University of Pisa

Computer Science Department

Master degree in ICT Solution Architect

Notes of Mobile and Cyber-Physical Systems

Based on the lectures of prof. S. Chessa, F. Paganelli

A.Y. 2022-2023

Contents

Those notes have been written in Obsidian and converted to LaTeX: may contains typos and other type of errors, both grammarly and conceptually.
The following notes cannot be

Chapter 1

IoT & Smart Environment

There is no universal accepted definition. According to Journal of Ambient Intelligence and Smart Environment: *"smart environments can be defined with a variety of different characteristics based on the applications they serve,* ***their interaction models with humans****, the practical system design aspects, as well as the multi-faceted conceptual and algorithmic considerations that would enable them to operate* ***seamlessly*** *and* ***unobtrusively****"*

They are considered smart for the final user because can **recognize context, situations and activities**, **able to figure out use need at right time** and **provide services**. They also have in common several characteristic like **persasive**, **unobtrusive cyber-physical devices**.

For Iot or **Internet of Things** we mean physical objects embedded with electronics, software and sensor, with **network connectivity**. Generally, they are objects with **sensors** and **actuators**: they respectively represent the two flow of data in-and-out of the IoT system because **sensors** can receive input data and elaborate them by resulting in a performed physical action on the physical environment by **actuators**.

IoT Devices Each IoT devices is composed by 4 main components:

- **Sensors/actuators**
- **Microcontroller**
- **Wireless interface**
- **Software for business logic**

We can identify in IoT a *layered architecture* pictured in 1.1 And, by istantiating this model we can obtain the following scenario:

Sensors at **perception layer** can be of different type based on the nature of activity that need to be carried out:

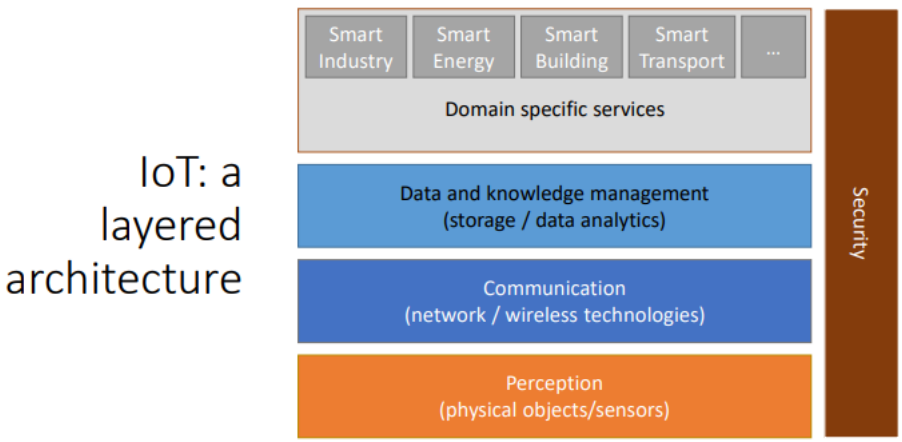

Figure 1.1: undefined undefined

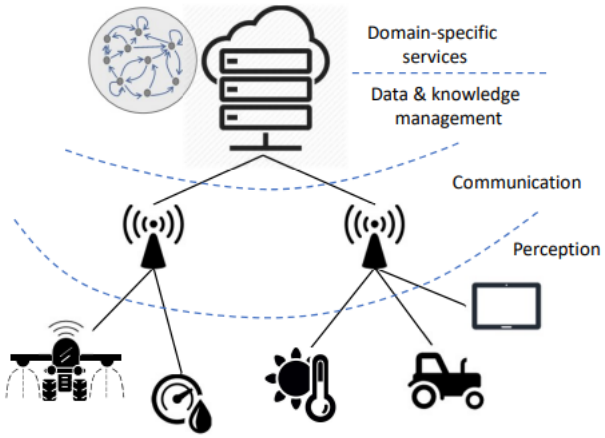

Figure 1.2: undefined undefined

- Specific: they're a fully-functional component and not need to be istantiate along other components to accomplish the desiderated task
- Aspecific: they're part of a chain of interaction with other component and this interaction (*e.g. Camera that effect facial recognition task*) allows to perform the intended task

Platforms for IoT Sensors and actuators are the edge of the cloud: behind internet, data is stored, processed and presented in the cloud. So **IoT Platforms** provide software layer between IoT devices and applications. Their functionalities may be distributed between devices themselv, gateways and server in the cloud or at the edge. Those platform not allow only data collection but can perform several complex functionalities with a various range of activities (*as mentioned, facial recognition can be a service offered/performed via AWS SageMaker connected to AWS IoT*). Those platforms allows a wide range of functionalities:

- **Identification**: provides a unique way to identify things in the platform. There are several established standard, based on context:
  - *IP Address*: for Internet
  - *MSISDN – Mobile Station International Subscriber Directory Number*: for telephony identification
  - *URI - Universal Resource Identifier*: on resources exposed to web
  - *UUID - Universally Unique Identifier*: in computer system and bluetooth-based communication system
- **Discovery**:
  - A mean to find devices, resource and/or services within an IoT deployment inside the same network of devices. This allows also to obtain properties, features and services and locate via identification.
- **Device Management**: as *initial setting* allows to pair, secure setting and key distribution, manage configuration and binding of devices with services, calibrate the sensors, localize devices etc. Managing involves also **Automatic Software Updates** (*both for software and firmware*), **device real-time monitoring** (*battery level, internal tempeture, energy consumption*), **diagnostic/default detection**, **remote control of devices** (*by activate pheriperals and remote rebooting, without human interaction*) and **system logging and audit management**. Clearly there are different standard based on a given context. The most popular are:
  - **OMA DM (Open Mobile Alliance Device Management)**: mainly for management activity of mobile terminals
  - **OMA LWM2M (Open Mobile Alliance Lightweight Machine2Machine**: used for management of IoT devices

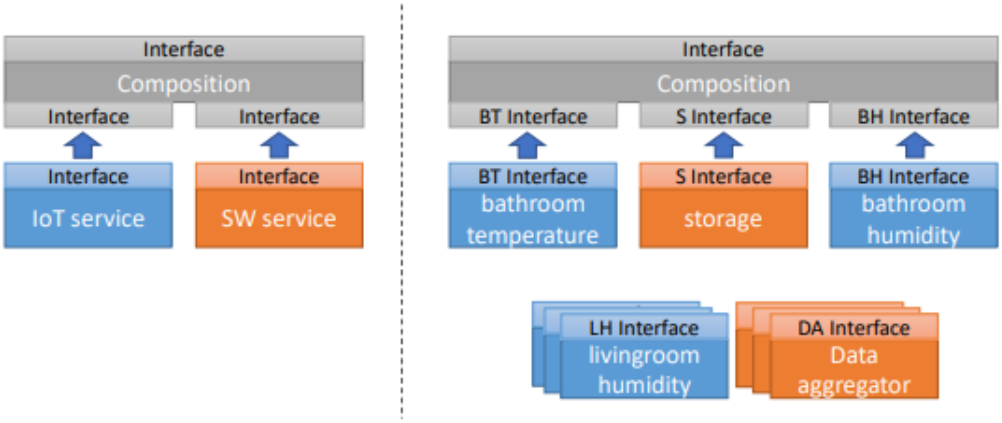

Figure 1.3: undefined undefined

  - **BBF (Broadband Forum) TR-069**: management for end-users appliances (*e.g. DSL terminals, etc.**)

- **Abstraction/Virtualization**: an IoT device is seen as a service so the physical device is associated with its *digital twin*. This provide a way to represent IoT devices and their context, enabling a broad of function on the virtual-device like reasoning and AI-processing over data.
- **Service Composition**: build a composite service by integrating services of different IoT devices and SW components as sketched in 1.3

Based on the previous layered architecture model, we add the details just mentioned and provide a general picture on how those are distributed among the layers, obtaining the model pictured in 1.4

NoSQL databases

No-SQL databases are used in big data and real-time application: they're suitable to scale on horizontal cluster of machines.

As main example, we use **MongoDB**: what in SQL were *Records* in MongoDB are **documents**. Documents have a *JSON*-like data syntax: in the following snippet, they're all *fieldUnknown Node :: textDirective* entry.

```
{
name: "sue",
```

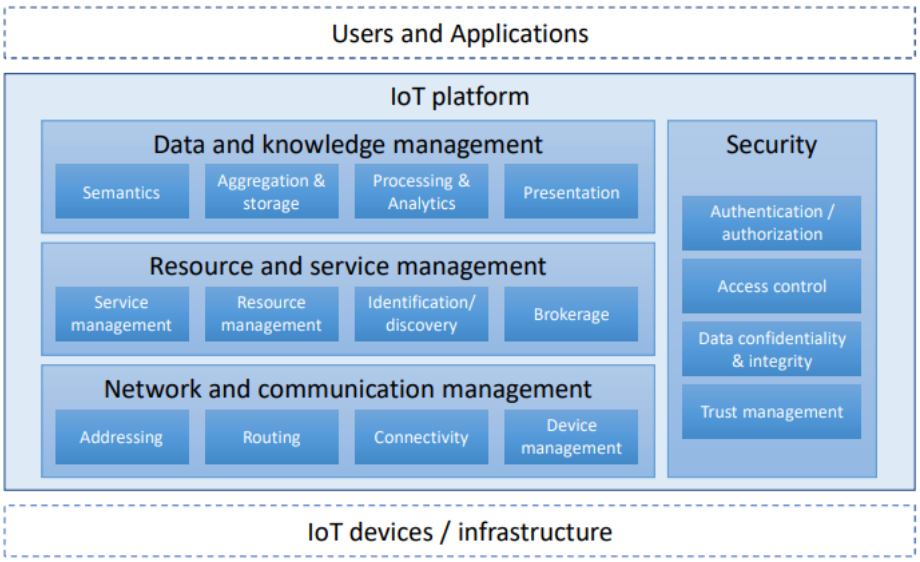

Figure 1.4: undefined undefined

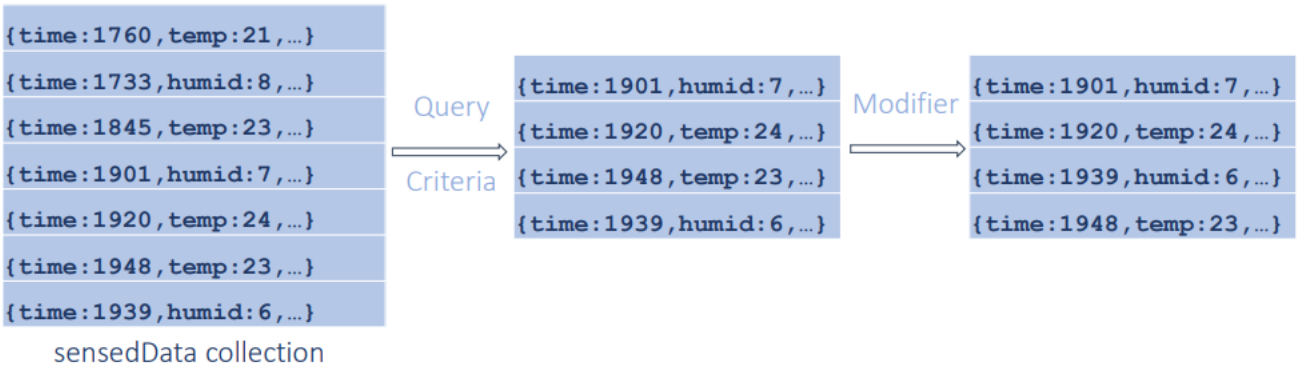

Figure 1.5: undefined undefined

```
age: 26,
status: "X";
group: ["news", "sport"]
}
```

A set of *name/value* pairs are separated by commas: a `value` can also be an array. So MongoDB **documents** correspond to **native data types** in many programming languages: can embed other documents and arrays to reduce the need for expensive joins.

What were **tables** in SQL, in *MongoDB* are **collections**: documents can be grouped in collection but, differently from SQL, different documents in the same collection can have different structure.

Queries Usually a **query** targets a specific **collection**: it specify criteria and conditions that identify a set of documents in the collection. As SQL, a query can also include a *projection* that specifies the fields to return or include modifier of the output (*like sorting of result, manipulating data format, etc*). Here an example:

```
db.sensedData.find({time:{$gt: 1900}}).sort({time:1})
```

which is represented by the image pictured in 1.5.

We can **modify the data** by update/create/delete: the `update` and `delete` operations can specify the criteria to select the documents to update or remove. Here an example of `insert` query (*referring the pictured in 1.6*):

```
db.sensedData.insert(
{
time:2011,
humid: 5,
... }

)
```

Relevant issues in IoT

There are many issues in IoT and they're not easy to address:

- **Performance**: because microcontroller have a very low computation capacity there are physical and computational constraints to evaluate based on the type of activity to perform. This induces to optimize various parameters and metrics.
- **Energy efficiency**
- **Security**
- **Data analysis/processing**:
- **Communication/brokerage/binding**: How to bring together data producers (*sensors*) with consumers (*users/actuators/applications*).

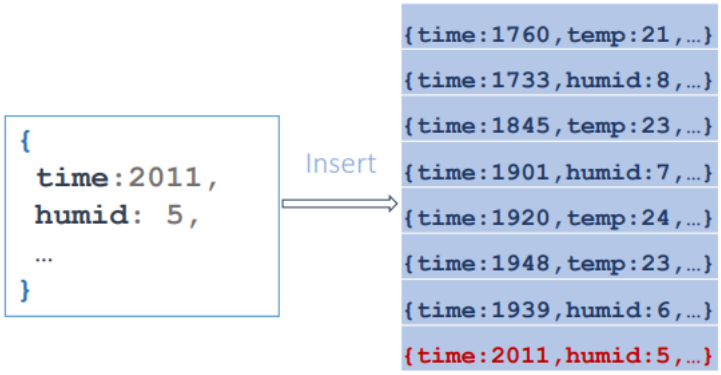

Figure 1.6: undefined undefined

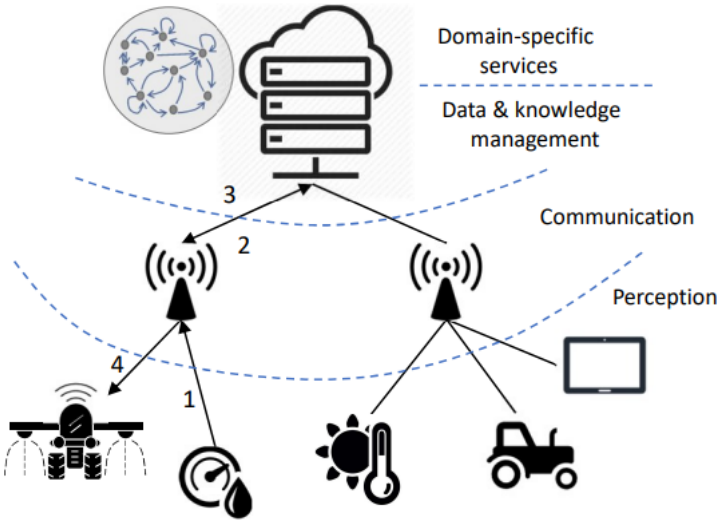

Figure 1.7: undefined undefined

- **Data representation**: data formats and standardization
- **Interoperability**: many standard already exists at different level, like:
  - **MAC Level**: *Bluetooth, IEEE 802.15.4*
  - **Network level**: *ZigBee, Bluetooth, 6LowPan,*
  - **Application level**: : *MQTT, CoAP, oneM2M*

1.0.1 IoT issues

In IoT we can have **latency** and **reliability** issues: this happens because the physical devices can be (and they are) far from each other, this without any doubt adds latency to the iot network. With reliability we mean the process of the network to lose nodes or packets due to some type of **failure/incident**. We saw 2 approaches:

- **Everything is on the Cloud:** this means that all the processing is done in the cloud, the devices at the perception level are meant only to acquire data and send it to the cloud servers, where all the processing is done. Here latency impacts a lot, since the amount of data sent to the cloud can be huge.
- **Some Processing on the Edge**: We can adopt some perception devices (still sensors) with a little bit of computational power (small cpu) to process some of the input data, in this way we can reduce the amount of packets sent to the Cloud servers. It's not mandatory to use the cloud, if your devices can make all the processing alone, then you don't need any cloud.

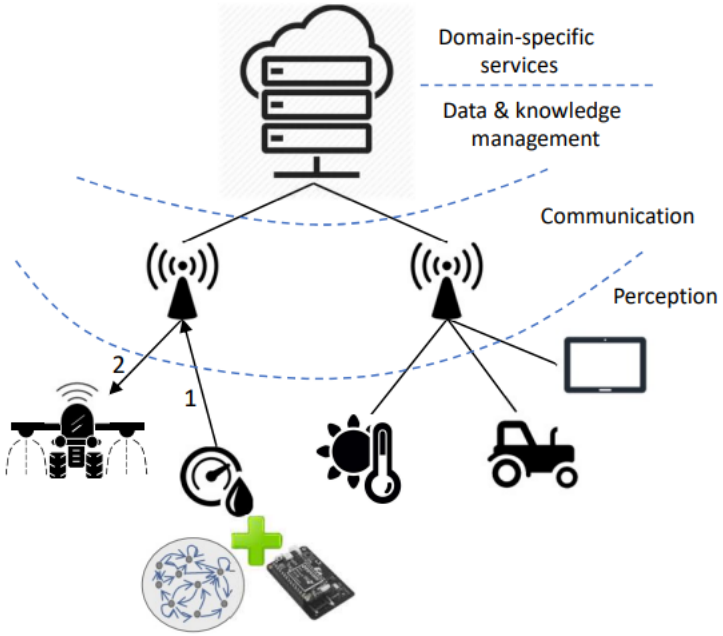

Figure 1.8: undefined undefined

Let's see what some terms means by referring the scenario pictured in 1.9:

1. **Edge**: the edge of a typical enterprise network is a network of IoT-enabled devices consisting of sensors and perhaps actuators, these devices may comunicate with one another. A cluseter of sensors can comunicate with a central device that aggregate the data to be collected by a higher level entity. A **gateway** interconnects the Iot devices with the higher level comunication networks, it performs the traslation between the protocols used in the comunication networks (on top of **perception** layer) and those used by devices. Perception layer uses some protocol that needs to be translated to enter the network. In **Poche Parole** in the edge we find the **physical** devices of the perception layer.

