PRISMTECH

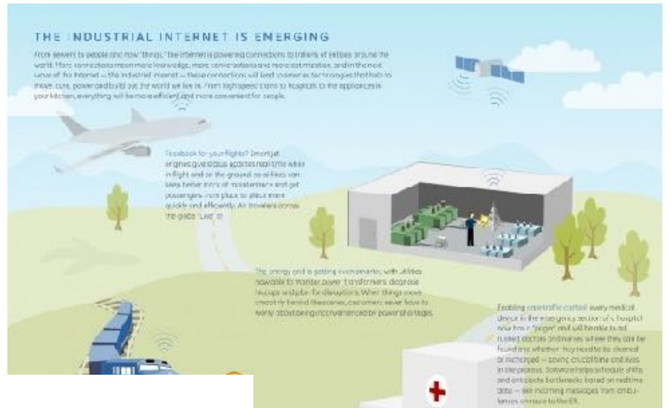
AMQP, CoAP, MQTT and DDS

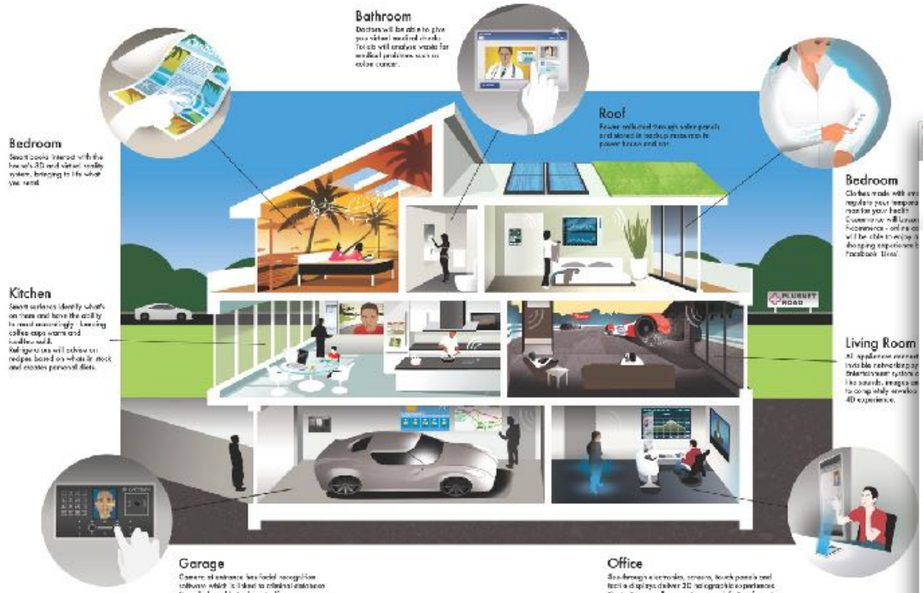
Angelo Corsaro, PhD
Chief Technology Officer
angelo.corsaro@prismtech.com

The IoT Data Pipeline

Collect | Store | Analyse | Share

- Collect the data from the cyber-physical world
- Depending on applications this could be:
  - Sensor data
  - Market Data
  - Web page statistics
  - …

