol stack

Linux OS security aspects

- Networking
  - reference model - defines (for a networking system) the number of layers, the services they provide and the protocols used
  - the most known reference models are: ISO/OSI and TCP/IP
  - ISO/OSI reference model
    - represents the theoretical approach in designing a networking system. To this respect, it is the most complete, but, in practice, it wasn't very well adopted because it's harder to implement and also came later than TCP/IP.
    - it's made out of 7 layers:

Linux OS security aspects

- Networking
  - ISO/OSI reference model (cont.)

Linux OS security aspects

- Networking
  - ISO/OSI reference model (cont.)
    - layers description[2]:
      - The Physical layer defines all the electrical and physical specifications for devices.
      - The Data Link layer provides the functional and procedural means to transfer data between network entities and to detect and possibly correct errors that may occur in the Physical layer.
      - The Network layer provides the functional and procedural means of transferring variable length data sequences from a source to a destination via one or more networks while maintaining the quality of service requested by the Transport layer.
      - The Transport layer provides transparent transfer of data between end users, thus relieving the upper layers from any concern while providing reliable data transfer.
      - The Session layer controls the dialogues (sessions) between computers.

Linux OS security aspects

- Networking
  - ISO/OSI reference model (cont.)
    - layers description (cont.):
      - The Presentation layer transforms data to provide a standard interface for the Application layer.
      - The Application layer provides a means for the user to access information on the network through an application.

Layer	Function
Application	Network process to application
Presentation	Data representation and encryption
Session	Interhost communication
Transport	End-to-end connections and reliability
Network	Path determination and logical addressing (IP)
Data link	Physical addressing (MAC & LLC)
Physical	Media, signal and binary transmission

Linux OS security aspects

- Networking
  - TCP/IP reference model
    - the *de facto* standard, developed by the US DoD (Department of Defence)
    - has fewer (4 or 5) layers than the ISO/OSI reference model[2]:
      - The Network Access layer describes the physical equipment necessary for communications, such as twisted pair cables, the signalling used on that equipment, and the low-level protocols using that signalling.
      - The Internet or Internetworking layer defines IP addresses, with many routing schemes for navigating packets from one IP address to another.
      - The Host-To-Host (Transport) layer deals with opening and maintaining connections, ensuring that packets are in fact received.
      - At the Process layer or Application layer the "higher level" protocols such as SMTP, FTP, SSH, HTTP, etc. operate.

Linux OS security aspects

- Networking
  - TCP/IP protocol stack
    - represents the foundation of today's Internet
    - currently at version 4 (TCP/IPv4)
    - almost all modern OSs (Operating Systems) include a TCP/IP stack
    - the 4 layers[2]:

4. Application	DNS, TFTP, TLS/SSL, FTP, HTTP, IMAP, IRC, NNTP, POP3, SIP, SMTP, SNMP, SSH, TELNET, ECHO, BitTorrent, RTP, PNRP, rlogin, ENRP, …
	Routing protocols like BGP, which for a variety of reasons run over TCP, may also be considered part of the application or network layer.
3. Transport	TCP, UDP, DCCP, SCTP, IL, RUDP, …
2. Internet	Routing protocols like OSPF, which run over IP, are also be considered part of the network layer, as they provide path selection. ICMP and IGMP run over IP are considered part of the network layer, as they provide control information.
	IP (IPv4, IPv6)
	ARP and RARP operate underneath IP but above the link layer so they belong somewhere in between.
1. Network access	Ethernet, Wi-Fi, token ring, PPP, SLIP, FDDI, ATM, Frame Relay, SMDS, …

Linux OS security aspects

- Networking
  - TCP/IP protocol stack (cont.)
    - the layer abstraction is done by using encapsulation (a lower layer encapsulates information from the upper layer)
    - the main protocols: Ethernet, IP (Internet Protocol), TCP (Transport Control Protocol), UDP (User Datagram Protocol)
    - the main concepts used: MAC address, IP address, TCP / UDP port, packet, connection
    - example: Wireshark demo
  - network services
    - represent applications that provide services that can be accessed over the network
    - examples: HTTP service, SMTP service, DNS service, etc.
    - they are the main targets for network attacks

Linux OS security aspects

- Networking
  - network services (cont.)
    - security issues - bad configuration, application vulnerabilities, authentication mechanisms vulnerabilities, too many privileges
  - types of attack:
    - protocol weaknesses exploitation (arp poisoning, IP spoofing, DHCP takeover)
    - traffic analysis (sniffing)
    - MITM (Man-In-The-Middle)
    - DoS (Denial of Service)
    - services vulnerabilities exploitation
  - attack vector - the succession of actions an attacker performs in order to reach his objective