2. **Fog**: It's an infrastructure of local servers and access points, connected together to provide fast response time to the **edge devices** and to reduce the amount of data that goes to the cloud, typically the fog devices are deployed **physically near the edge**. So rather than storing data **permanently** or for **long period** in a central storage we do as much processing as possibile on the **Fog** level. Processing elements at these levels may deal with high volumes of data and perform data transformation operations, resulting in the storage of much lower volumes of data. Examples: **Evaluation**, **Formatting**, **Expanding / decoding**, **Distillation / reduction**, **Assessment**. You can think of Fog as the opposite of Cloud: **Cloud: centralized storage and processing for a relatively small number of users Fog: distributed processing and storage resources close to the massive number of IoT devices**

3. **Core**: the core **network connects far fog networks and provides access to other networks.** The core network uses high-performance routers, high-capacity transmission lines and multiple interconnected routers for redundancy and capacity. It may connect to other type of devices, such as high performance servers and databases or private cloud capabilities. **This means that you can connect fog networks to the cloud.**

1.0.2 The AI approach to IOT

AI aims at getting computers to behave in a smarter manner, **either through curated knowledge or through machine learning**. **Curated Knowledge:**

Machine learning: Subfield of AI that deals with automatic systems that can learn from data. The system is fed by examples to learn to associate input with output, then when an input (**never seen**) is given, the model/system produces anyway an output. If **well trained** the output will mostly be correct, due to generalization capability of ML. **3 ML PARADIGMS**: **Unsupervised learning**: In

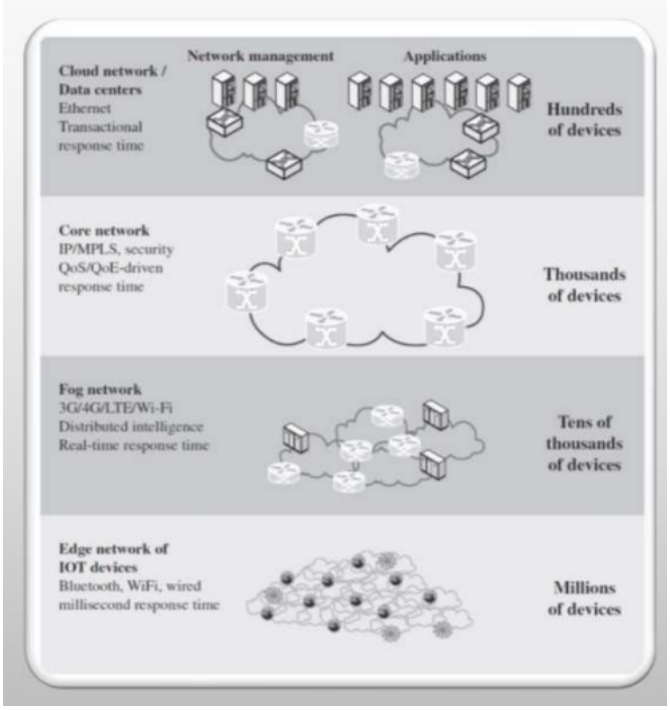

Figure 1.9: undefined undefined

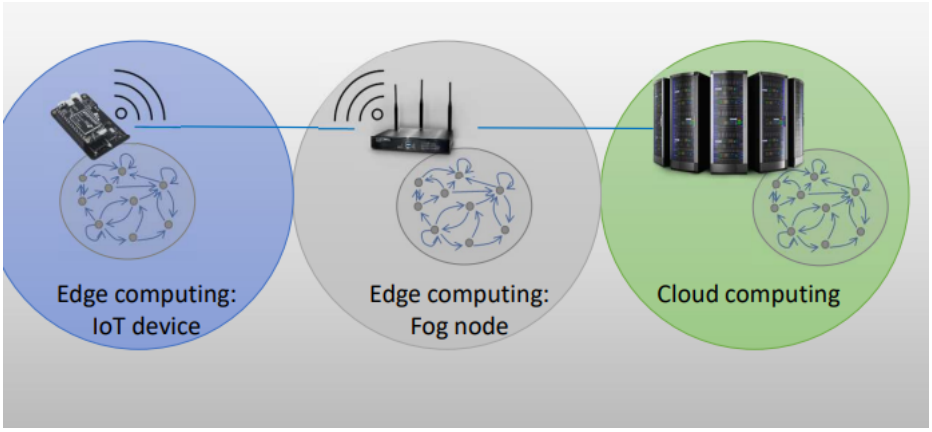

Figure 1.10: undefined undefined

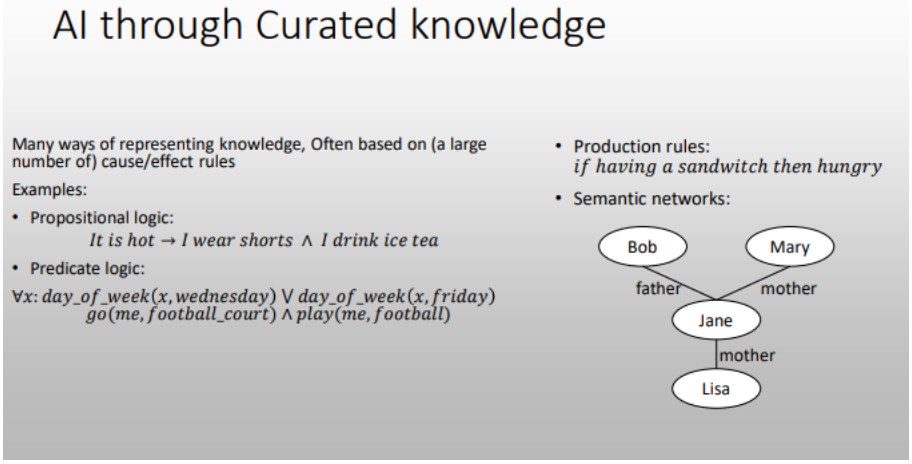

Figure 1.11: undefined undefined

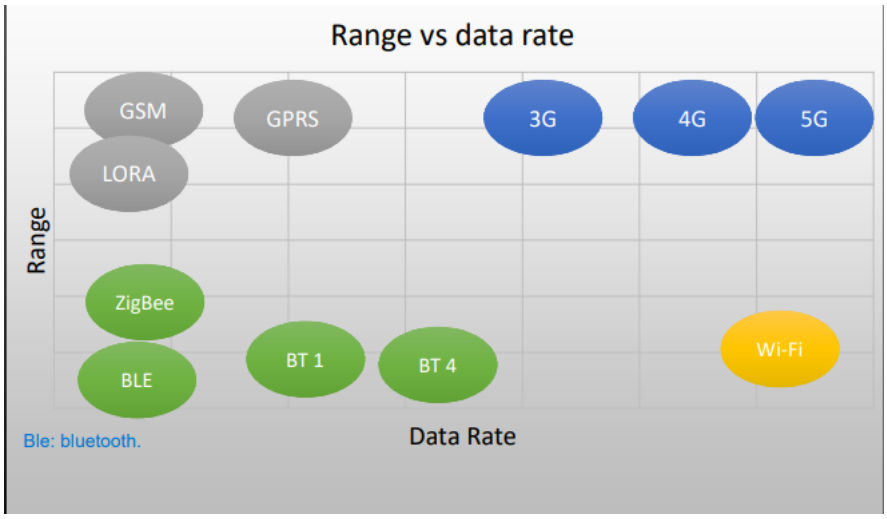

Figure 1.12: undefined undefined

unsupervised learning, the algorithm is presented with a dataset and must find structure or relationships within that dataset on its own. The goal is to identify underlying patterns or groupings within the data. **Without any type of labeled data**.

The following 2 are used in IoT:

- **Supervised learning**: learn from past examples, **every example is a pair input + desired output**, aims at predicting the future or interpret the present.
- **Reinforcement learning**: learn from examples, **but the main difference is that the example is a pair input + reward**, in this way the system knows that the answer is correct if there's a reward. (*E.g. to learn a game the reward can be +1 for winning, -1 for losing, 0 otherwise*).

SLIDE 60 non so proprio che scrivere

1.0.3 IoT and emerging Paradigms

Blockchain is a **shared and trusted public ledger** for making transactions, everyone can inspect and nobody can alter it. the blockchain thus provides a single point of truth: it is shared and **tamper-evident**, in case of tampering it's quickly noticed. Blockchains shifts the IoT paradigm from **centralized storage** to a **decentralized one** in a distribuited ledger. Supports the expanding of the Iot Ecosystem. Since the **ledger is public** this implies a **reduce in maintenance costs** and provides trust in data produced. An implementation of this approach can be seen on having different companies that work in a supply chain, everyone of them needs to **check the quality of the product** along the chain. Each company in supply chain can **query the ledger** and check the latest transaction, thanks to **smart contracts** that are used to certify each intermediate transaction.

1.0.4 Interoperability & Reference Standards

A straight implementation of an IoT solution is not a problem by itself, you can design the solution from the bottom (Physical layer) up to the application layer. This approach is called **Vertical Silos** (No external comunication, tutto quello che fa parte dell'infrastruttura è interno al silos) since this system only has your devices and every change/update requires **your intervention**. This creates a **Vendor Lock-in** for your clients because the silos prevents them to use devices from other vendors (Wii con crossfit, altri sono riusciti a crearlo e venderlo a prezzo più basso). Vendor lock-in forces high costs to **migrate** to another vendor, customer need to fully redesign and deploy a new solution. **Standards** are useful to fix this problem. The first standards we saw are for **Wireless technologies**:

1. v**IEE 802.11** aka **WIFI**: is a **family of standards** useful for networks where devices are close to each other (Range = 100 meters) but need high data rate. There's a new standard IEEE 802.11A, B, AC. They added new features such as additional frequency bands (5GHz). Since Wi-Fi networks operate on specific frequency bands, with this band the network is faster and less congested. **Increasead trasmission range and bit rate.** Another useful feature is Roaming between access points, this means that a device changes to the nearest access point in the network.

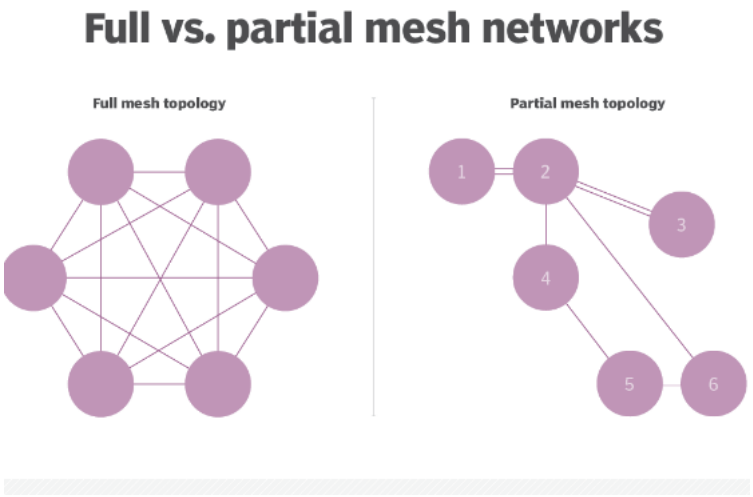

Figure 1.13: undefined undefined

2. **IEEE 802.15.4 and Zigbee**: is a wireless communication standard for low-rate wireless networks, where devices are very close to each other (low range). It defines **both physical and MAC layers**. Usually used when developing low-power sensor networks. Low power = Low throughput (115 Kbps, molto poco) and low duty cycle (percentage of time sensor is sensig). Thanks to the fact that Zigbee network can create a **mesh topology**, we can create larger range networks. In partial mesh topology, nodes basically forward the data to the desired node, in this way the network can grow. The general idea is pictured in 1.13.

3. **Bluetooth**: Higher data rate than ZigBee, but the range is basically the same. Used to make multimedia comunication. Small networks with **master-slave comiunication**, master sends the data and the slave acquire it.

Why standards? Usually motivated by a reduction of the costs for development of a technology, the price for a new technology is high but as soon as other companies try to create the same one this will **instantly drop the price**. For instance think about a technology that has been developed on the lower layers, in this case comunication layer (Wifi) the companies can move to higher layers in order to create their own technology and sell it at their price.(a quel livello non si guadagna più). **Coopetition** can be a good approach for companies, they work with eachother to develop a standard but still deploy their own technologies, built on top of those standards. **Usually happens when technology becomes mature,the big revenues are somewhere else.** As said before, this happened for wireless comunications, we saw lot of different standards. **Interoperability problems moved to the higher layers**.

The problem of **interoperability** arises between consortium of standards. Nowadays, the problem of interoperability and thus of standarization is moving up at middleware/application layer. For ZigBee, it covers many layers: it defines a network, transport and application layer specific for ZigBee and incompatible with others.

When there are too may standard avaialble and they are not compatible, the solution is to use the concepts of *endpoints* or **integration gateways**: allows to translate from a given standard to another. The gateway is limited to a single layer but they're able to translate a layer protocol to another one.

We have different scenarios/configurations as sketched in 1.14

In the left case, even if the same vendor with same protocol, the nodes need a communication to the service gateway that provides access to the internet. In the right case, we have different vendors that commonly use the same protocol, respecting a common standard: even in this case we need the sservice gateway to access the rest of the world. The service gateway allows to map *the rest of the world* to interact with it.

For devices that are not using internet, there is no need to provide external connectivty: differently for ZigBee or MQTT in which the service gateway provide the functionality of transmitting external data. Different scenario is pictued in 1.15

In the left scenario, we have 3 networks which talks using a common standard provided by the **integration gateway**: in each network, the node talk each other using specific protocols, different from network to network. A **integration gateway** is able to speak different languages/protocols and able to

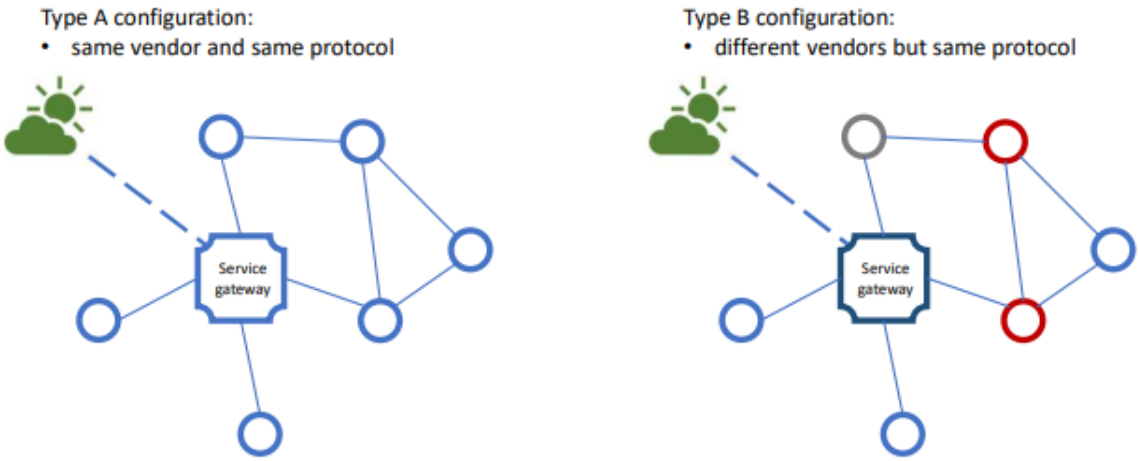

Figure 1.14: undefined undefined

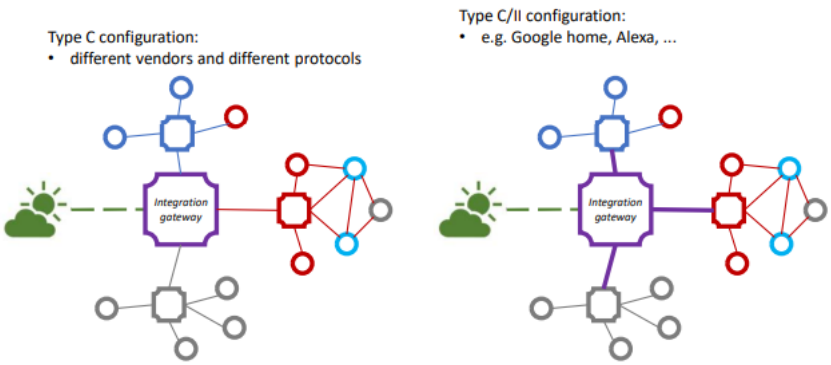

Figure 1.15: undefined undefined

map one protocol in another. The mapping can be complicated becase can define different behaviours of the devices: in case of a protocol that *push data* and other that differently *pop data* are not translable in one another. So the integration gateway need to translate the *behaviour*: it also need to complete the packet structure, frame tramission, etc.

In type C/II scenario, the key idea is that the protocol standard or under-the-hood technology is driven by the players (*Google, Amazon*) that in practice determine the specific technology of *integration agteway*, pushing other players to develop in accordance to their technology to guarantee interoperability.