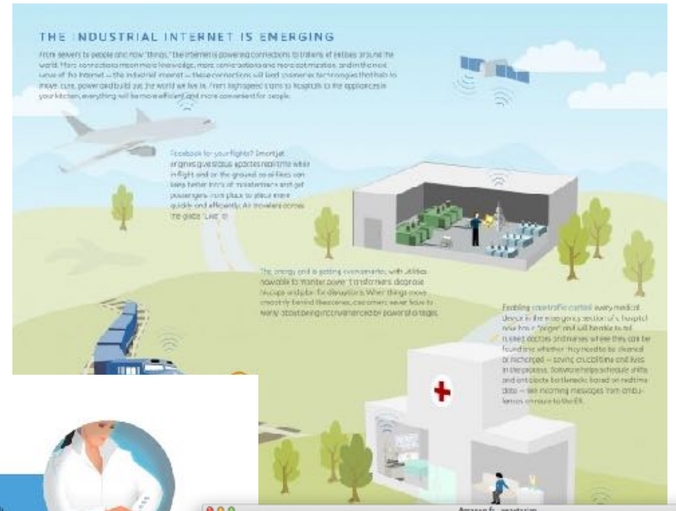


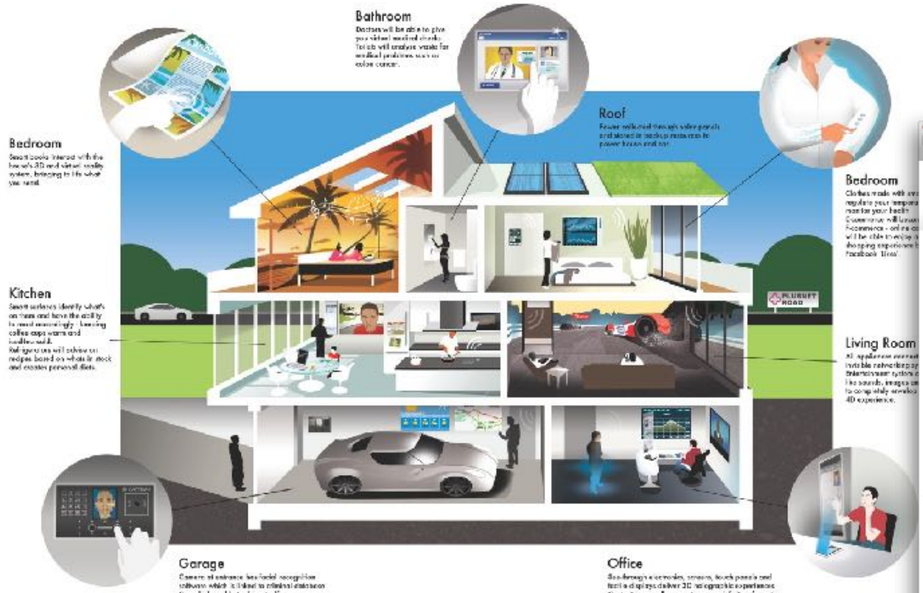


PRISMTECH

Collect | **Store** | Analyse | Share

- Organise , Store/Cache the data for on-line and off-line processing

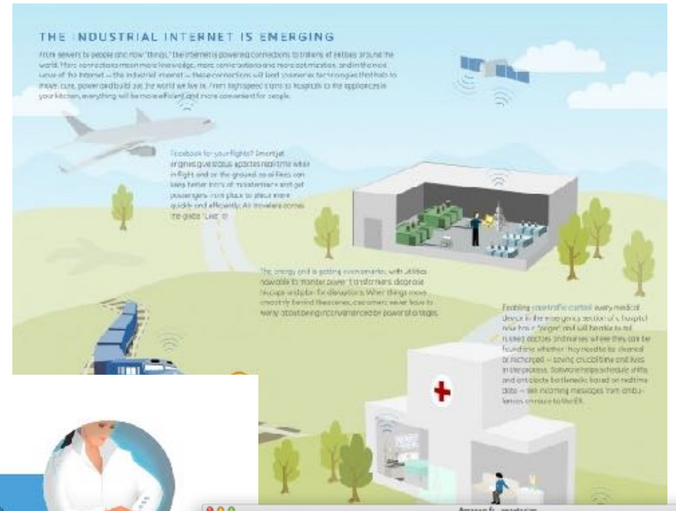


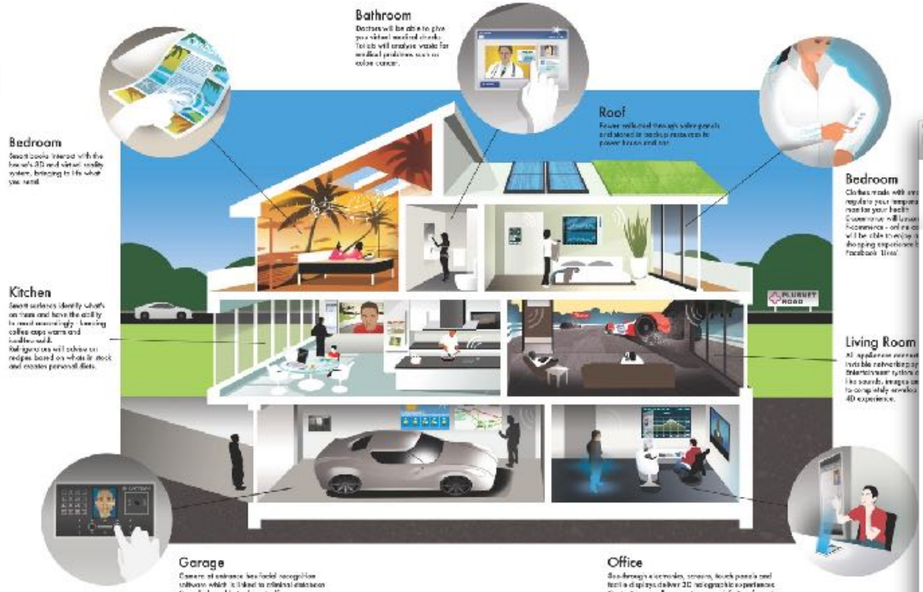


PRISMTECH

Collect | Store | **Analyse** | Share

- Make sense of the data
- Detect short term / long term trends
- …

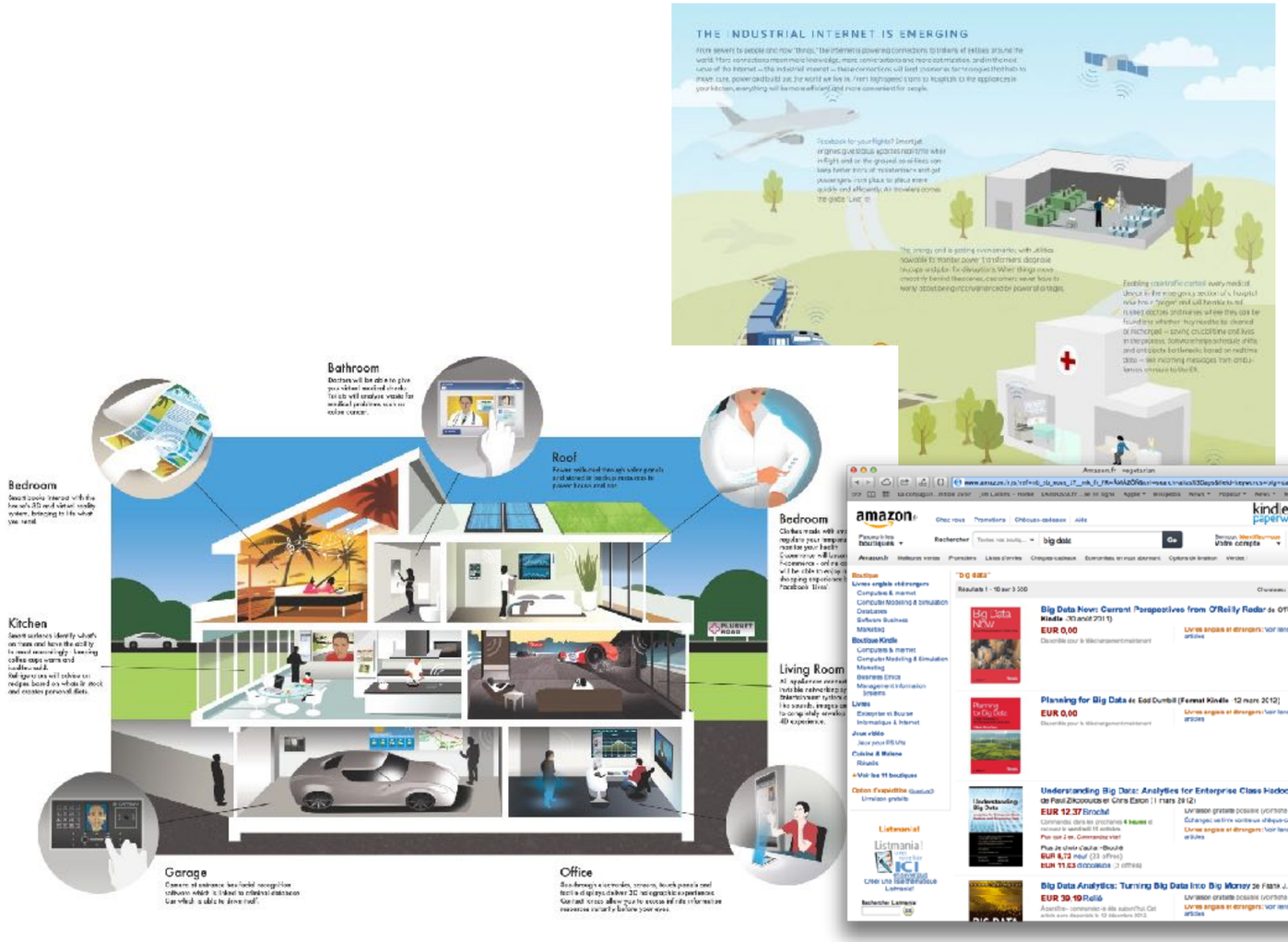




PRISMTECH

Collect | Store | Analyse | Share

- Distribute Analytics -- or any other kind of clues about the data -- to applications that are supposed to act, display, publish, store, etc.



PRISMTECH

Data Sharing in IoT

- Efficient and scalable Data Sharing is a key requirement of practically any IoT system
- The degree of performance required by the data sharing platform varies across Consumer and Industrial Internet on Things applications
- Several standards have been proposed to address this key need of the IoT world

PRISMTECH

Data Communication Standards

Top Contenders

The most discussed and promoted standards for the IoT are currently (in alphabetical order) AMQP, CoAP, DDS and MQTT

Advanced Message Queueing Protocol (AMQP)

- Originally defined by the AMQP consortium as a messaging standard addressing the Financial and Enterprise market
- AMQP is an OASIS standard that defines an efficient, binary, peer-to-peer protocol for for transporting messages between two processes over a network. Above this, the messaging layer defines an abstract message format, with concrete standard encoding
- https://www.oasis-open.org/committees/amqp/

Constrained Application Protocol (CoAP)

- CoAP is an IETF RFC defining a transfer protocol for constrained nodes and constrained (e.g., low-power, lossy) networks, e.g. 8-bit micro-controllers with small amounts of ROM and RAM, connected by Low-Power Wireless Personal Area Networks (6LoWPANs).
- The protocol is designed for machine-to-machine (M2M) applications such as smart energy and building automation.
- CoAP provides a request/response interaction model between application endpoints, supports built-in discovery of services and resources, and includes key concepts of the Web such as URIs and Internet media types.
- CoAP is designed to easily interface with HTTP for integration with the Web while meeting specialised requirements such as multicast support, very low overhead, and simplicity for constrained environments.