Linux OS security aspects

- Networking
  - network security tools[8]:
    - filters
      - firewalls (iptables, Windows firewall)
      - IDSs / IPSs (Intrusion Detection / Prevention Systems) (snort)
    - scanners
      - nmap, nessus
    - sniffers
      - tcpdump, wireshark (former Ethereal), ettercap, kismet

Linux OS security aspects

- General (DAC vs MAC)
  - access control - the rules used to determine what actions a subject can perform on an object
  - DAC (Discretionary Access Control)
    - represents "a means of restricting access to objects based on the identity of subjects and/or groups to which they belong. The controls are discretionary in the sense that a subject with a certain access permission is capable of passing that permission (perhaps indirectly) on to any other subject (unless restrained by mandatory access control)."[9]
    - "DACs are specified by the owner of an object, who can apply, modify, or remove them at will."[10]
    - examples: a user has full permissions over the files he owns, a process inherits the permissions of the user that launched it

Linux OS security aspects

- General (DAC vs MAC)
  - DAC (Discretionary Access Control) (cont.)
    - offers limited auditing capabilities
    - used in almost all current OSes
  - MAC (Mandatory Access Control)
    - represents "a means of restricting access to objects based on the sensitivity (as represented by a label) of the information contained in the objects and the formal authorization (i.e., clearance) of subjects to access information of such sensitivity."[9]
    - "MACs are specified by the system. They cannot be applied, modified, or removed - except perhaps by means of a privileged operation."[10]
    - it is based on the principle of least privilege and on the idea that "that which is not expressly permitted, is denied"

Linux OS security aspects

- General (DAC vs MAC)
  - MAC (Mandatory Access Control) (cont.)
    - provides extended auditing capabilities
    - implementations:
      - Linux: SELinux, AppArmor
      - FreeBSD: TrustedBSD
      - Solaris: Trusted Solaris
    - SELinux
      - developed primarily by the US National Security Agency
      - included in Red Hat Enterprise Linux and Fedora Core
      - main concepts:
        - subjects, objects and actions
        - security classes (for objects) - define the types of objects present on a system

Linux OS security aspects

- General (DAC vs MAC)
  - MAC (Mandatory Access Control) (cont.)
    - SELinux (cont.)
      - main concepts (cont.):
        - security attributes (for subjects and objects):
        i. user identity
        - denotes SELinux User who created the object
        i. role
        - file objects all labeled “object_r”
        - process Objects include Role
        i. type
        - most used section of security context
        - type enforcement rules revolve around this field
        - security context - the three security attributes mentioned make a security context

Linux OS security aspects

- General (DAC vs MAC)
  - MAC (Mandatory Access Control) (cont.)
    - SELinux (cont.)
      - main concepts (cont.):
        - security policy - the set of rules loaded and enforced by the kernel on the base of which all access decisions are made

        It is configured by the administrator but enforced by the system.

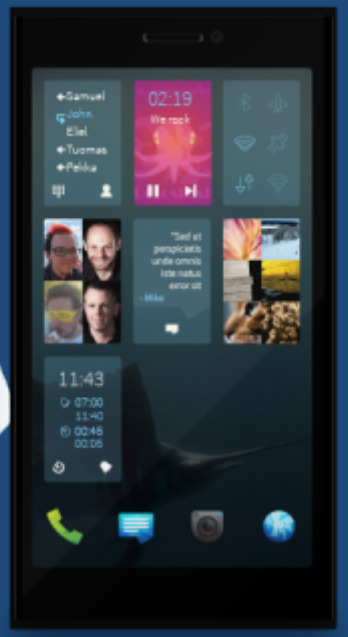
Resources

[1] IBM developerWorks: http://www-128.ibm.com/developerworks/
[2] Wikipedia: http://www.wikipedia.org
[3] TLDP: http://www.tldp.org
[4] Linux on-line manuals (man capabilities)
[5] FreeOS.com: http://www.freeos.com/articles/4015/
[6] Ext2 intro: http://web.mit.edu/tytso/www/linux/ext2intro.html
[7] LinuxHQ.com:http://www.linuxhq.com/guides/TLK/fs/filesystem.html
[8] sectools.org: http://sectools.org/
[9] TCSEC: http://www.radium.ncsc.mil/tpep/library/rainbow/5200.28-STD.html
[10] O'Reilly, Bill McCarty – SELinux – NSA's Open Source Security Enhanced Linux

2D Barcode URL

http://acs.ase.ro | http://www.dice.ase.ro

Scan the Tag
to get the web
Mobile Address



Thank you!