Provide a general purpose integration service is complex: a better solution is forming a *network* of integration gateway that allows to map one protocol to another one or subset of specific-translated protocol. So we obtain a network of **distributed integration gateways** as sketched in 1.16

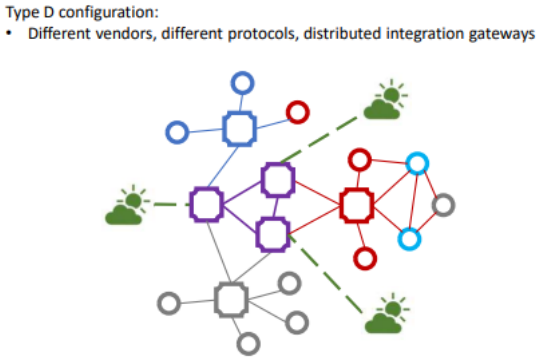

Figure 1.16: undefined undefined

Brief Security in IoT

The general context which we refer is pictured in 1.17

Some devices are constrained, unconstrained or provide security features: some of them, despite not having security features, they're connected directly with internet. A paper of 2014 poses to problem of security in IoT: the pressure of time-to-market and fasten development phases set the problem of security on devices that are constrained in terms of capabilities or powers so security sometimes became optional because it's not fittble in that types of devices.

Usually those devices are not capable to run a full-fledge OS but only a simple OS without security features that are necessary to implement policies, even during updating the software-specific features.

The problem arises in the topics of *confidentiality* and *authentication* (*e.g. wearable, physiological sensors*).

Security Standards The IUT-T standard recommendation includes a list of security requirements for the IoT: it describes functional requirements during capturing, storing and trasferring7processing data of things as well as provision services. The requirements concern:

- **Communication security** (*secure, trusted, and privacy protected communication capabilities*): enforces confidentiality and integrity of data during data transmission or transfer
- **Mutual authentication and authorization**: mutual authentication and authorization between devices (or device/user) according to predefined security policies. Before a device (or an IoT user) can access the IoT. The authentication pattern can follow 2 main methods:
  - Remote: directly to the cloud but poses complexity due to device's constraints
  - Local: between devices or authentication service at most one-hop far way or with low overhead for devices The mutual authentication is meant as *in both direction between devices and gateway.*

The **gateway** is the main building block of the security mechanism and the devices implement only a subset of logic in accordance to the security operations carried out by the gateway.

Authentication can happen at different levels: authenticate devices only at physical layer is important but itself it's not a guarantee that the behaviour at the application level is correct (*e.g. tampered device at physical layer*). There are also other features like protect privacy for devices and gateway, self-diagnosis and self-repair. A critical phase is the deployment of the devices because initially they're not fully configured and may have not been still identified the correct gateway to connect.

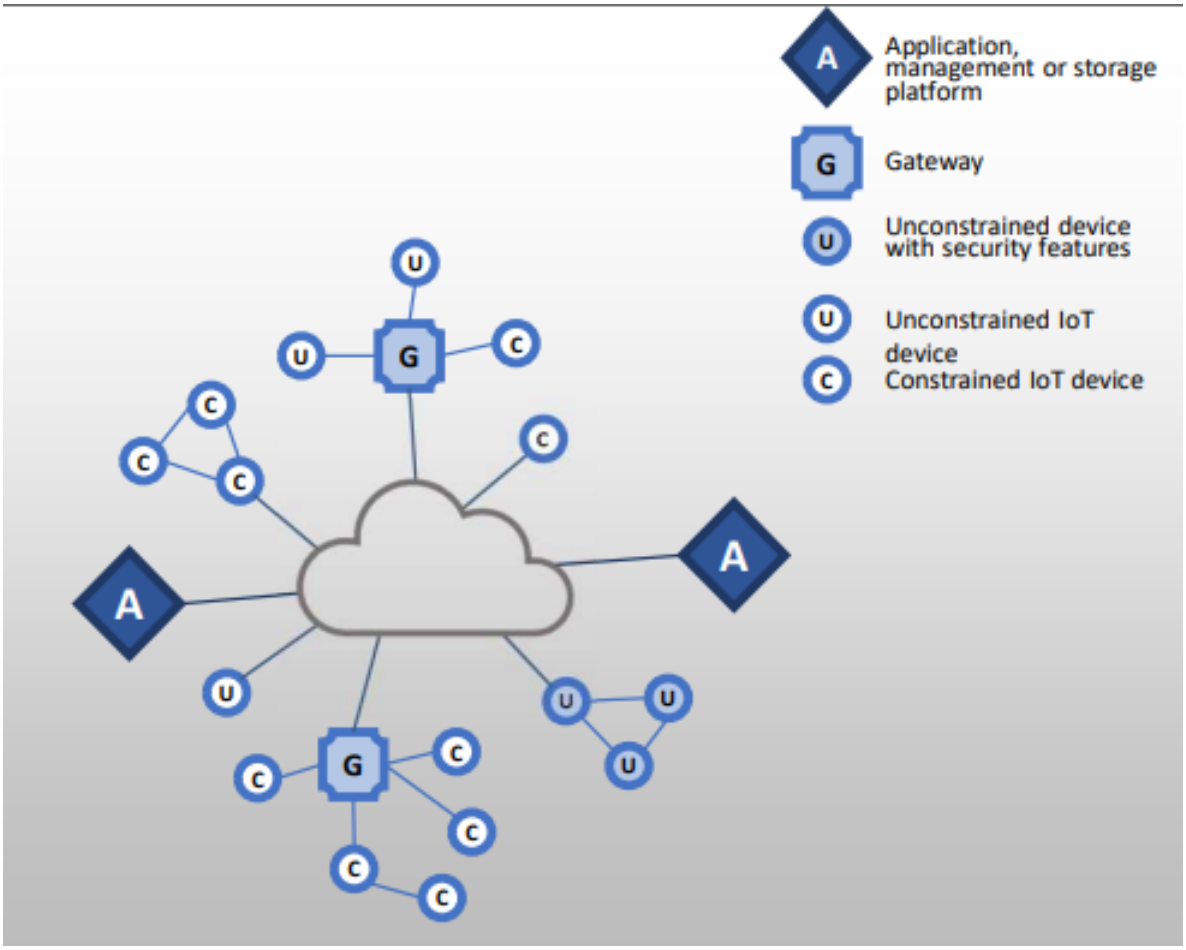

Figure 1.17: undefined undefined

Chapter 2

MQTT - Message Queueing Telemetry Transport

MQTT covers application, session and presentation layer (*both directly or undirectly by using middleware*). MQTT uses an instance of *publish-subscribe* model between users and consumer: internally, as we will see, is implemented as a client-server architecture. To connect to internet, the IoT devices must adopt internet protocol suite. However, internet stack is too resource-compsuntive respect to constrained IoT devices that usually are *lossy, low power, etc.*

IoT Requirements The IoT devices have different requirement based on given context and offered/requested features.

- *Multi-hop/Mesh networking* allows to provide a connectivity and communication among shared group of devices.
- Addressing:

MQTT it's a *lightweight publish/subscribe reliable messaging transport protocol.* Lightweigth thanks to small code footprint and low network bandwidth, with minimal packet overhead (slightly better than TCP). It builds upon *TCP/IP* with port `1883` or `8883` for using MQTT with SSL (*with overhead*).

Internally MQTT uses a **client/server** architecture: this bought the major complexities on the server side. Provide QoS and it's *data agnostic* so it's suitable both to M2M and IoT.

Publish/subscribe model The main components of the model are **publishers, subscribers** and **broker or event service**. The fist two are both seen as clients and do not know each other, the latter it's the serer known both by publishers and consumer. The **publisher** produce event or any data: interact only with the **broker** and does not know if the data is consumed/read by other entities. The subscribes express interest for an event (*or a pattern of events*): receive the notifications from the broker whenevr the event is generated. Publishers and subscribers never interact each other: they interact indirectly only by the *broker(s)*. Those pattern allows to be publisher and subscriber to be decoupled in **time, space and syncrhonization**.

The broker know publisher/subscribers and receive all incoming messages from the first, filter them all and distributed them to the interested subscriber. It also need to manage request of subscription/unsubscription.

A **publish/subscribe** interaction can be implented in different ways: the boker is usually an *independent* agent and manage, coordinating with publishers/subscribers, the operations of `publish, subscribe, notify, unsubscribe` as briefly introduced in 2.1

The **broker** is delegated to storage and management of subscriptions.
In figure 2.2 example in which different subscribers register themself to one or more *topics* (*smartphone wubscribe to T&H or temperature and humidity* while *computer subscribe only to temperature topic*).

This model allow to **decouple the space** because pub/sub do not need to known each other and do not share anything (*they don't known the IP/port of each other and how many peers are subscribed*). It allows also the **time decoupling** because they don't need to run at the same time, guaranteeing the **asyncrhonicity** of the model. The asyncrhonous model is pictured in 2.3.

The decoupling increments the **scalability** of the actors involved: the operations on the broker can be easily parallelized and are event-driven, allowing scalability to a very large number of devices by parallelizing the broker.

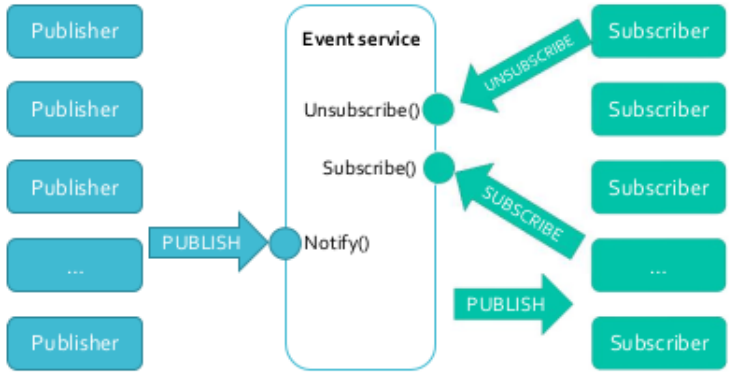

Figure 2.1: undefined undefined

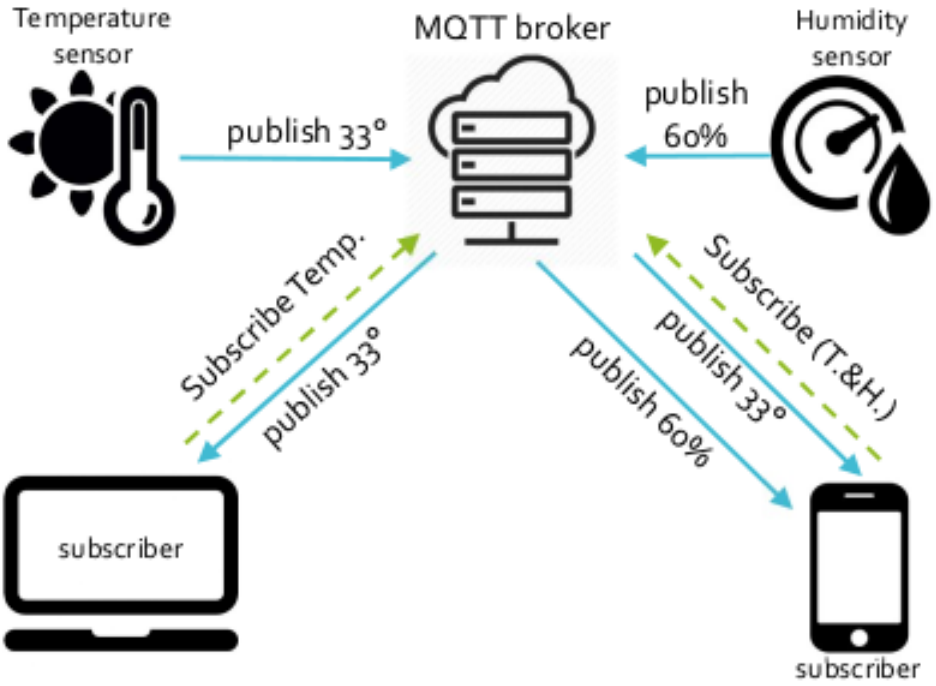

Figure 2.2: undefined undefined

Filtering The filtering features managed by the broker can be based on three different criteria:

- *Based on* ***subject topic***: the subject (*or topic*) is a part of the message and clients subscribe only to a specific topic. They are usually represented by a string, possible organized in a hierarchical taxonomy.
- *Based on* ***content***: the client subscribe for a specific query (*like* `temperature > 30°`) and that query is used by the broker to filter the messages. This methods involves to understand the semantics of the data transmitted or at least be able to *read* the data so add complexity con context in which security mechanism (*like encryption*) are used to transmit/store data.
- *Based on* ***type***: filtering of event is based on both *content* and *structure* so the type refers to the type/class of data that can be customized by the subscriber by defining in a strongly typed language the desired object structure. This methods introduces complexities in case publishers and subscribers are written in different language and there is no a unique *translation of type* because their type system is not uniquely determinate. In those scenarios tight integration of the middlewaew and language paradigms (*like Object Programming*) can ease the presented problem.

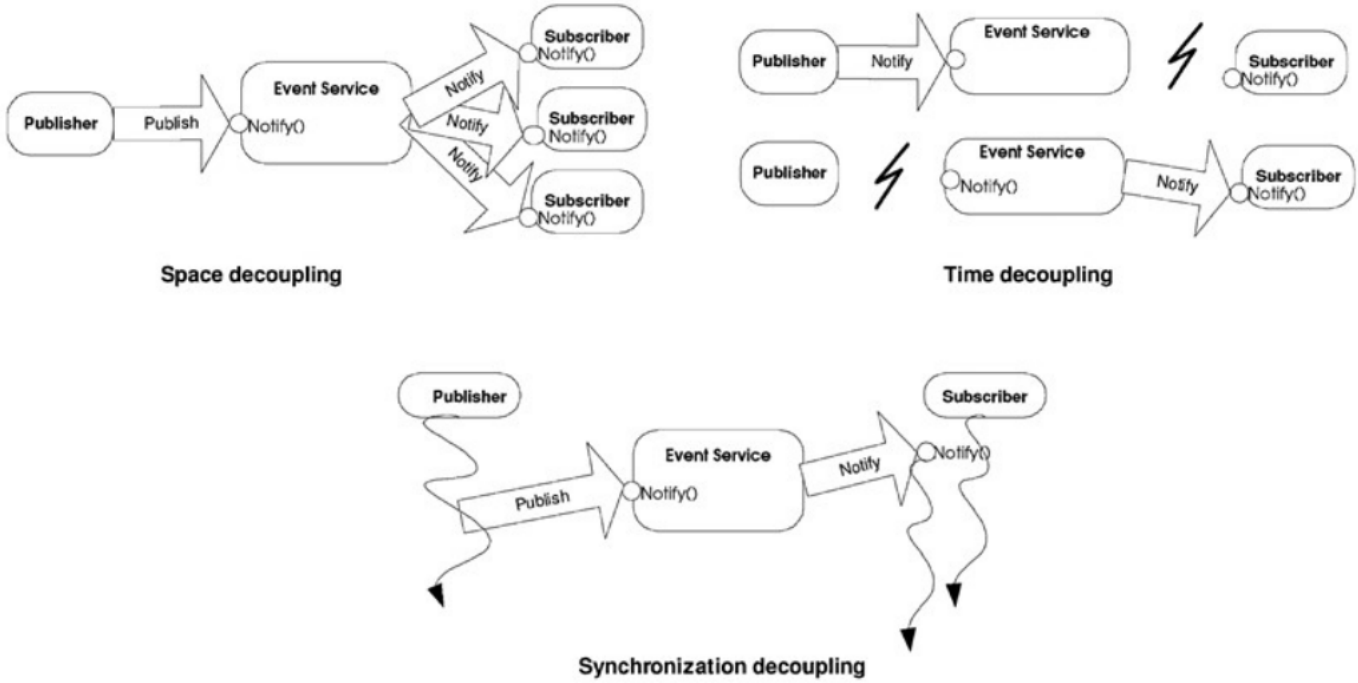

Figure 2.3: undefined undefined

Highlights In the proposed model (*sketched in 2.3*), publishers and subscriber **need to agree on topics beforehand** and publisher cannot assume that somebody is listening to the messages because is there are no subscriber messages are not readed by anyone.
Despite pub/sub do not need to known each other, they need to known the *hostame/port* of the **broker** beforehand, to connect and establish a TCP connection to it. In most application, the delivery of messages is near-real-time but in cases when the subscribers are offline, the broker is able to store the messages *only if* subscribers have been connected with a **persistent session** and they're already subscribed to the *topic*. In case of delay or unreliable message the retransmissions is necssary and the latency issues have to been taken into account. Also the deployment phase take time due to the time needed both by clients to connect to brokers.

MQTT decouples the syncrhonization allowing to obtain an *asyncrhonous model*: this model implies the use of callback functions on the subscriber side. Libraries that implements MQTT allows this async model by enabling callbacks. The complexities of the MQTT protocol are shifted to the broker work which in general represent a much more powerful machine respect to subscribers/clients.

As mentioned, the *subject-based* filtering of a message involves the use of *topic*: topic is based on a hirerarchy of topics that are meaning to the purposes of the appication. MQTT offers additional *QoS* in term of message reliability: it's based on top of TCP by adding an application level ackwnoledge. It's expressed by 3 levels of QoS (0, 1 ,2). The last two levels ensure the ACK at application level: the level 0 corresponds to TCP reliability level.

2.0.1 MQTT operations

MQTT includes operations at layer 5-6 of ISO/OSI layer, as indicated in 2.4.

CONNECT A client connects to a broker by sending a `CONNECT` message. This enstaurate a TC connection with the broker. The CONNECT Messages contains:

- *Client ID*: it's an optional identifier, if absent is setted to 1. It's necessary in case of persistent connection: allows to restore connection even in case the connection with the broker is broken, allowing to restore the session from the last *time* the exchange of the message happened. Using no client ID implicit declare the *no session state* so this setting the *Clean Session* flah to true.
- *Clean Session (Optional)*: it's FALSE if the client request a persistent session.