PRISMTECH

Data Distribution Service (DDS)

- DDS is the Object Management Group standard for data-centric publish subscribe
- DDS is based on a fully distributed architecture and is equipped with a Dynamic Discovery Service that provide information about “who” and “what” is available
- DDS comes with a rich set of QoS that control data availability, resource usage, e.g. network, memory, etc., and traffic prioritisation
- The DDS standard family has recently been extended with RPC, Security, Web Integration

PRISMTECH

Message Queueing Telemetry Transport (MQTT)

- MQTT was defined originally by IBM in the mid 90's as a lightweight protocol for telemetry
- MQTT supports a basic publish/subscribe abstraction with three different level of QoS
- MQTT has recently gained much attention as a potential candidate for data sharing in the IoT

OASIS
Advancing open standards for the information society

MQTT.ORG

PRISMTECH

Main Characteristics

	Transport	Paradigm	Scope	Discovery	Content Awareness	Data Encoding	Security	Data Prioritisation	Fault Tolerance
AMQP	TCP/IP	Point-to-Point Message Exchange	Device-2-Device Device-2-Cloud Cloud-2-Cloud	No	None	Defined	TLS	None	Impl. Specific
CoAP	UDP/IP	Request/Reply (REST)	Device-2-Device	Yes	None	Defined	DTLS	None	Decentrailsed
DDS	UDP/IP (unicast + mcast) TCP/IP	Publish/ Subscribe Request/Reply	Device-2-Device Device-2-Cloud Cloud-2-Cloud	Yes	Content-Based Routing, Queries	Defined	TLS, DTLS, DDS Security	Tranport Priorities	Decentralised
MQTT	TCP/IP	Publish/ Subscribe	Device-2-Cloud	No	None	Undefined	TLS	None	Broker is the SPoF

Focus…

- DDS and MQTT are the two protocol gaining more traction as candidate for the IoT
- DDS applications are today mostly in IIoT while MQTT applications in the CIoT
- The reminder of this presentation will focus on DDS and MQTT

Protocol Analysis

MQTT v3.1.1

- MQTT is a Client Server publish/subscribe messaging transport protocol
- The protocol was designed for constrained environments and communication in Machine to Machine (M2M) and Internet of Things (IoT) contexts where a small code footprint is required and/or network bandwidth is at a premium
- MQTT transports opaque data and does not define a mechanism to express the payload encoding
- MQTT is a session oriented protocol — uses the transport connection to identify the session — that and runs over TCP/IP and WebSockets

MQTT v3.1.1

- MQTT provides three quality of service for message delivery:
  - **At most once:** messages are delivered according to the best efforts of the operating environment. **Message loss can occur**. This level could be used, for example, with ambient sensor data where it does not matter if an individual reading is lost as the next one will be published soon after.
  - **At least once: m**essages are assured to arrive but **duplicates can occur**.
  - **Exactly once:** messages are **assured to arrive exactly once**. This level could be used, for example, with billing systems where duplicate or lost messages could lead to incorrect charges being applied.

DDS v1.2

- The family of DDS standards define an API for peer-to-peer, data-centric, publish/subscribe and an efficient interoperability wire protocol (DDSI) for disseminating data
- The DDS standard was designed to address the requirement of a wide range of applications ranging from resource constrained embedded systems to large scale Consumer and Industrial Internet of Things (IoT) Systems
- DDS provide a rich set of QoS providing control on:
  - **Data Availability**: reliability and availability (for late joiners) of published data
  - **Resource Usage**: memory and bandwidth utilisation
  - **Timeliness:** data prioritisation and end-to-end traffic differentiation

DDS v1.2

- DDS is data centric, as such:
  - Supports an extensible payload encoding scheme and supports natively multiple encodings, such as CDR (binary and efficient), PCDR (extensible, binary and efficient), JSON and XML
  - Supports content filtering and queries. Where content filters are used to route information while queries are used to efficiently select received data
- DDS can operate on both best effort transports such as UDP/IP as well as reliable transports such as TCP/IP

Reliability in DDS and MQTT

	At Most Once	At Least Once	Exactly Once*	Eventual Exactly Once*
MQTT	QoS = 0	QoS = 1	QoS = 2	N.A.
DDS	**Reliability** = BEST_EFFORT **History** = Any	DDS doen't deliver duplicates	**Reliability** = RELIABLE **History** = KEEP_ALL	**Reliability** = RELIABLE **History** = N

(*) Assumptions:

- The network can temporarily becoming unavailable
- No application crashes

PRISMTECH

MQTT Code Example

```
package org.eclipse.pahodemo;

import org.eclipse.paho.client.mqttv3.MqttClient;
import org.eclipse.paho.client.mqttv3.MqttException;
import org.eclipse.paho.client.mqttv3.MqttMessage;

public class PahoDemo {

  MqttClient client;

  public PahoDemo() {}

  public static void main(String[] args) {
    new PahoDemo().doDemo();
  }

  public void doDemo() {
    try {
      client = new MqttClient("tcp://localhost:1883", "pahomqttpublish1");
      client.connect();
      MqttMessage message = new MqttMessage();
      message.setPayload("A single message".getBytes());
      client.publish("pahodemo/test", message);
      client.disconnect();
    } catch (MqttException e) {
      e.printStackTrace();
    }
  }
}
```


PRISMTECH

```
class Consumer {

  int req_count=0;
  MqttClient client;

 public void start() throws Exception {
    // Create the MqttClient connection to the broker
    client=new MqttClient("tcp://localhost:1883", MqttClient.generateClientId());
    client.connect();
  }

  public void handle(String target, Request baseRequest,
    try {
      // Publish to the broker with a QoS of 0 but retained
      client.publish("countingjetty/handlerequest", Integer.toString(req_count).getBytes(),0,true );
    } catch (MqttException e) {
      e.printStackTrace();
    }
  }

  @Override
  public void doStop() throws Exception {
    super.doStop();
    // Cleanly stop the Mqtt client connection
    client.disconnect();
  }
}
```

What's Simpler?

- Beside simple hello world examples that use Strings, when writing actual data with MQTT, you need to take care of serialisation… And ensure you use the right deserialisation routine on the receiving side…
- DDS, completely hides away this issues to you
- BTW… Not to mention the ClientID management….

MQTT

```
val msg = new MqttMessage()
builder.setSeqn(count)
builder.setPayload(buf)
builder.setTimestamp(System.nanoTime())
val sample = builder.build()
sample.writeTo(os)
val bytes = os.toByteArray
msg.setPayload(bytes)
msg.setQos(qos)
msg.setRetained(false)
broker.publish(pingTopic, msg)
```

DDS

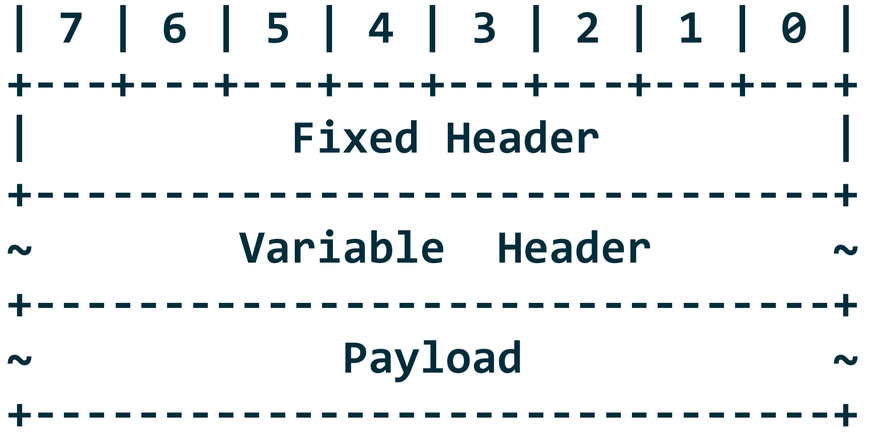
```
val sample =
   new PingData(count, buf,
                System.nanoTime())
dw.write(sample)
```

Message Structure

MQTT Message Structure

MQTT messages are composed by:

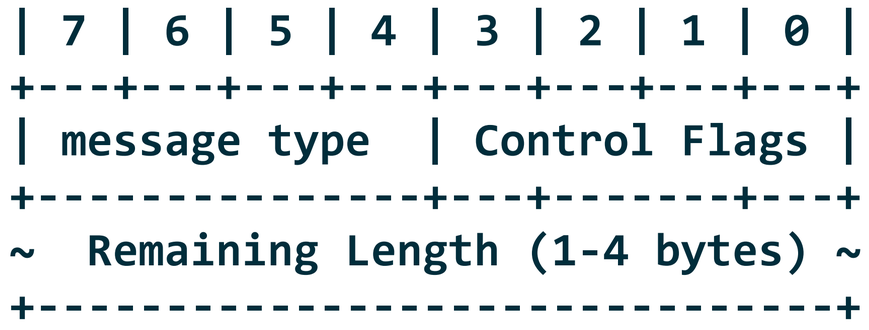
- **Fixed header:** included in every message
- **Variable Header:** Included in some packets. Its content depends on the packet kind
- **Payload:** Included in some packets



PRISMTECH

MQTT Fixed Header

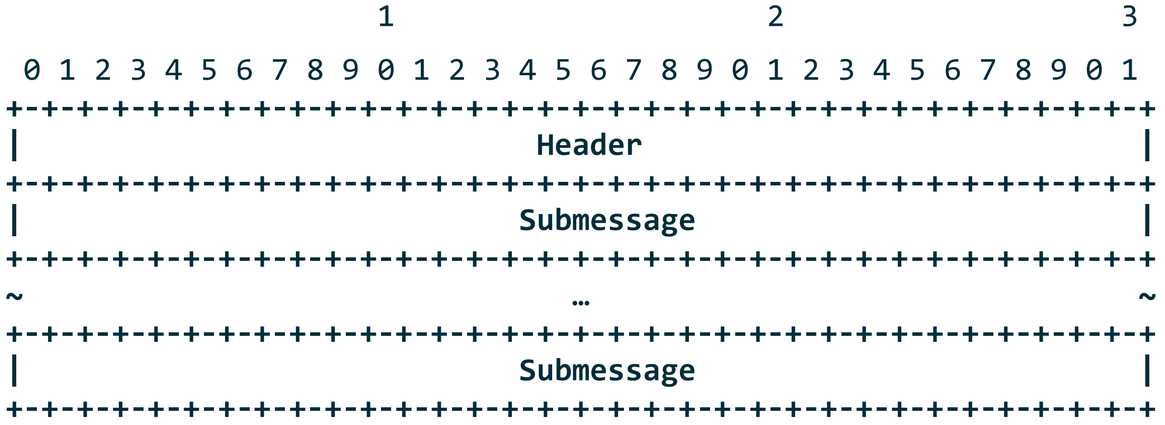
- The MQTT fixed header has a length that can vary from 2 to 5 bytes
- The first two nibbles represents the message type and packet specific control flags
- From the second byte starts the remaining packet length. This uses a “variable-length-encoding” that can take up to 4 bytes