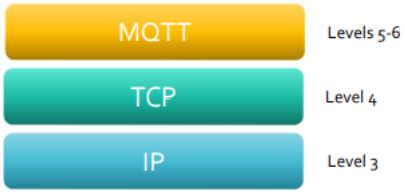

Figure 2.4: undefined undefined

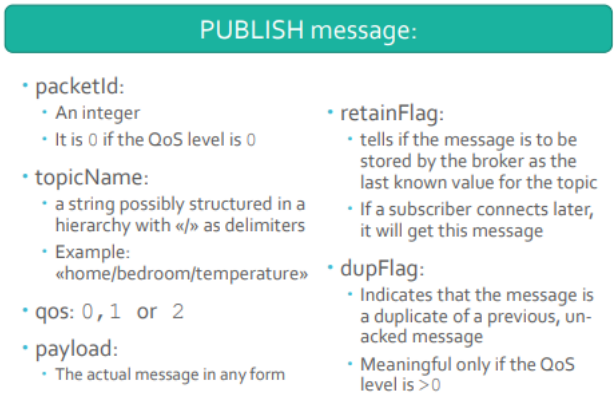

Figure 2.5: undefined undefined

- *Username/Password (optional)*: No encryption unless used with SSL
- *Will flags (optional)*: allows in case of disconnection ungracefully to sent a **last will message** to the subscribers by the broker (*in case of broken connection with the broker and/or damage*).
- *KeepAlive (optional)*: the broker expect to receive from the client periodic message of alive. If not received, the broker can assume that the client is dead and close the TCP connection (*possibly sending the last will message to each subscriber*). The `KeepAlive = 0` turn of this mechanism, indicating that the subscriber will never send the alive messages to the broker.

The broker acknoledges to the CONNECT message with the CONNECTACK message that states if the connection is accepted or rejected and also informs the client (in case of persistent session) information about the previous session.

PUBLISH

Each message contains a **topic** and a **payload** of whatever type because its managed as a bunch of bytes. Only the subscriber is able to understand the semantics of the payload.

The publish message is sent by the publisher to the broker and forgetted. Then it's forwarded by the broker to the subscriber. The `PUBLISH` message contains the following data (*summarized in 2.5*):

- `packetId`: an integer, is 0 if the QoS level is 0
- `topicName`Unknown Node :: textDirective string possibly structured in a hierarchy wih slash `/` delimeters
- `qos`: 0,1 or 2 (*see later*)
- `payload`: the actual message in any form (*usually interpreted as bytes*)
- `retainFlag`: tell if the message is to be stored by the broker as the last knoen value for the topic. If a subscriber connects later, it will get this message
- `dupFlag`: indicates that the message is duplicate of a previous un-ACKed message. This field is meaningful only if the `qos` is greather than 0.

  The `retainFlag` allows to send a specific message for later subscriber: store it to the broker.

When the **broke**r receives a `PUBLISH` message, it:

- acknowledges the message (*if requested, QoS¿0*)
- processes the message (*identify its subscribers*)
- delivers the message to its subscribers The **publisher**:
- leaves the message to the broker
- does not know whether there are any subscribers and whether or when they will receive the message

SUBSCRIBE There can be a difference of QoS between publisher and broker vs subscriber and broker: even in case of 3 different subscriber with different QoS, the broker will manage them independently. The structure of the `SUBSCRIBE` message is the following:

- `packetId`: an integer
- `topic1`: a string (*as in publish message*)
- `qos1`: 0,1 or 2 Usually a `SUBSCRIBE` message contains a list of `(topic, qos)` because a subscribe message can refer to multiple topics with the relative QoS.

The **broker** acknowledges the subscriber with a `SUBACK` message. It contains the following fields:

- `packetId`: the same integer of the relative `SUBSCRIBE` message
- `returnCode`: one for each topic subscribed. The value `128` indicates failures (*e.g. the subscriber is not allowed or the topic is malformed*) while `0,1,2` indicates success with the corresponding QoS granted (*that can be lesser than the QoS requested*).

UNSUBSCRIBE A client can unsubscribe a topic to stop receiving the related messages. The `UNSUBSCRIBE` message is composed by a `packetId` and a list of topics `topic1, topic2, ..., topic_n`. The `UNSUBACK` message have also a `packetId` and contains the same data of the `UNSUBSCRIBE` message.

2.0.2 Topics

Topics are strings that are organized into a hierarchy: each level is separated by a `/` *slash*. The subscriber can use **wildcards** to specify a group of topics:

- `home/firstfloor/+/presence`: select all presence sensors in all rooms of the first floor
- `home/firstfloor/#`: select all sensors in the first floor If using only a dash as a topic, you will subscribe to all topics, including the reserver one. Topics that begins with **"$"** are reserved for internal statistics of MQTT so they cannot be published by clients. An HiveMQ example:

```
$SYS/broker/clients/connected $SYS/broker/clients/disconnected $SYS/broker/clients/total
$SYS/broker/messages/sent
$SYS/broker/uptime
```

Topics must do not contain space, are short for memory storage optimization and packet size, use ASCII char UTF-8. It's also common practice to embed the clientID in topic or an *unique identifier* like `sensor1/temperature` so that onlt the client with the same `clientID` can publish such a topic.

2.0.3 Quality of Service

The QoS is an agreement between the send and the receiver of a message: the underlying TCP allows the QoS to gurantee delivery and ordering of the messages. In MQTT the QoS is an agreement between the *clients (publishers/subscribers)* and the *broker*. MQTT define three levels of QoS:

- **At most once** (`level 0`)
- **At least once** (`level 1`)
- **Exactly once** (`level 2`)

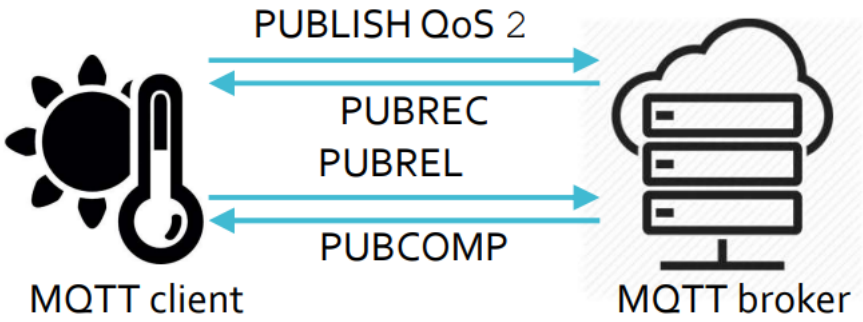

Figure 2.6: undefined undefined

So QoS is used both between publisher and broker and between broker and subscriber. It's called *best effort* delivery because messages are not acknoledge by the receiver. When used between publisher and broker messages are not stored by the broker and immediately forwarded. It provised the same guarantees as the TCP Protocol: it guarantees delivery as long as the connection remains. If one of the two peers disconnect there's not guarantee anymore.

Messages are numbered and stored by the broker until they are delivered to all subscribers with QoS level 1. Each message is delivered at least once to the subscribers with QoS 1. A message may be delivered more than once so it's suitable when the subscribers can handle the management of duplicated messages. Subscribers send acknowledgements by sending `PUBACK` packets.

It's the slowest mechanism to guarantee QoS because check that each message is received exactly once by the recipient. It uses a **double two-way handshake** by sending `PUBLISH` and the relative ACK by the `PUBREC` Packet. Respectively, the client send `PUBREL` to acknoledge the reception of the message: the broker now can drop the message from its storage and declare the transmission complete by sending a `PUBCOMP` packet. So for each message of QoS level 2 there is need four times the exchange of packets required by the level 0. To summarize QoS level 2, the figure 2.6, describe the entire exchange.

The **broker**:

1. receives the PUBLISH with a message
2. Process the PUBLISH
3. Sends back a PUBREC
4. Keeps a reference to the message until it receives PUBREL The **client**:
5. Sends PUBLISH with a message
6. Waits for PUBREC and then store PUBREC and discards the message
7. Sends back a PUBREL to inform the broker
8. Waits for PUBCOMP and then discards the PUBREC

The *double two-way handshake* is necessary because the first handshake allows to send the message, the second is to agree to discard the state. The `PUBREC` message alone is not sufficient because if it's lost, the client will send again the message so the broker needs to keep a reference to the message to identify duplicates. With `PUBCOMP` the broker known that it can discard the state associated to the message. This level of QoS is suitable when the clients *cannot manage duplicates* so the complexity to guarantee avoiding duplicates is managed by the broker that keep tracks of ACK and stores messages, with high overhead.

2.0.4 Persistent sessions

Persistent sessions keep the state between the client and the broker: if a subscriber disconnect, when it connects again, it does not need to subscribe again the topics. The session is associated with the `clientId` defined with the `CONNECT` message. In a persistent session the data stored are:

- All subscriptions to topics

- All QoS `1,2` messages that are not confirmed yet
- All QoS `1,2` messages that arrived when the client was offline A persistent session is requested at `CONNECT` time (*`cleanSession` flag setted to FALSE*): the `CONNACK` message confirms whether the session is persistent. Also clients have to **store state** in persistent session:
- Store all messages in QoS `1,2` flow, not ACKED By the broker
- All received QoS `2` messages not confirmed by the broker Messages of persistent sessions will be stored as long as the system allows it. The persistent mechanism can be avoided when refers to publishing-only clients that uses onlt QoS level 0 or if old messages are not important or when miss some data is not a problematic.

2.0.5 Retained messages

A publisher has to gurantee that its messages are actually delivered to the subscriber, even for later subscriber. When a client connects to the broker and subscrbes a topic, it does not known when it will get any message. For each topic, we can have **only one retained messages**, generally the *last one message* sent by the publisher. This way a new subscriber is immediately updated with the *state of the art*. A retained message is a normal message with the flag `retainFlag` setted to TRUE: the message is stored by the broker so if a new retained message is published, the broker will keep the last one received. When a *client* subscribes the topic of the retained message, the broker immediately sends the retained message for that topic: this can also works combined with **wildcards**.

The mechanism of **retained messages** and **persistent sessions** are completely decoupled: the second are messages kept by the server even if they had already been delivered. To delete a retained message is sufficient to publish a retained **empty** message of the same topic. This mechanism is usefull for *unfrequent updates* of a topic: for *example*, consider a device that update its status (`ON/OFF`) on topic `home/devices/device1/status` so if it publish ON, this status will remain for long time and if the message if retained, all subscribers will easily know the device is on.

2.0.6 Last will and testament

Last will and testament feature is used to **notify other clients about the ungraceful disconnection of a client**. At `CONNECT` tune, a client can request the broker a specific behavior about its last will: it's a normal message with `topic, retainFlag, QoS` and `payload` stored by the broker as a last will. When the broker detects the client is **abruptly disconnected**, it ends the last will message to all subscribers of the topic specific in the last will message. If the client send `DISCONNECT`, the stored last will message is discarded because the underlying assumption is that before correctly disconnect the clients has been already notified other clients. So, the broker sends a last will message if:

- Occurs an IO Network error
- The client does not send the `KeepAlive` message in time
- The client closes the network connection without sending `DISCONNECT`
- The broker closes the connection with the client because of a protocol error

The last will is specified in the `CONNECT` Message and contains 4 optional fields:

- `lastWillTopic`: a topic
- `lastWillQoS`: either 0,1,2
- `lastWillMessage`: a string
- `lastWillRetain`: a boolean flag
-

The last will message can be combined with **retained messages**: imagine a scenario in which a devices updates the status (`ON/OFF`) on topic `home/devices/device1/status` so it's powered and publish `ON` with a *retained message*. If the device crashes and then it abruptly disconnect, it does not publishes `OFF`: *last wll message* can be useful here because could be a retained message with payload `OFF` so the **subscribers and the futues ones are properly informed**, even in case of unwanted disconnection.

KeepAlive

The KeepAlive mechanism assure that a client and its connection with the broker is still **alive**: the client sends periodics messages to the broker that prove its liveness. The *frequency* of these messages is declared in the `CONNECT` message. The keep alive message must be sent by the client before the connection expiration of the interval set with the CONNECT message. The entire process is sketched in 2.7

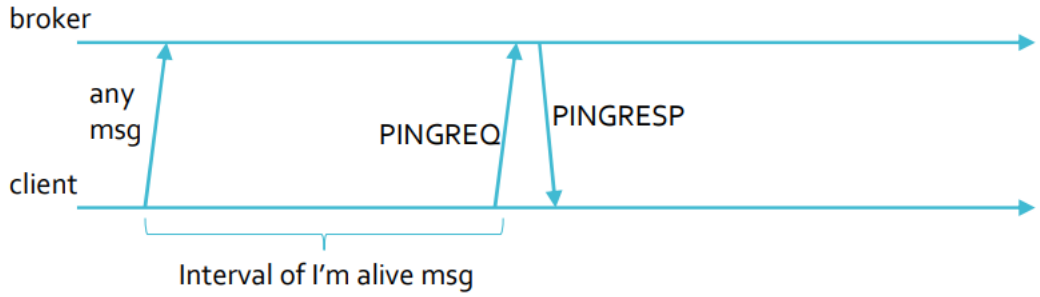

Figure 2.7: undefined undefined

KeepAlive timer (*managed by the broker to control the liveness*) is resetted by both `PINGREQ` messages and by pulbish a message on a topic: this allows to be consired alive by the broker. So if the client does not send `PINGREQ` in time or any other message, the broker turns off the TCP connection and sends the last will message.

2.0.7 Packet format

The structure of an MQTT **control packet** is the following:

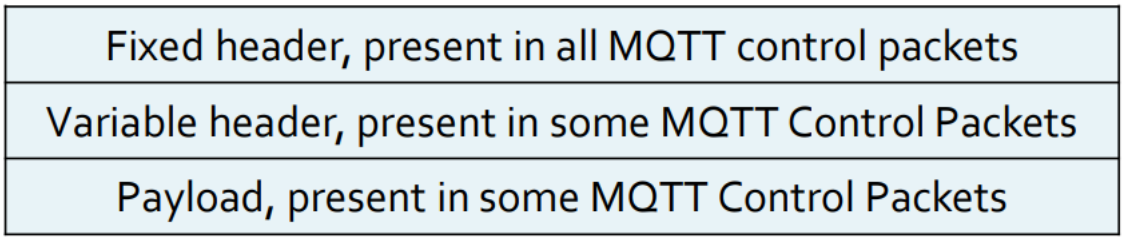

Figure 2.8: Control packet structure

The **fixed header** is composed by **2 bytes**, the first contains (2.9):

- *MQTT Control packet type*: 4 bit (*range 7 to 4*)
- *Flags specific to each MQTT control packet type*: 4 bit (*range 3 to 0*) The second byte contains the **remaining length** which is the **length** of the variable header and payload: 7 bits are used to encode the remaining length while one bit (*the MSB, index 7*) is a flag that specifies that there is another field.

The **control packet type** are pictured in 2.10

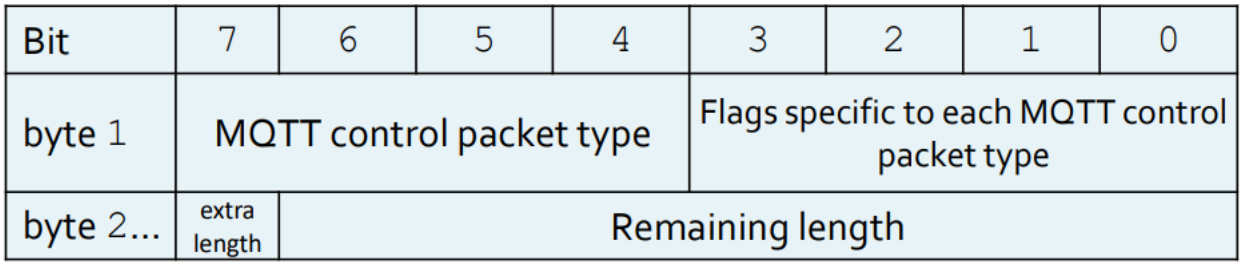

Figure 2.9: Fixed header structure

Name	value	direction of flow
reserved	0	forbidden
CONNECT	1	client to server
CONNACK	2	server to client
PUBLISH	3	client to server or server to client
PUBACK	4	client to server or server to client
PUBREC	5	client to server or server to client
PUBREL	6	client to server or server to client
PUBCOMP	7	client to server or server to client
SUBSCRIBE	8	client to server
SUBACK	9	server to client
UNSUBSCRIBE	10	client to server
UNSUBACK	11	server to client
PINGREQ	12	client to server
PINGRESP	13	server to client
DISCONNECT	14	client to server
reserved	15	forbidden

Figure 2.10: Control packet type

The **variable header** contains the `packetId` (*encoded with 2 bytes)*, only the packets regarding the `CONNECT` and `CONNACK` does not contains the `packetId` while the `PUBLISH` Ppacket contains this information only if `qos > 0`. It can contains other information depending on the control packet type: for example, `CONNECT` packets include the protocol name and version, plus a number of flags (*see CONNECT).** The **payload** can be empty or, in case of `CONNECT` message it contains data about the client or for `PUBLISH` can be optional (*empty payload are allowed, e.g. to reset retained message*). For example, for `CONNECT` Packet includs:

- client identifier (mandatory)
- will topic (optional)
- will message (optional)
- Username (optional)
- Password (optional)

In the table 2.11 is listed when the payload is required.