PRISMTECH

DDSI Message Structure

- A DDSI message is composed by a Header and a sequence of Sub-Messages
- The structure of DDSI message make the protocol extensible as new sub message can be added w/o jeopardising backward compatibility
- Each receiver only interprets the sub-messages it understands



PRISMTECH

DDSI Header

The header is composed by

- 4 bytes magic
- Protocol version
- Vendor identifier
- Identifier of the participant that has produced the message (GUID-Prefix)
  - A participant may have multiple communication end-points (e.g Data Reader and Data Writers)
  - Each endpoint is identified by a 4 bytes ID, leading to 16 bytes GUID for identifying any communicating entity in the system

PRISMTECH

Sending Data...

MQTT Publish Message

Fixed Header

- DF: Duplication Flag
- QoS = 0, 1, 2
- R: Retain

Topic Name (Variable Header)

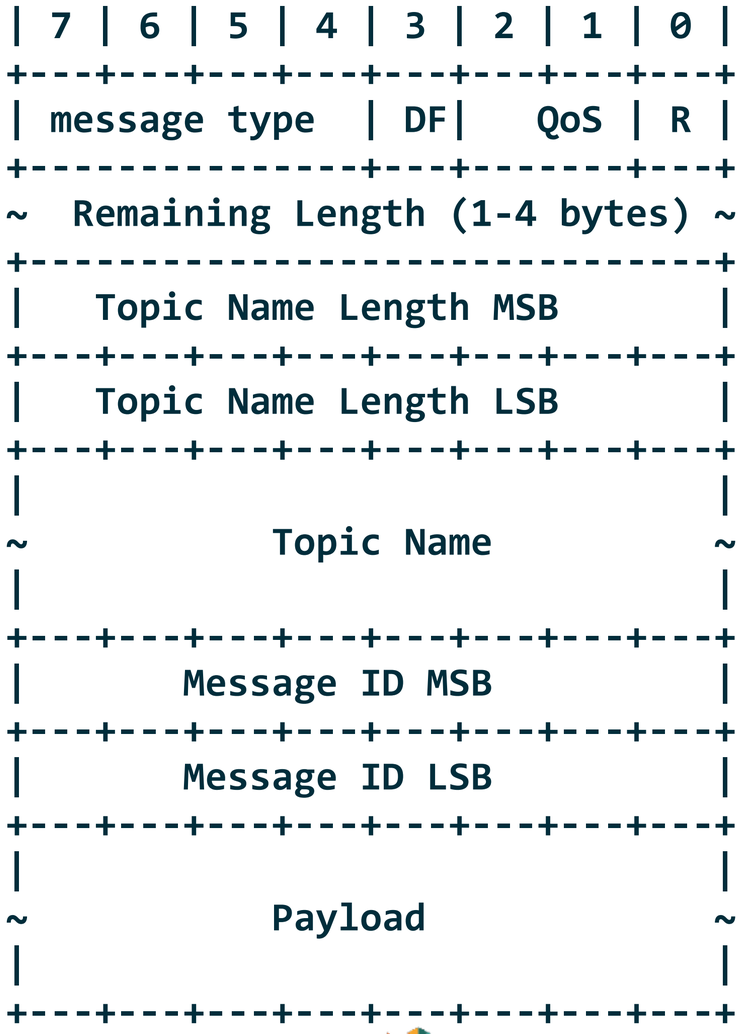
- The length depends on the string chosen by the application
- It is likely, that to avoid collisions, users will give relatively long names such as:
  - `com/acme/myapp/mytopic`

Message ID

- Only present for QoS=1,2 and used to uniquely identify messages

Payload

- Contains the data, but no information on the serialisation format



PRISMTECH

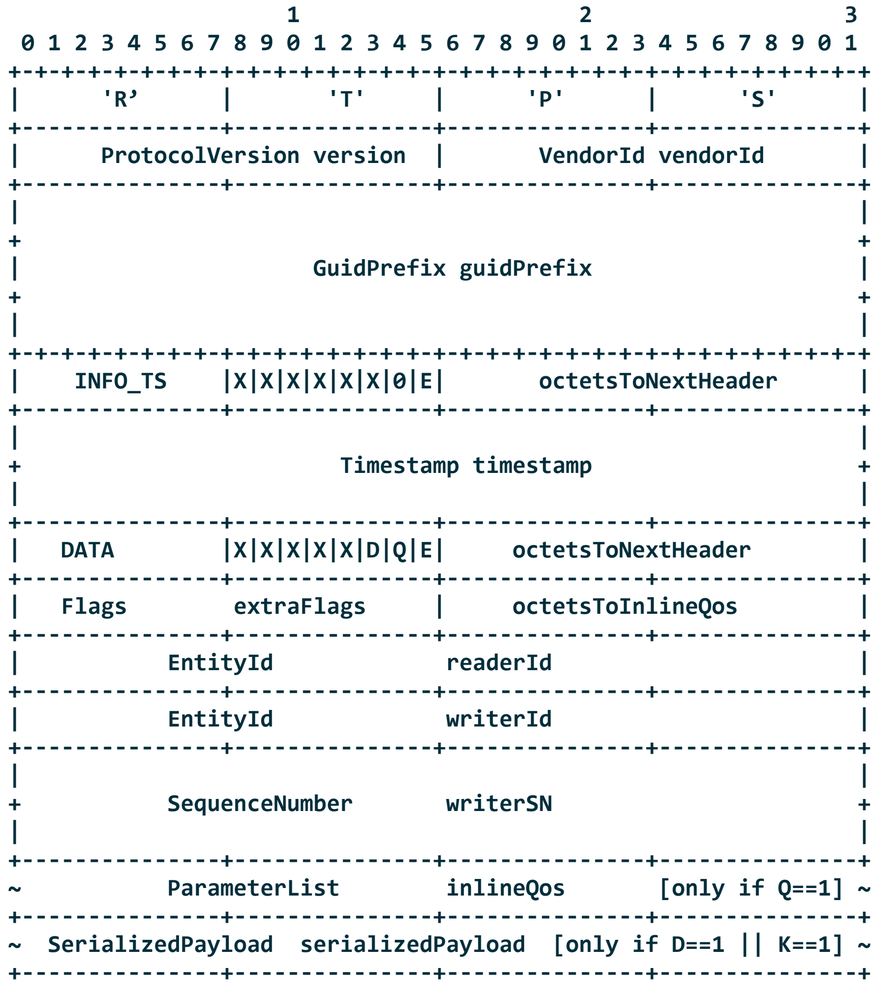
DDS Data Message

INFO_TS Submessage

- This is required when source time-stamp is used to order samples coming from different sources

DATA Submessage

- Provides information on the origin and possibly the destination of the message
- Sequence number
- Potentially some inline QoS and parameters



PRISMTECH

Overhead Analysis

Message Size

Proto	QoS	Data Message Size (bytes)	Transport Overhead (bytes)
MQTT	0	(2 to 5) + 2 + length(TopicName) + length(Payload)	**TCP/IP**(v4): 20-40
MQTT	1,2	(2 to 5) + 2 + length(TopicName) + 2 length(Payload)	**TCP/IP**(v4): 20-40
DDS	DestinationOrder = DestinationTimestamp	44 + length(Payload)	**UDP/IP**(v4): 8 **TCP/IP**(v4): 20-40
DDS	DestinationOrder = SourceTimestamp	56 + length(Payload)	**UDP/IP**(v4): 8 **TCP/IP**(v4): 20-40

MQTT Topic Name

- The MQTT topic name is included in every PUBLISH message, this it is important to understand its structure
- In general Topic name have the format:
  - com/mycompany/app/deviceKind/deviceID
- Notice that the device or appliance ID is useful to include to to be able to subscribe to the flow coming from a specific device, e.g. the refrigerator, as opposed to all instance of a given device
- As such the length of the topic name, in real applications is on the order of tens of bytes