2.1 MQTT on Arduino

Include only a subset of MQTT specific, excluding:

- No SSL/TLS
- QS levlw 2
- Payload limited to `128` bytes

Control packet	payload
CONNECT	required
CONNACK	none
PUBLISH	optional
PUBACK	none
PUBREC	none
PUBREL	none
PUBCOMP	none
SUBSCRIBE	reserved
SUBACK	reserved
UNSUBSCRIBE	reserved
UNSUBACK	none
PINGREQ	none
PINGRESP	none
DISCONNECT	none

Figure 2.11: MQTT required payload commands

Constructors

- `PubSubClient()`
- `PubSubClient (server, port, [callback], client, [stream])`: create a fully configured client instace. The API includes other *PubSubClient* with lesser parameter. The parameters are:
  - Server address: IP address of the broker (*as an `IPAddress` object*)
  - Port: port used by the broker. Because SSL/TLS is absent, usually is the 1883.
  - Callback: a pointer to a function that allows to distinguish between subscriber, publisher or both. They are implemented on top of interrupts and allows to intercept publishing/invokation.
  - Client: an instance of the `Client` class that cna connect to a specified internet IP address and `portCLlient` (*e.g `EthernetClient`*)
  - Stream: an instance of `Stream`, used to store received messages (*e.g. `mqtt_stream`*)

Functions

- `boolean connect(clientId, username, password, willTopic, willQoS, willRetain, willMessage)`: connect the client, specifyng the **will message**, username and password (*as basic authentication mechanism*)
- `void disconnect`
- `int publish(topic, payload, length, retained)`: publishes a message to the specified topic, with the **retained flag** as specified. Return `FALSE` if publish fails, either due to connecion lost or message too long, while return `TRUE` if publish succeeds. The API includes other `publish()` with lesser methods.
- `boolean subscribe (topic, [qos])`: topic is a `const char []`, payload is a `byte[]`, length is `byte`, retained is `boolean` and qos is an `int`.
- `boolean unsubscribe(topic)`:
- `boolean loop()`: to support the keepAlive features

	MQTT	HTTP
pattern	publish/subscribe	client/server
focus of communication	single data (byte array)	single document
size of messages	small and small header	large, details encoded in text form
service levels	3 QoS levels	same service level for all messages
kind of interaction	1 to 0; 1 to 1; 1 to N	1 to 1

Figure 2.12: MQTT vs HTTP

- `int connected()`: check whether the client is connected to the server
- `int state()`: return the current state of the client The following function configure the parameter to the server if still not initialized by the constructor:
- `PubSubClient setServer (server, port)`
- `PubSubClient setCallback (callback)`
- `PubSubClient setClient (client)`
- `PubSubClient setStream (stream)`

See more bootstrap example at https://pubsubclient.knolleary.net/api.

2.1.1 MQTT Competitors

HTTP is a valid alternative to MQTT, despite it's not speficially designed to operate for IoT devices at appliaiton level. A comparison is sketched in 2.12.

HTTP used in IoT device can be used in two different ways:

1. Clients are server and accept accept connection from services
2. Service is the server and accept incoming connection from clients (devices) This paradigm poses problems on scalability, also considering the size of messages and the focus of communication (*because HTTP is not suitable for constraned devices*). In real world application both solution are deployed (*e.g. TeamSpeak, AWS IoT*) because nowadays devices are not entirely constrained on power/computation capacity.

There are several **limitations** on MQTT: the need for a centralized broker can be limiting in scenario wher we have several point-to-point communication because the overhead of a broker may easily become not compatible with end devices capabilities as the network scales up. Also, the broker is a **single point of failure** and the underlying TCP protocol does not come for free: it's not cheap for low end devices. Exists also an MQTT based on UDP, not standardized nowadays.

2.1.2 Brief CoAP (Constrained Application Protocol)

Standardized in `RFC-7252`, it's specialized for **web transfer** and it's suitable for machine-to-machine for use with constrained nodes and constrained networks. It used under the hood the UDP Protocol. The general key idea is pictured in 2.13

CoAP also support a direct conenction between devices, without relying on an intermediate bridges to translate/support the communication.

CoAP is designed to work with nodes with *8 microcontrollers* with small amounts of ROM and RAM. Constrained networks such as IPv6 over 6LoWPANs: it concerns the size of the header, compressing addresses and header used in IPv6. The main strenghts are:

- Native UDP
- Multicast
- Security embedded in the specification

- Asyncrhonous communication While, the weaks are:
- Standard maturity
- Message reliability noy quite sophisticated, similar to MQTT QoS

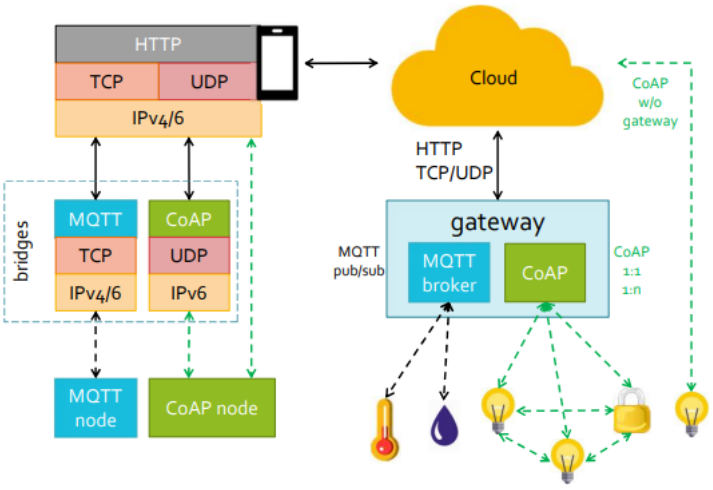

Figure 2.13: MQTT vs CoAP

Chapter 3

ZigBee

It covers almost all layer from phyisical to application layer. It's a stamdard for wireless sensors and actuators networks: it's considered an IOT standard even if use TCP/IP as its internal mechanism. The protocol is developed and promoted by industrial private alliance called ZigBee alliance.

It covers different application cases like *home automation, health care, consumer and industrial automation*. It's a competing standard with Bluetooth, especially in eHealth environment where ZigBee failed but had more successfull application in other sectors.

The main requirements that drove the designing of the protocol were:

- Network completely autonomous, self-organiziting with minimal or nothing manual intervention
- Very long battery life: ideally a ZigBee device should last 10 years
- Low data rate: allows to guarantee a very long battery life by optimizing performances. This allows trading performances for battery life.
- Interoperability of ZigBee devices from different vendors

This criteria result in some main features, like:

- Standard based
- Low cost
- Can be used globally: it's complex to guarantee because in the range of private frequencies, there is a limit of power. This constraints are country-specific but almost all countries have a subset of frequencies in common.
- Reliable and self-healing
- Support large number of nodes: up to 30.000 devices that using short range (200m) communication. For larger distances there is the need to introduce *multi hops networks*.
- Easy to deploy (by definition)
- Very long battery life
- Secure embedded

Despite is not mentioned, the *transport layer* features are implemented by some sub-layers between network and MAC layer.

IEEE 801.15.4 Standard Define the specification of the physical and MAC ayers for low rate PAN. It operates on top of the frequencies of WiFi and Bluetooth: using the same frequencies but not perceived or talk each other, just consider each other as *noise*. The frequency that are defined by the specification are:

- 868–868.6 MHz (e.g., Europe) with a data rate of 20 kbps;
- 902–928 MHz (e.g., North America) with a data rate of 40 kbps; or
- 2400–2483.5 MHz (worldwide) with a data rate of 250 kbps

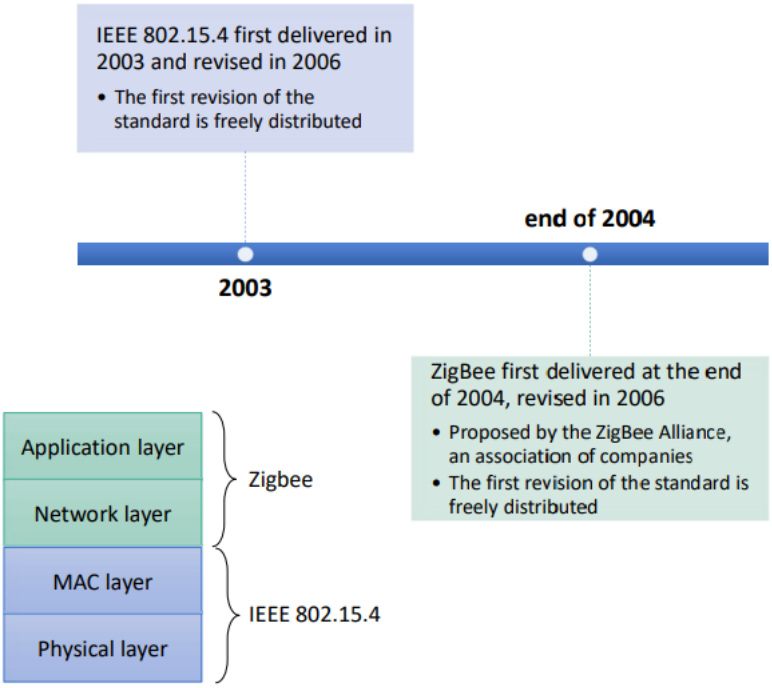

Figure 3.1: undefined undefined

3.0.1 ZigBee Standard

The components described in the ZigBee specification are pictured in 3.2.

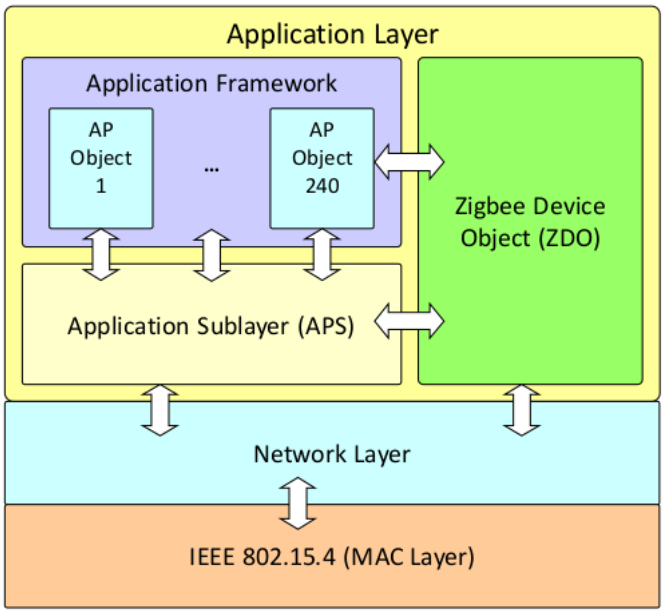

Figure 3.2: ZigBee Specification overview

The Application layer of ZigBee is defined by 3 sub-layer:

1. Application Framework: contains ip to 240 **Application Objects (APO)** in which each APO is a user defined ZigBee application.
2. ZigBee Device Object (ZDO): provides services to let the APOs organize into a distributed application. Guarantee interoperability between different devices of different vendors
3. Application Support sublayer (APS): provides data and managemnt services to the APOs and ZDO.

Service primitives

Each layer provides its data and magement service to the upper layer so **each service** is specified by a set of primitives of four generic types. Between all the services and all the layer, only **4 primitives** are used/consumed between each layer as sketched in 3.3:

- **Request**: It is invoked by the upper layer to request for a specific service;
- **Indication**: It is generated by the lower layer and is directed to the upper layer to notify the occurrence of an event related to a specific service;
- **Response**: It is invoked by the upper layer to complete a procedure previously initiated by an indication primitive;
- **Confirm**: It is generated by the lower layer and is directed to the upper layer to convey the results of one or more associated previous service requests. Services may not use all the four primitive (*see later*).

Here an example of *service primitives* that allows to visualize the flow: same levels of different nodes interact by means of those 4 primitives.

Data trasnfer The data transfer between node assume an uniform name called **Data Unit**, encapsulated at different level with different packet layers, here specified in 3.5.

For each layer:

- *Application Protocol Data Unit*
- *Network Protocol Data Unit*
- MAC Protocol Data Unit
- *Physical Protocol Data Unit*

3.0.2 Network Layer

At network level we identify 3 types of devices:

1. **Network coordinator**: Despite been a infrastructurless protocol, a coordinator have the function of create the network by taking decisions/interfacing with the rest of the network. Allows to bootstrap the network.
2. **Router**: provide routing features in a large network and can also have services capability. From the point of view of routing capabilities, they have the same capabilities or coordinator.
3. **End device**: they are not able to communicate alone with the rest of the network and use coordinators/routers to communicate. They must be attach to the coordinator to be able to communcate over the network. They do not need to implement the entire standrd, differently from router/coordinators that must have the capabilities to implement the standard fully. Generally they correspond to a RFD or FFD with simple features (***RFD***: *Reduced Functional Interface*, ***FFD***: *Full Functional Device*)

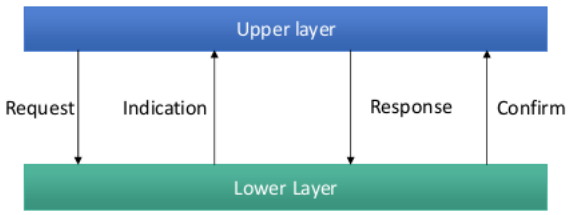

Figure 3.3: undefined undefined

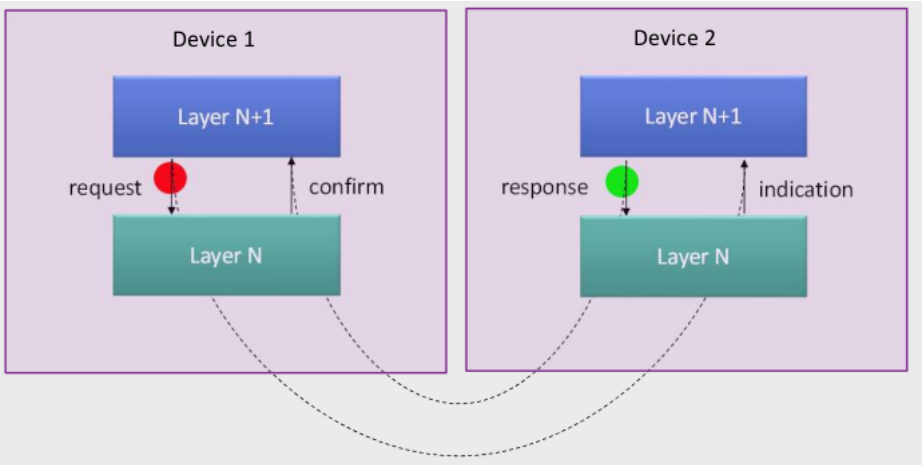

Figure 3.4: undefined undefined

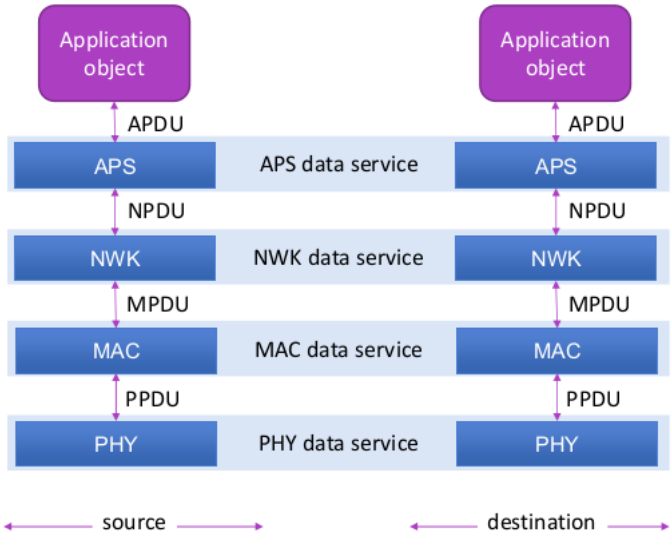

Figure 3.5: undefined undefined

We can have different **network topologies** as specified in 3.6.

In a **star shaped topology**, the router have the same roles as end device: they uses the *superframe*. In the **tree topology**, the *multi-hop network* (*not fully addressed in those notes*) functionalities are supported and the nodes can communicate only via the intermediary routers: they can or cannot use *superframes*. In **mesh networks**, the communication is possible even between nodes witout relying on the *superframe infrastructure* (*superframe is the topic of the lecture on CSMA/CA, see later chapters*).

The network layer provides services for:

1. Data transmission (both unicast & multicast)
2. Network initialization
3. Devices addressing
4. Routes management & routing
5. *Management of joins/leaves of devices*: because the networks is formed incrementally, formed a network with an arbitrary topology.

In 3.7 table that summarizes which services uses the 4 primitives already mentioned:

As highlight the table, not all four primitives are used in the listed operations: particularly, the `GET` and `SET` are functionalities used **locally to set/get parameters of the network layer**, internally, without using communication means.

Before communicating over a network, a ZigBee must either:

1. Form a network so it's a **ZigBee coordinator**
2. Join an existing network so it's a **ZigBee router/end-device** The role of the device is chosen at **compile-time** by specifing the code ad application level when writing the **Application Object** code.

Network Formation The *network formation* is initiated by the `NETWORK-FORMATION.request` primitive: it's invoked by a device that acts as a **coordinator** and that is not already in another PAN. The operation uses the MAC layer services to **look for a channel** that doe snot conflict with other existing networks: it **selects a PAN identifier** which is not alredy in use by other PANs. The process is pictured in 3.8

1. At application layer is invoked the `NETWORK-FORMATION.request`
2. The network layer execute an *energy scan* to verify if there is noisy on the channel
3. The MAC layer return the result of the energy scan: at this point, the network layer request a second scan, differently from the first because it's an **actve scan** to lookup nearest PAN networks. This is due to avoid to have two different PAN with the same identifier so a preventive scan allows to select a PAN ID unique among the already existing.
4. The MAC Layer return a list of existing PAN ID and the Network Layer now can set the PAN ID.

The process of `NETWORK-FORMATION` continues as showed in 3.9

Now the Network later can configure the MAC Layer starting from the time the `START.request` is sent and the MAC Layer can start to send beacon frames. After that, the MAC layer can answer with a

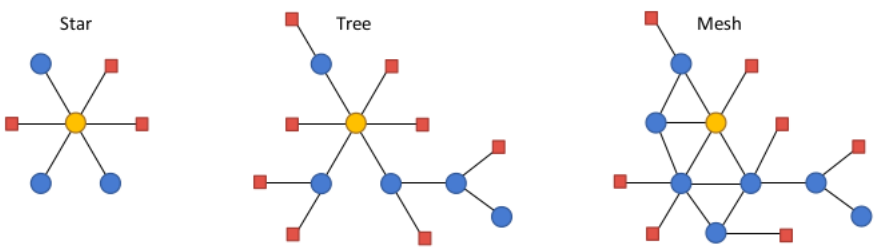

Figure 3.6: undefined undefined

Name	Request	Indication	Confirm	Description
DATA	X	X	X	Data transmission service
NETWORK-DISCOVERY	X		X	Look for existing PANs
NETWORK-FORMATION	X		X	Create a new PAN (invoked by a router or by a coordinator)
PERMIT-JOINING	X		X	Allows associations of new devices to the PAN (invoked by a router or by a coordinator)
START-ROUTER	X		X	(Re-)initializes the superframe of the PAN coordinator or of a router
JOIN	X	X	X	Request to join an existing PAN (invoked by any device
DIRECT-JOIN	X		X	Used by the coordinator or by the routers to force an end device to join their PAN
LEAVE	X	X	X	Leave a PAN
RESET	X		X	Resets the network layer
SYNC	X		X	Allows the application layer to synchronize with the coordinator or a router and/or to extract pending data from it
GET	X		X	Reads the parameters of the network layer
SET	X		X	Set parameters of the network layer

Figure 3.7: undefined undefined

`START.confirm` that will be translated also by the network layer with a `NETWORK-CONFIRMATION.confirm` to the application layer to confirm that the PAN has been created.

To **join** inside a PAN network there are two ways:

1. The end-device or the router can communicate with the coordinator to request the access: this method is called **Join Through Association**
2. The coordinator request a device to join the network: this method is called **Direct Join**

Join through association (`CHILD-SIDE`) The protocol flow is pictured in 3.10

The device that wish to join a PAN:

1. Performs a `NETWORK-DISCOVERY` to look for existing PANs

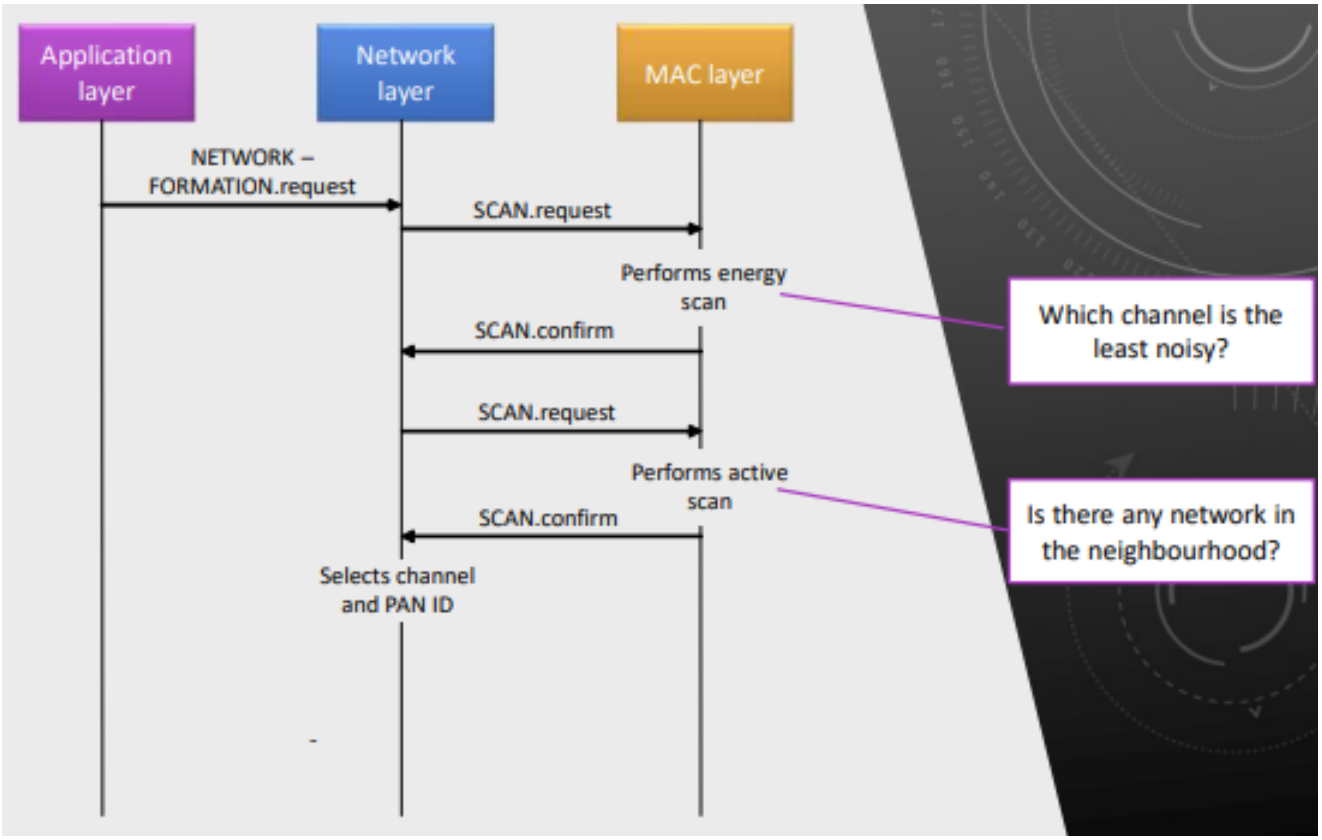

Figure 3.8: Network Formation exchange diagram

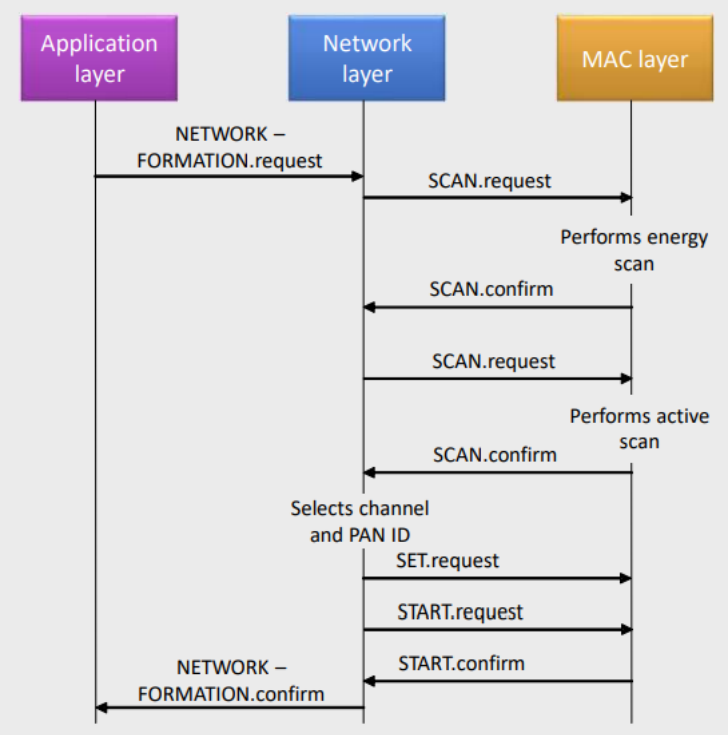

Figure 3.9: undefined undefined

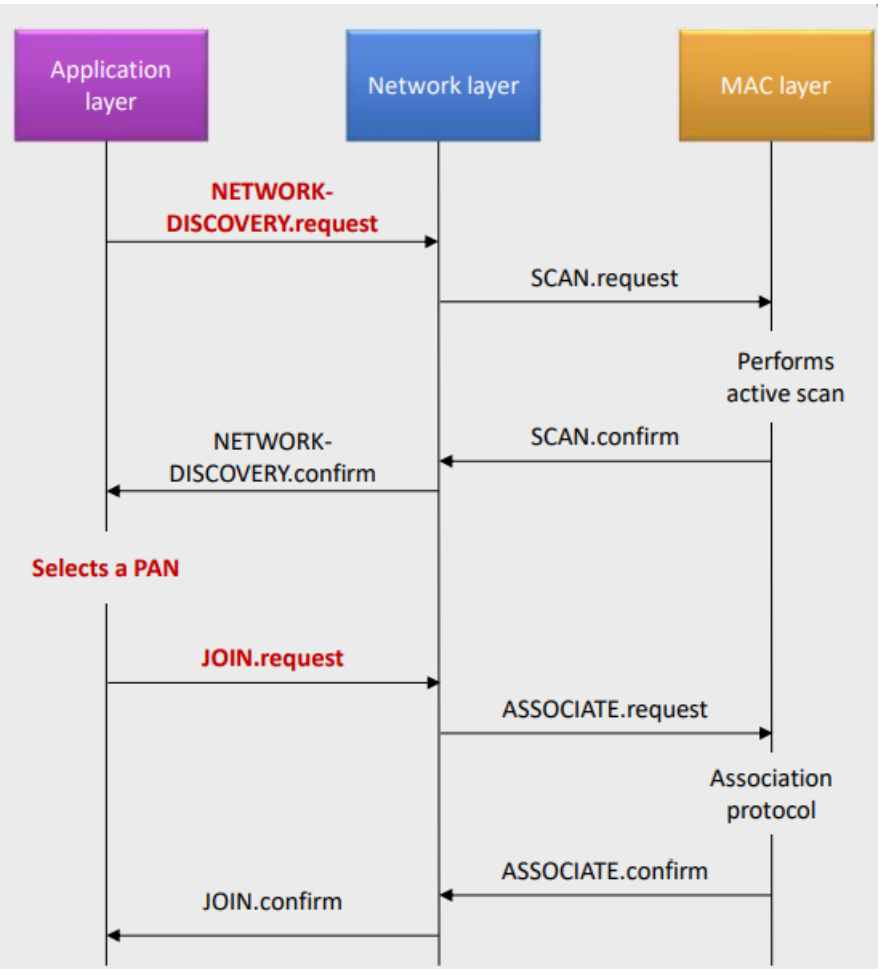

Figure 3.10: Join Through Association

2. Select a PAN

3. Invokes `JOIN.request` with parameters like the *PAN Identifier* of the slected network, a *flag* indicating whether it joins as a router or as an end device. This request in the *network layer* selects a "*parent node P*" (*in the PAN*) from its neighbourhood. In a **star topology** `P` is the coordinator and the device join as end-device, while in a **tree topology** `P` is the coordinator or a router and the device can join as a router or as an end-device.

4. The association protocol obtains from `P` a $16-bit$ **short address**: the `ASSOCIATE.confirm` returns to the netowkr layer the short address that will be used in any further communication over the network. The `JOIN.request` is triggered at application level: this require preliminarly to scan the network to find a free channel to communicate. The scan is perfomed troguht `NETWORK-DISCOVERY` which return a list of PAN network to connect: this allows to select the PAN and send the `JOIN` request. Depending on network topology, the association of a new device can be structured by identified implicitly a **root father** so the topology assume the form of a sort of a tree: the end-devices will be the leaves while the internal nodes are the router, the *coordinator* is the root node. This tree-structure topology is used to assign the addresses to new devices, coordinated by the *ZigBee coordinator* that decides and control parameters like *number of leaves, number of routers, depth of the tree* and so on. Each router have assigned its own set of addresses to assign to new joined devices: the assignment of the address is performed by ensuring that new devices are highest in the network and are at a lower number of hops from the root/parent router.

3.0.3 Application Layer

The application layer comprends various components (*as shown in 3.11*):

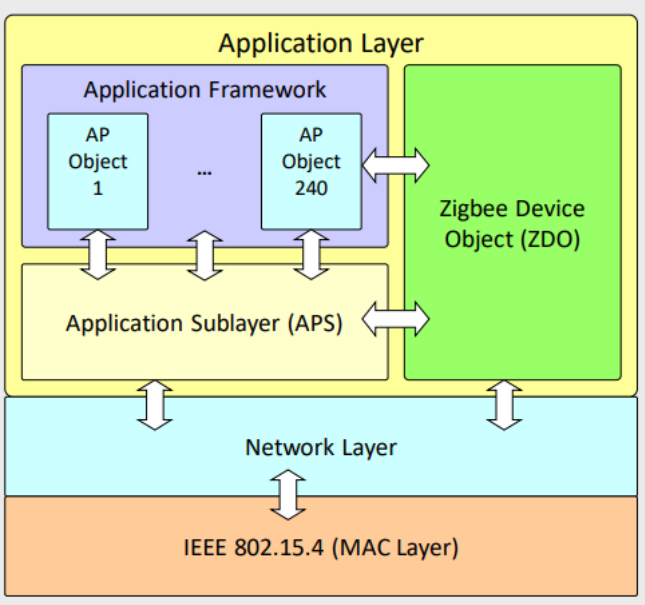

Figure 3.11: undefined undefined

1. **Application Framework**: contains up to 240 **Application Objects (APO)** in which each APO is a user defined ZigBee application.

2. **ZigBee Device Object (ZDO)**: provides services to let the APOs organize into a distributed application. Guarantee interoperability between different devices of different vendors

3. **Application Support sublayer (APS)**: provides data and managemnt services to the APOs and ZDO.

1. Application Framework The application framework can contains up to 240 different *Appication Object (APO)*, each one uniquely identified inside the netowrk. Simplest APO can be queried with *Key Value Pair data service (KVP)* by using primitives like `Set/Get/Event` transactions on the APOs

attributes: native KVP disappeared in the second version of ZigBee but it's now present in the cluster library. Complex APO can have conpelx states and communicate with the *Message data Service*. In figure 3.12 an example of complex APO: we have two devices, the first with two endpoint, the second with three endpoint: the logic of this device can be easily implemented using two `ON-OFF` attribute and two endpoint in the device A. **APOs 5B, 6B, and 8B** have a single attribute containing the status of the bulbs (on/off) that is configured as servers at the application layer while **APO 10** and **APO 25** are switches, and they are configured as clients at the application layer. The attributes of **APOs 5B, 6B**, and **8B** can be set remotely from the **APOs 10A** and **APO 25A**.

2. Application Support Sublayer (APS) The APS define the **endpoint, cluster, profile IDs and device IDs**. It's responsible for:

- Data service (*light transport layer*) that provides support for the communication's reliability and packet filtering
- Management services which includes mantaining a local binding table, a local group table and a local addresss map.

The **endpoints** are identified by a number between 1 and 240: an endpoint can be seen as the equivalent of a Unix socket, connecting virtual wires to applications. The **cluster** is a protocol that can be used to implement the functionality of a given application: cluster defines the attribute that contains the information about the APO. A cluster is defined by an identifier of ***16 bit***, local inside the **profile**. In ZigBee a **profile** defines a set of application layer protocols that enable interoperability between devices that perform specific functions or serve specific purposes. A profile specifies the standard ways in which devices should communicate and exchange information, making it easier for devices from different manufacturers to work together.

A profile consists of one or more **clusters**, which are groups of related functionality that can be supported by a device. Each cluster is identified by a unique ***cluster ID**, which is a 16-bit integer value.

A cluster defines a specific set of messages and commands that devices can use to communicate with each other. *For example, the ZigBee Home Automation profile includes clusters for controlling lights, thermostats, and door locks. Each cluster specifies the data format and message types that devices should use to exchange information related to that functionality.*

In the previous table, the *Basic Cluster* with ID `0x0000` allows to retrieve basic information about the device, the *Power Configuration Cluster* allows to set the *minimum and the maximum temperature* of the device. Also, the *OnOff Cluster* allows to set the *switch* ON or OFF, as expected.

The *APS **Device IDs*** range from `0x0000` to `0xFFFF`: this value has two main purposes:

1. To alows human readable displays (*e.g. an icon is realted to a device*)
2. Allows ZigBee toools to be effective also for humans:
   (a) A device may implement the on/off cluster but you don't know whether it's a bulb or a oven so you only kwno you can turn it on or off. The DeviceID tells you what it is, but it does not tell you how you can communciat ewith it: this is given by the IDs of the cluste rit implements.

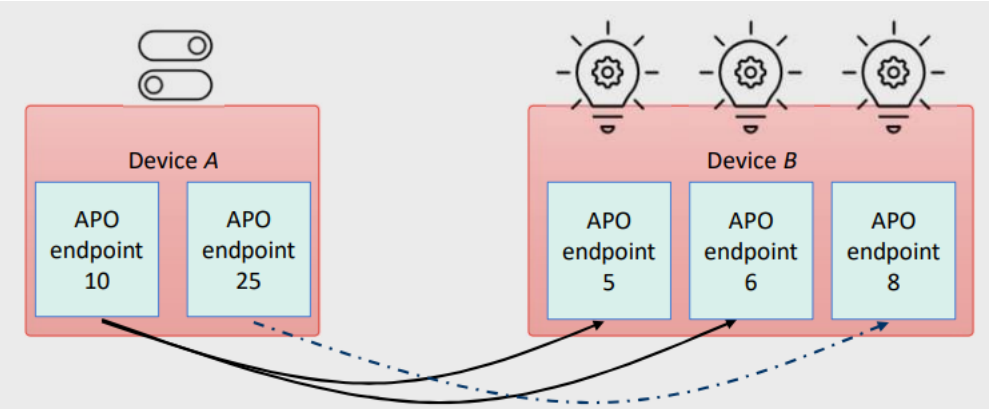

Figure 3.12: APO complex example

Cluster Name	Cluster ID
Basic Cluster	0x0000
Power Configuration Cluster	0x0001
Temperature Configuration Cluster	0x0002
Identify Cluster	0x0003
Group Cluster	0x0004
Scenes Cluster	0x0005
OnOff Cluster	0x0006
OnOff Configuration Cluster	0x0007
Level Control Cluster	0x0008
Time Cluster	0x000a
Location Cluster	0x000b

Figure 3.13: undefined undefined

Name	Identifier	Name	Identifier
Range Extender	0x0008	Light Sensor	0x0106
Main Power Outlet	0x0009	Shade	0x0200
On/Off Light	0x0100	Shade Controller	0x0201
Dimmable Light	0x0101	Heating/Cooling Unit	0x0300
On/Off Light Switch	0x0103	Thermostat	0x0301
Dimmer Switch	0x0104	Temperature Sensor	0x0302

example from the Home Automation Profile

Figure 3.14: undefined undefined

ZigBee discovers services in a network based on profile IDs and cluster IDs, but not on device IDs because each device is specified by the combination of Application profile and cluster ID. Here example of device IDs:

The **APS *Services*** provides data service to both APOs and ZDO: allows to ***bind a service to the ZDO*** and ***group management services*** (*see later*). APS data service enables the exchange of messages between two or more devices within the network: the data is defined in terms of primitiviles like `request(send), confirm(return status of trasnmission), indication(receive).`

The reliability of a data service is ensured by using the **ACKNOWLEDGE messages** for data trasmission as the schema 3.15 indicates.