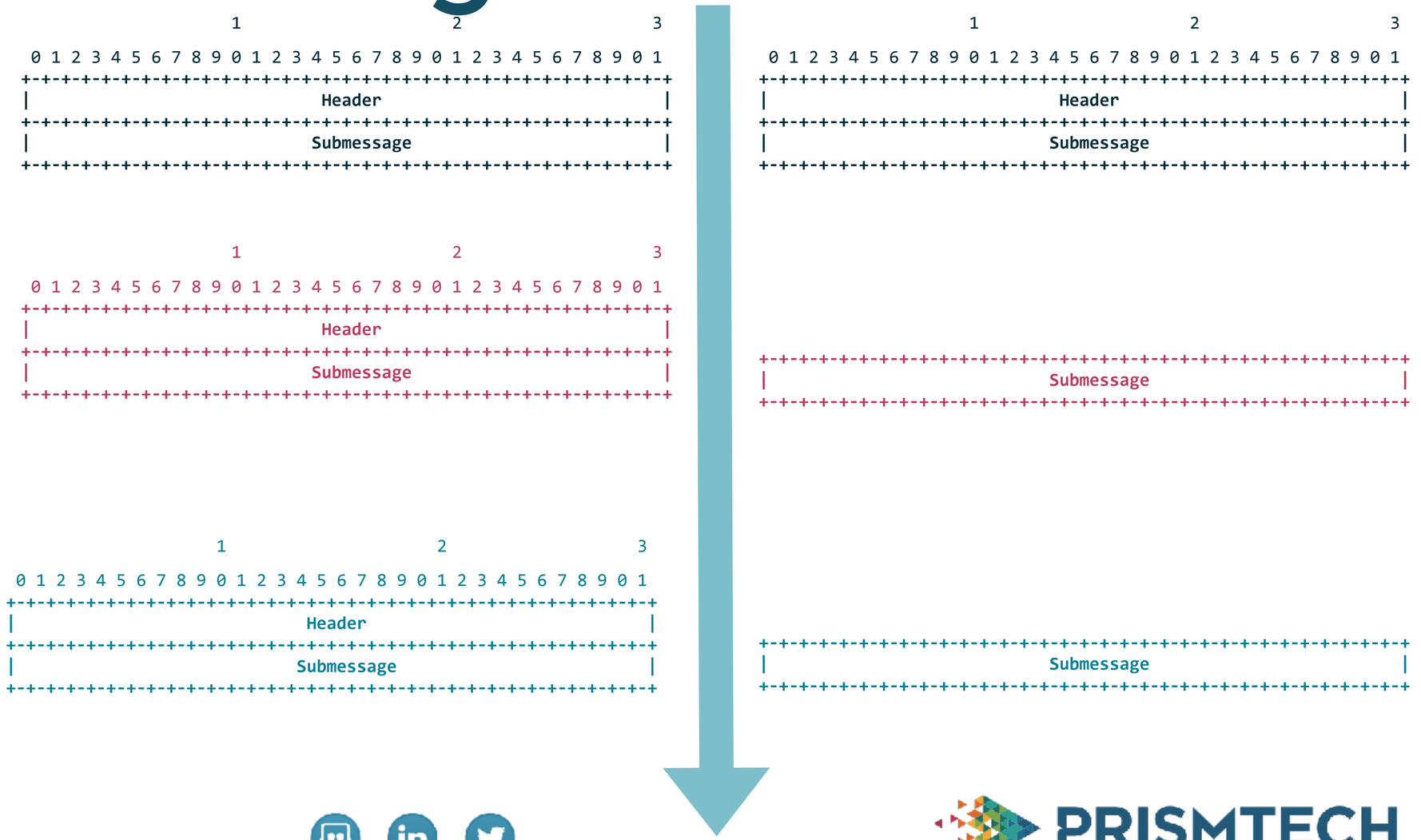
20 Character MQTT Topic Name

Proto	QoS	Data Message Size (bytes)	IP (v4) Headers	Total Overhead (bytes)
MQTT	0	(2 to 5) + 2 + 20 + length(Payload)	**IP**: 20-40 **TCP**: 20-40	**min** = 20 + 20 + 24 = **64** [TCP/IP] **max** = 40 + 40 + 27 = **107** [TCP/IP]
MQTT	1,2	(2 to 5) + 2 + 20 + 2 length(Payload)	**IP**: 20-40 **TCP**: 20-40	**min** = 20 + 20 + 26 = **66**[TCP/IP] **max** = 40 + 40 + 29 = **109** [TCP/IP]
DDS	DestinationOrder = DestinationTimestamp	44 + length(Payload)	**IP**: 20-40 **UDP**: 8 **TCP**: 20-40	**min** = 20 + 8 + 44 = **72** [UDP/IP] **max** = 40 + 8 + 44 = **94** [UDP/IP] **min** = 20 + 20 + 44 = **84** [TCP/IP] **max** = 40 + 40+ 44 = **124** [TCP/IP]
DDS	DestinationOrder = SourceTimestamp	56 + length(Payload)	**IP**: 20-40 **UDP**: 8 **TCP**: 20-40	**min** = 20 + 8 + 56 = **84** [UDP/IP] **max** = 40 + 8 + 56 = **106** [UDP/IP] **min** = 20 + 20 + 56 = **96** [TCP/IP] **max** = 40 + 40+ 56 = **136** [TCP/IP]

PRISMTECH

TCP/IP DDSI Streaming

- When running DDSI over TCP/IP one can transform a stream of DDSI messages into a single streamed message
- This allows to drop the headers…



PRISMTECH

DDSI Streaming Over TCP/IP

Proto	QoS	Data Message Size (bytes)	IP (v4) Headers	Total Overhead (bytes)
MQTT	0	(2 to 5) + 2 + 20 + length(Payload)	**IP**: 20-40 **TCP**: 20-40	**min** = 20 + 20 + 24 = **64** [TCP/IP] **max** = 40 + 40 + 27 = **107** [TCP/IP]
MQTT	1,2	(2 to 5) + 2 + 20 + 2 length(Payload)	**IP**: 20-40 **TCP**: 20-40	**min** = 20 + 20 + 26 = **66**[TCP/IP] **max** = 40 + 40 + 29 = **109** [TCP/IP]
DDS	DestinationOrder = DestinationTimestamp	24 + length(Payload)	**IP**: 20-40 **TCP**: 20-40	**min** = 20 + 20 + 24 = **64** [TCP/IP] **max** = 40 + 40+ 24 = **104** [TCP/IP]
DDS	DestinationOrder = SourceTimestamp	36 + length(Payload)	**IP**: 20-40 **TCP**: 20-40	**min** = 20 + 20 + 36 = **76** [TCP/IP] **max** = 40 + 40+ 36 = **116** [TCP/IP]

PRISMTECH

Summary

- MQTT makes a lot of effort to maintain header very compact, yet the Topic-Name included in the Variable Header of every PUBLISH introduces back a lot of overhead
- In real applications the overhead of DDS when running on UDP/IP is very close to the MQTT overhead
- When DDS runs over TCP/IP in streaming mode has lower overhead than MQTT, e.g. those using real topic names
- Thus in summary, MQTT gives away loads of protocol features not to gain so much in efficiency.