The ***APS Group Management*** provides service to build and mantains groups of APOs: a group is identified by a $16-bit$ address and each APO in the group is identified by the pair network *address/endpoint*. The `ADD-GROUP` primitive allows to add an APO to a group and `REMOVE-GROUP` primitive to remove an APO from a group: takes group number and endpoint number; if the group does not exists it's created. The nformation about groups are stored in a *group table* in APSs.

APS Binding The binding operation allows an endpoint on a node to be conncted (*bound*) to one or more endpoints on another nodes: the binding is **unidirectional** and can be configued only by te ZDO of the coordinator or of a router. So the binding provide a way to specify the destination address of the messages: this method is called **Indirect Addressing**. When a Message is routed to

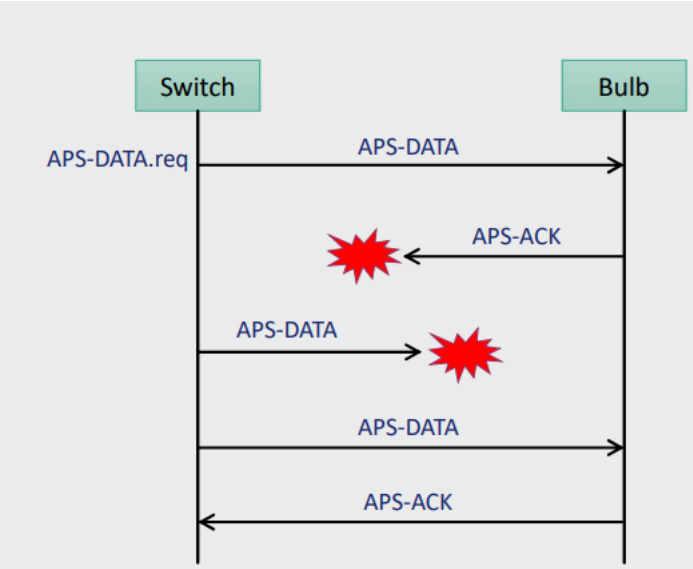

Figure 3.15: undefined undefined

the destination APO based on its pairs `<destination endpoint, destination network address>` the method is usually called **Direct Addressing**. Direct addressing might be unsuitable for extremely simple devices while *Indirect addressing* exploit the ***Binding table***.

Two main primitives are used for binding, respectively `BIND` and `UNBIND`:

1. `BIND.request` creates a new entry in the local binding table. It takes in input the tuble `<source address, source endpoint, cluster identifier, destination address, destinatione endpoint>`
2. `UNBIND.request` deletes an entry from the local binding table

In the **Indirect addressing**, the binding table matches the *source address* `<network addr, endpoint addr>` and the *cluster identifier* into the pair `<destination endpoint, destination network address>`. The binding table is stored in the APS of the ZigBee cordinator and/or of the other router: it's updated on explicit request of the ZDO in the routers on in the coordinator. It's usually initialised at the network deployment.

In the scenario of light bulb and a switch, can be complex from the switch side to know to which light bulb connect but, this information is kwnown by the installer. Despite operating directly on the device, the installer can set and entry in the **coordinator's binding table** so that the messages arriving to the switch will be forwarded to the light bulb.

Here an example of **binding table** with a request `APS-DATA.req` of cluster 6 from EP 5 on node `0x3232` that in **indirect routing** will generate 4 data requests:

- Node `0x1234`, endpoint `12`
- Multicast to group `0x9999`
- Node `0x5678`, endpoint `44`

In the binding table showed in 3.16, the `Src Addr` and the `Dest Addr` are formatted as a MAC address: this choice has been made because the devices can disconnect/re-connect to the network for many reasons, requiring a reconfiguration of the addresses in the binding table. The column `Addr/Grp` allows to tag as destination a group of devices (`G`). Also, the same devices can be the source of dirrent messages (`Src EP = Endpoint`).

The *APS **Address Map*** 3.17 allows to associated the $16-bit$ network address with the $64-bit$ IEEE MAC address: ZigBee end devices (*ZED*) may chabge their 16 bit network address when they leave and join again. In that case, ana nnouncement is sent on the network and every node updates its internal table to preserve the binding.

3. ZigBee Device Object (ZDO) ZDO is a *special application* attached to ***endpoint*** `0`: it implement the basic functionalities expected by the devices. It also implement the end-device or router or coordinator based on the specific configuration given by the **ZigBee Device Profile** that describe the cluster that

must be supported by any ZigBee device, defines how the ZDO implements the services of discovery and binding and how it maanges the network and security. So the main ZDO services are:

- Device and service discovery
- Binding management
- Network management
- Node management

Device and service discovery The **ZigBee Device Profile** specifies the device and the *service discovery mechanisms* to obtain any information about devices and services in the network. The **device discovery** allows a device to obtain the (*network or MAC*) address of other devices in the network: starting from an unknown address it allows to identify the relative device inside the network. It follow an **hierarchical implementation** in which a router return to its parent its address and the address of all the end-devices is associated to itself. Then the coordinator returns the address of its associated devices.

A newly joined device does not know anything about the network: to provide a service it must also known, require and use the services offered by other devices. This actions are performed thanks to **ZigBee Device Profile** that allows thhe service discovery of other devices. The service discovery send queries that are based on *cluster ID* or *service descriptors*: the coordinator responds to service discovery queries by returning lists of endpoint addresses matching with the query: the hierarchical implementation is effective because *each router colelct information from its associated devices and forwards it to its parents.*

The other mechanism to understand what service a device is providing is called **Device Discovery**: the network is created according to a parent-child relation (with a coordinator), building as a tree (as the addresses tree previously mentioned). To known which service a device offer, a joined device ask to its coordinator that may not have fully all the information and may ask to it's parent node, recursively. Each node is responsible for the subtree under its domain: in case of missing information, intermediate routers prepare information report to answer the queries.

Src Addr (64 bits)	Src EP	Cluster ID	Dest Addr (16/64 bits)	Addr/Grp	Dest EP
0x3232...	5	0x0006	0x1234...	A	12
0x3232...	6	0x0006	0x796F...	A	240
0x3232...	5	0x0006	0x9999	G	_
0x3232...	5	0x0006	0x5678...	A	44

Figure 3.16: undefined undefined

IEEE Addr	NWK Addr
0x0030D237B0230102	0x0000
0x0030B237B0235CA3	0x0001
0x0031C237b023A291	0x895B

Example of APS address map

Figure 3.17: undefined undefined

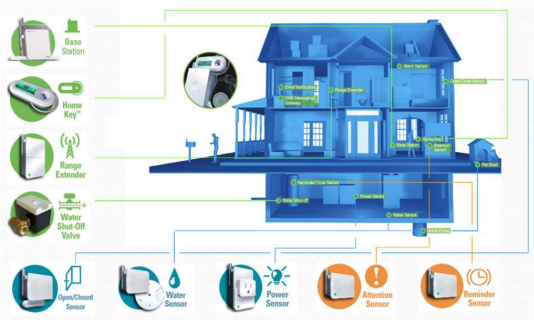

Figure 3.18: undefined undefined

The **Service Discovery** provides a report that tells for a single device what are the cluster to which it responds: for example, a device can responds telling that implement a ON/OFF cluster. The service discovery can be direct to the coordinator and is it's responsability to collect information and answer for the whole network. The requests can be directed also to end-device but due to low-power devices, they may not be ready to answer the queries regarding the services they offer. The service discovery does not tell where the device is physically located, only the service offered and their relative identifier (Cluster ID, Device ID, etc): to obtain the location is necessary that a device implements the **Identify Cluster** that force a device to identify itself by blinking a led.

Binding management The ZDO processes the binding requests received from local or remote EP: both to add entries and delete entries from the APS binding table (*by invoking primitives*). This process require having an IEEE address.

Managing network and nodes The *network management* implements the protocols of the coordinator, a router or an end-device accordin gto the configuration settings established either via a programmed application or at installation. Differently, the *node management* involves the ZDO to serves incoming requests aimed at perfoming network discover, retrieving the routing and binding tables of the device and managing `join/leaves` of nodes to the network.

3.0.4 ZigBee Cluster Library (ZCL)

The ZCL is a repostory for cluster functionalities: it's a *working library* with regular updates and new functionalities. The develoeprs are expected to use the ZCL to find relevant cluster funcionalities to use for their application. The main goal is to avoid re-inventing the wheel, supports **interoperability** and facilitates maintanability: two devices must implement the same application layer to use each other. The ZCL allows to implement the same set of commands and functionalities, standardized by ZCL. Let's consider the example of home automation cluster sketched in 3.18

In a base home automation system we have many application execute at the same moment: the ZCL allows to specify the behaviours of the involved devices based on a ***client-server model***.

A **cluster is a collection of commands and attributes**, which define an interface to a specific functionality. The **device** that stores the attributes, by keeping an internal state, is the **server of the cluster** while the device that manipulates/writes the attributes is the **client of the cluster**. It's not necessary that a server have more computational power than end-devices.

If we think an application that switch the light, it will have an attribute `ON/OFF` and the client will be the switch that turn on/off the light while the server will be the light bulb itself. Cluster are grouped in **functional domains**:

- *General*: allows the access and control of any device, irrespective of their functional domain. Includes general basic functions, implemented by all devices.
- *Closures*: for shade controllers, door locks.
- *HVAC*: for pumps (*like fan, heating, etc*)
- *Lighting*: control lights
- *Measurement and sensing*: illuminance, presence, flow, humidity

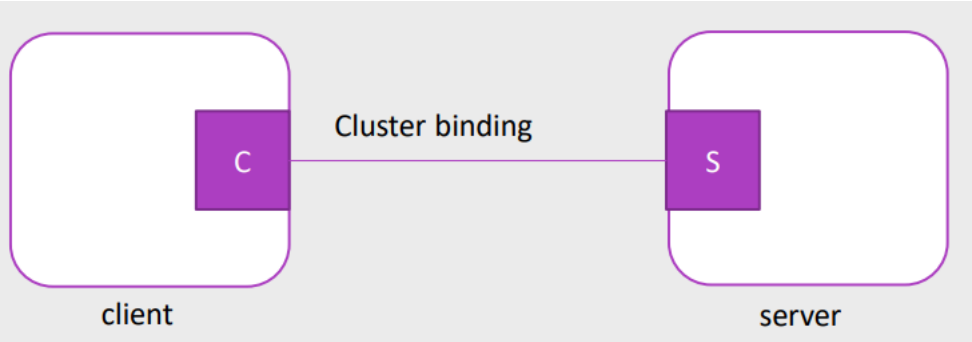

Figure 3.19: undefined undefined

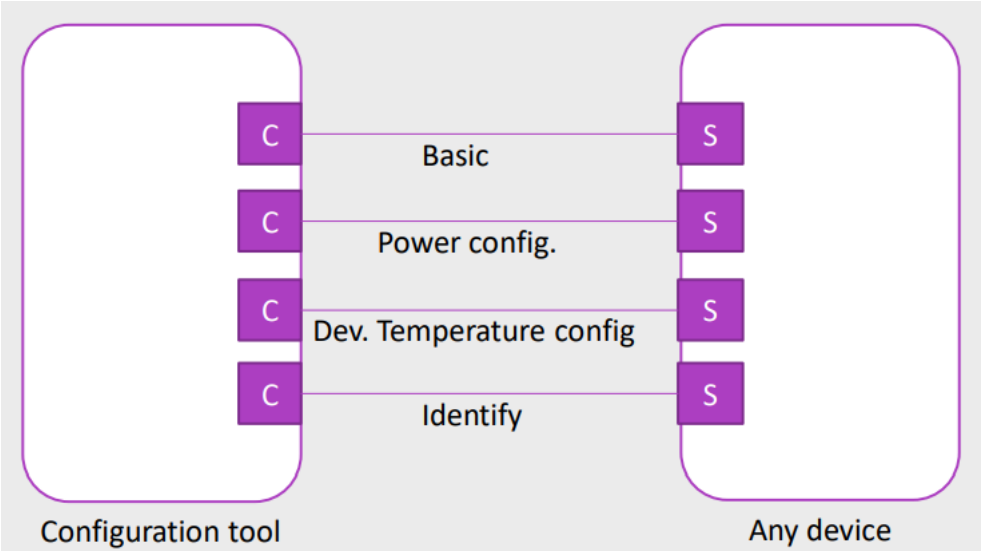

Figure 3.20: undefined undefined

- *Security and Safety*: security zone devices (*e.g. passive infrared*)
- *Protocol interfaces*: interconenct with other protocols Each functional domain, for each server, defines **commands** and **message structure**. The **commands** are messages (*with their own header and paylaod*) with a format specified by ZCL: the commands can be read/write attribute or also can be used for dynamic attribute reporting (*useful to periodic reading of the sensors*). The *types of command* are generally tree:

1. Read/write an attribute
2. Configure a report and read a reporting response: allows to request periodic attribute/configuration reading. Requests for report whenever an attribute/configuration changes. The client ask the server to report periodicly about the state of the server: the server will send periodically a report to the client, withoyt any further request. Also defines parameters of the reporting like *duration, period, minimum change, etc.*
3. Discover attributes: to discover the IDs and types of the attributes of the cluster that are supported by the server

The *schemes of use* 3.20 show a typical use of **device configuration and installation cluster:** the configuration phase allows to setting up some parameters by using several clusters. The `Basic` allows to retrieve software version, manufacturer ID, etc. The `PowerConfig` return information about battery and power configuration. The `Temperature Configuration` allows to set treshold, etc. Last, the `Identify` cluster allows to identify physically a device.

Following the schema, usually the installer configure the devices by using specific configuration tools that allows to connect to the devices, apply the specific settings and read their status. As an example, the `Identify` allows to recognize the device by enabling a blinking led.

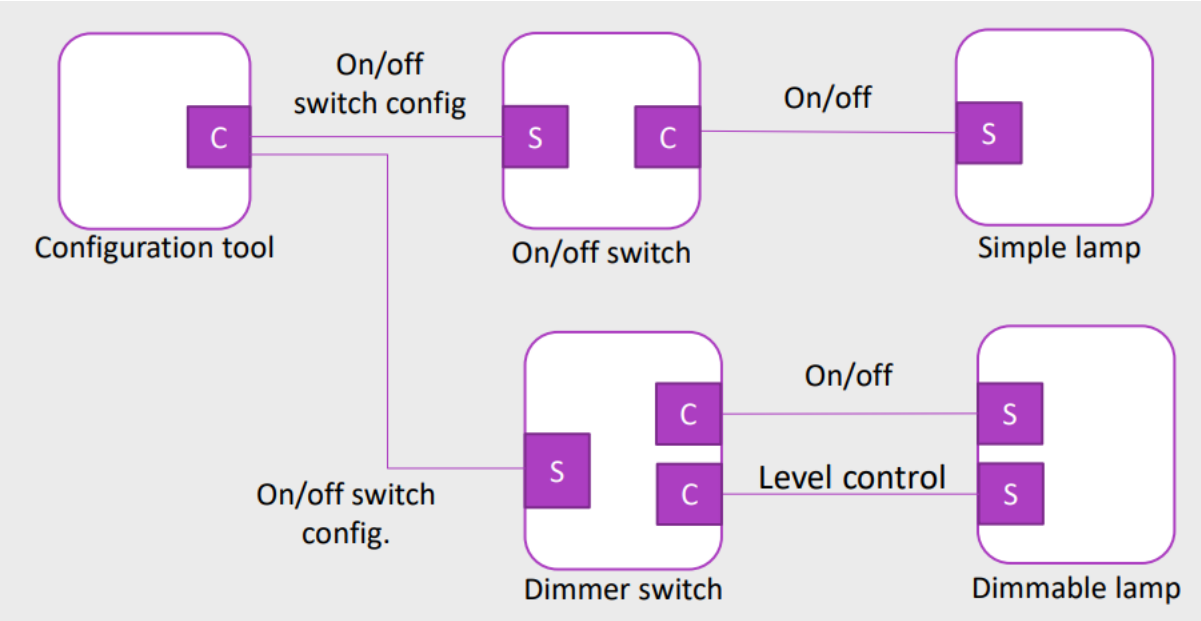

Figure 3.21: undefined undefined

Applied to our `ON/OFF` previous example, the relative scheme obtained is pictured in 3.21

In this case we want to configure a set of APO, we have one ON/OFF switch which is connected with a "*Simple Lamp*", we have to create the association between the lamp and the switch, in the same way we must also create the association between the "dimmer switch" and the "dimmable lamp".

This association is created by the **Configuration Tool**, working as a client sets the configuration of the On/Off switch device in such a way that when it wants to turn the lamp on or off it will find the corresponding device and make the request for turning on or off. The same goes for configuring the Dimmer Switch. The configuration tool must also configure the simple lamp. The configuration allows to setup the **binding table** to the **coordinator** by knowing the addresses of the devices and relative APO. This can be done by invoking device and service discovery, alongside physical discovery.
Typically what happens is that the installer has a tablet and it connects to the Zigbee Network, using an application you can access information about the network. When I run this configuration each device knows what it is and what it can do, a lamp for example knows it is a lamp but it is necessary to carry out a **commissioning procedure** so that the lamp is connected to an on/off switch which allows the ignition. It's necessary to execute the procedure of *commissioning* to simplify the deployment of complex smart system, like in a scenario with hundreds of devices to be deployed and configured.

Each devices has many of information assigned to it, contained in the `BASIC` cluster, like the ones listed in 3.22.

The `ZCLVersion` is mandatory to be sure that the ZCL is the right one implemented at application level, specially due to continue updating. Also the `PowerSource` is mandatory because provide information about power sources. Usually the attributes in `BASIC` cluster are in *read-only* access.

Among those information, highly relevant are the `PowerSource`, an 8-bit number that can have the values listed in 3.23.

The value in the last table indicates the **type of battery**, if there is an emergency power supply, etc. Those devices have also the attribute called **tempetature measurement cluster** that allows to kown the tempetature *inside* the device itself. The attribute to know this temperature are specified by `Temperature` cluster as listed in 3.24.

The cluster specified uses only generic commands to *read attributes, discover attributes, configure and report.* There are no specific commands to program a simple devices as a temperature measurement.For complex devices that also implement part of the `Temperature` cluster there is usually an accessory appendix that specify other commands.

ZCL is designed to enforce a hierarchical approach 3.25 to device functionality by supporting backward compatibility and interoperability: instead of design every possible ZCL cluster for specific *lamp*, it's incremental and allows compatibility by also let the possiblity to create custom features not standardized by ZigBee.

Identifier	Name	Type	Range	Access	Default	Mandatory / Optional
0x0000	*ZCLVersion*	Unsigned 8-bit integer	0x00 – 0xff	Read only	0x01	M
0x0001	*ApplicationVersion*	Unsigned 8-bit integer	0x00 – 0xff	Read only	0x00	O
0x0002	*StackVersion*	Unsigned 8-bit integer	0x00 – 0xff	Read only	0x00	O
0x0003	*HWVersion*	Unsigned 8-bit integer	0x00 – 0xff	Read only	0x00	O
0x0004	*ManufacturerName*	Character string	0 – 32 bytes	Read only	Empty string	O
0x0005	*ModelIdentifier*	Character string	0 – 32 bytes	Read only	Empty string	O
0x0006	*DateCode*	Character string	0 – 16 bytes	Read only	Empty string	O
0x0007	*PowerSource*	8-bit enumeration	0x00 – 0xff	Read only	0x00	M

Figure 3.22: undefined undefined

3.0.5 ZigBee Security Specification

Security services provided for ZigBee include methods for:

1. key establishment: create the keys without prior information
2. key transport: exchange keys previously setted/stored at configuration
3. frame protection: by using encryption
4. device management Security services are optional and have an overhead, minimum for low power devices, These services form the building blocks for implementing security policies within a ZigBee device. The underlying assumptions are that **level of security** depends on:

- the safekeeping of the symmetric keys: by avoiding stolen information about keys, also physically
- the protection mechanisms employed,
- the proper implementation of the cryptographic mechanisms and associated security policies involved. **Trust** in the security architecture comes from:
- trust in the ***secure initialization*** and installation of keying material and
- trust in the ***secure processing and storage*** of keying material.

Further assumptions are:

- **correct implementation of all security protocols**: the implementation must follow the standard, according to ZigBee standard and deployment operations
- correct random number generators
- secret keys do not become available outside the device in an unsecured way. The only exception to this is when a device joins the network.

Due to the *low-cost nature of ZigBee devices* we **cannot generally assume the availability of tamper resistant hardware**: it's possiible to some extent but it's not provided by the ZigBee standard (*can be possible by anti tamper hardware to a given point*). The level of protecntion is both at network level of physical level. Hence, physical access to a device may yield access to secret keying material and other privileged information, as well as access to the security software and hardware. Also, **different APO in the same device are not logically separated and lower layers are entirely accessible to the application layer**. Hence different APO in the same device must trust each other.

Attribute Value	Description
0x00	Unknown
0x01	Mains (single phase)
0x02	Mains (3 phase)
0x03	Battery
0x04	DC source
0x05	Emergency mains constantly powered
0x06	Emergency mains and transfer switch
0x07 – 0x7f	Reserved

Figure 3.23: undefined undefined

Identifier	Name	Type	Range	Access	Default	Mandatory / Optional
0x0000	*MeasuredValue*	Signed 16-bit integer	*MinMeasuredValue* to *MaxMeasuredValue*	Read only	0	M
0x0001	*MinMeasuredValue*	Signed 16-bit integer	0x954d – 0x7ffe	Read only	-	M
0x0002	*MaxMeasuredValue*	Signed 16-bit integer	0x954e – 0x7fff	Read only	-	M
0x0003	*Tolerance*	Unsigned 16-bit integer	0x0000 – 0x0800	Read only	-	O

Figure 3.24: undefined undefined

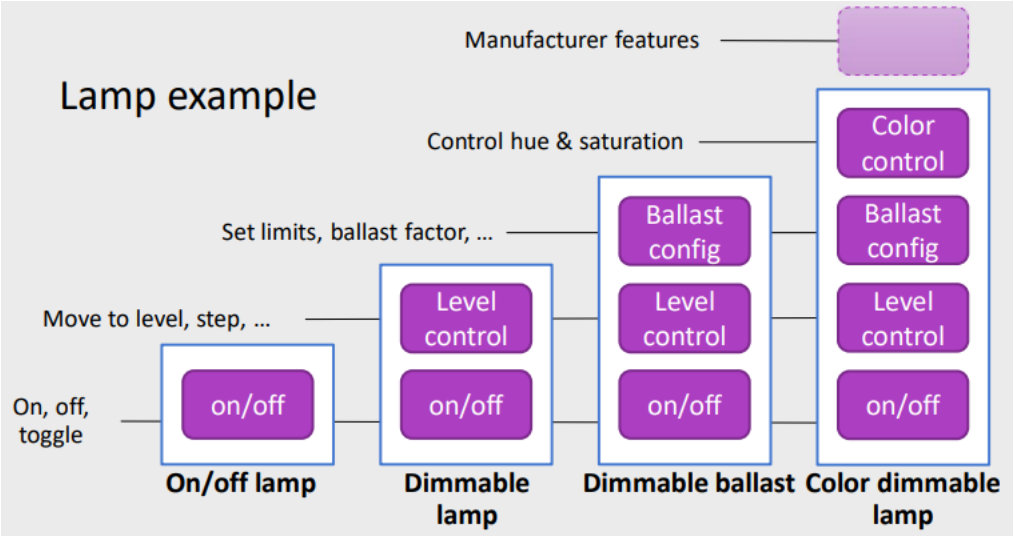

Figure 3.25: undefined undefined

Security Design Choices The security is granted by the **Trust Center** that it's a module in the *Application Layer*, tipycally allocated in the *ZDO* that distribute the security keys to the devices inside the same network. Two main keys are provided, at different level:

- **Single key per network** (*network level security*)
- **Single key per link** (*device-to-device level security*) The process flows: the devices inside the network uses a **Master key** to connect to the **Trust center**: this master key can be pre-assigned to the device (*write in ROM/Firmware*) or can be injected later in the device. The master key allows to establish a connection to request the **Network keys** or **Link keys**. Those three key allows ZigBee to provide a level of encryption corresponding to $AES - 128$ that it's enough for low power devices. The keys are provided by the APS. The low power and computing constraints of the devices implies that the encryption is **symmetric** and the decryption is made by the keys provided by the *application layer*. It aims to the protection of individual devices but not individual applciations in the same device: this allows the re-use of the same keying material among the different layers of the same device.

The layer that originates a frame is responsible for initially securing it: if a network command need protection this must be ensured at network level security. If protection from *theft of service* is required, network layer security must be used for all frames (*except the one when new device joins*). The reuse of key materials among different layers in a device and link-keys allows to have additional security from source to destination: an application can extends the security by adding other mechanism but it's entirely upon itself.

An *application profiles* should include policies to:

- Handle error conditions arising from securing and unsecuring packets: error conditions may indicate loss of synchronization of security material or may indicate ongoing attacks.
- Detect and handle loss of counter synchronization and counter overflow.
- Detect and handle loss of key synchronization.
- Expire and periodically update keys, if desired.

The keys The keys can be obtained by means of:

- Key transport: a mechanism for communcating a key from one device to another dervice or other device. It's a sensible phase because at that time the communication is unencypted: can be encrypted but with more overhead, depending the scenario context.
- Key establishment: a mechanism that involves the execution of a
- Pre-installation: at manufacturing time, install the key into the device or during the deploying phase

The **Network key** are *128-bit* shared by all devices: they are acquired either by key-transport or pre-installation and generally are of two types (***standard vs high security***). The **Link keys** are 128 bit, shared by all devices: they are acquired by key-transport, key-establishment or pre-installation. THe key establishment is based on pre-existing master key. Two types are considered (***Global vs Unique***). The **Master key** is a 128 bit key acquired either by key transport or pre-installation. Other keys are used to implement the secure transport of key material: they are derived from the link key with one-way functions.

The **Key Establishment** it's stated by the specification: it describe a variation of DIffie-Hellmann and it's called **SKKE (Symmetric-Key Key Establishment)**. Usually the key establishment involves two entities, an initiator device and a responder device. It evolves as following:

1. *trust provisioning based on trust information (typically the master key)*: the master key can be either pre-installed during manufacturing or it may be installed by a Trust Center (*for example, from the initiator, the responder, or a third-party device acting as a Trust Center*), or it may be based on user-entered data (*for example: PIN, password, or key*)
2. exchange of ephemeral data (a random number)
3. the use of this ephemeral data to derive the link key.

4. confirmation that the link key was computed correctly

The **Trust Center** is the device trusted by devices within a network: it's a funcionality implemented in the coordinator or a specified device. In a simple network can be the coordinator and it's usually used to obtain new **network/link key** by connecting to it and requesting new keys. All members of the network shall recognize exactly one Trust Center and there shall be only one in each secure network to provide a trust agreement between all the device in the same network.

The device that joins the network:

- obtain the keys from the trust center according to different protocols, depending on which keys they already have (or no key)
- in low-security applications the Trust Center give it the master key by using unsecured transport
- If the device already has the master key, it can establish a secure communication link with the Trust Center to transport the keys.

Chapter 4

IoT Design Aspects

IoT are special custom devices that need to manage the energy efficiency differently from standard market device. The devices are charaterized by:

- Low power
- Low cost
- Small
- Total autonomous

Each device is a full micro-system that embeds:

- Processor
- Memory
- Radio transceiver
- Battery powered or energy harvesting methods: depending on the specific scenario, both allows to avoid wired powered devices.

The energy efficiency based on battery-powered implies that the working life is limited in time: this implies the need to intervene on the field to change the battery. This operation can be costly, respect to the device cost itself. Some other issues are *adaptability to changing conditions* that arises for dynamic network management and programming, adapting its behavior as the environment change. This implies the neessity of dynamic programming to also preserve the interoperability. Another problem is that the devices comes in term of very low complexity and overhead, with reduced complexity of complexity execution and constrains on computation resources. Another issues concerns *multi-hop communication*: standard routing protocols like Distance Vector or similar relies on address table but due to constrained capacity both in term of storage and computational resoures, those model does not scale in IoT context multi-hop network.

This also introduce the problem of the *mobility* that need for dyamic routing protocols to scale efficiently, provided the underlying infrastructure.

Moore's Law

"The number of transistor that can be embedded in a chip grows exponentially (it doubles every two year)". As a comparison example:

- Intel 4004 in 1971: 2.3k transistors, 740 KHz and 4KB program memory
- Itel core i9 in 2017: 1.8G transistos, 4.4.GHz, 165W and 128 GB

The application of Moore's law to IoT is peculiar and subject to different interpretation,

1. The *performance doubles every two years* at the same cost
2. The *chip's size halves* every two year at the same cost
3. The size and the processing power remain the same but the cost halves every two year.

In IoT all those interpretation are true, based on the specifics scenario: there are application that require small sized sensors and or that have low power comsuption, can require higher processing capabilities. There is the need to balance the computational power to the costs, based on the specific application.

The Moore's law does not necessarily solve the problem in IoT design, at least in the near future.

4.0.1 Energy efficiency

It represent one of the critical aspects because devices are usually battery-powered, avoid wired powered devices and avoid maintenance cost. The energy efficiency is one of the main aspects in IoT design. The following diagram shows how the evolution of *processor, HD capacity and memory* grows faster than battery efficiency as pictured in 4.1

The improvements of the battery efficiency is not bound to the Moore's Law: this is also due to the physic limit. The consequences are that in we want to improve the battery efficiency, we need to improve the design aspects of an IoT system, improving the design of the components that heavily rely on battery.

In a laptop, most of the energy in spent on a screen, as shown in 4.2 image.

In a wireless sensor, the energy usage follows a different pattern than in laptop as pictured in 4.3: 20% of battery is used for sensing activities, the 40% is used for wireless and network interfaces activities and the remaining part, about 40%, is used by the processor and the chipset.

The percentage areas can vary based on the specific sensors. There is a trade-off between sensing and transmission: *"a general rule state that transmit 1 byte is equal to compute 1k byte"*. Usually the sensing activation frequency it's mandatory based on a given context: reducing it have a direct impact on the quality of sensing data. The opmitizable component is surely the processor area but, differently, the sensing area cannot be opmitized muchly because it both transmit and receive: the optimizable part is the one that concern the communication timeframe and behavior. A radio for **WIfi Network Interface** can be in three modes, each with different energy consumption:

1. **Sleep Mode**: consume very less and can get the radio operative in short time. The energy consumption is around 10mA.
2. **Listen Mode**: 180mA
3. **Receive Mode**: 200mA
4. **Transmit Mode**: 280 mA

For a sensor (*specifically the moto-clone*) the energy consumption is:

1. Sleep Mode: 0.0.16mW
2. Listen Mode: 12.35 mW
3. Receive Mode: 12.50mW
4. Transmit Mde: 0.1 power level, 19.2kbps: 12.36 mW
   1. 0.4 power level, 19.2kbps: 15.54 mW
   2. 0.7 power level, 19.2kbps: 17.76 mW

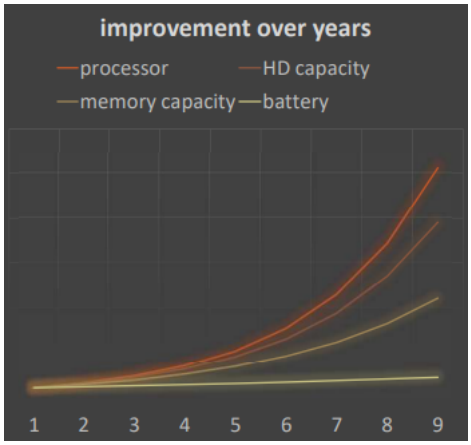

Figure 4.1: undefined undefined

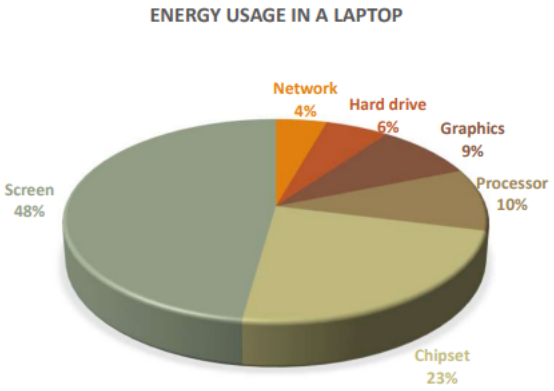

Figure 4.2: undefined undefined

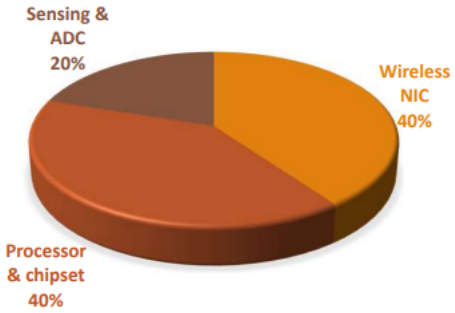

Figure 4.3: undefined undefined

The power level can change based on the distance that are needed to communicate, amplyfing the signal. In case of receive mode, it's also necessary to activate a decoder (from a given analogical signal) that require more energy. Those numbers show how it's not useful to put the sleep mode only, but remaining in a listen mode, letting the device comunicate. This had an impact on the MAC protocols for IoT which differs from the standard definition. So for sensors, as a rule of thumb, the approach is:

- transmit power is less than receive power
- listen power is more or less equal to receive power
- radio should be turned off as much as possible Switching on and off a component require energy: the decision to switch the status must not be reversed in a short timeframe.

The IoT device are not a general purpose device so it's not needed to be active all the time, providing the service: the key idea is to saving energy by **reducing the period of activity of a sensor**. This is called **duty cycle** and define the repetitivity of the activities of a sensors:

1. Sense: it's more efficient if it's repeated periodically and it's in a constant timeframe
2. Process and store
3. Transmit and receive A sensor alternates a period of activity and a period of inactivity (*defining a duty cycle*): during the inactivity the energy consumption is very low but the process, radio and I/O need to be freezed.

In general, the ***duty cycle*** of a system is defined as the fraction of one period in which the system is active. The radio activities can represent an exception to this, based on the specific activity that need to be carried out even during an inactivity period. The duty cycle is commonly expressed as a ratio or as a percentage. An example taken form Arduino is the pictured in 4.4: the `delay` instruction does not power off any component. The period of activity of this duty cycle is 400 ms, but the duty cycle itself does not guarantee to save energy consumption.

Instead, consider the following code in which we turn off the components:

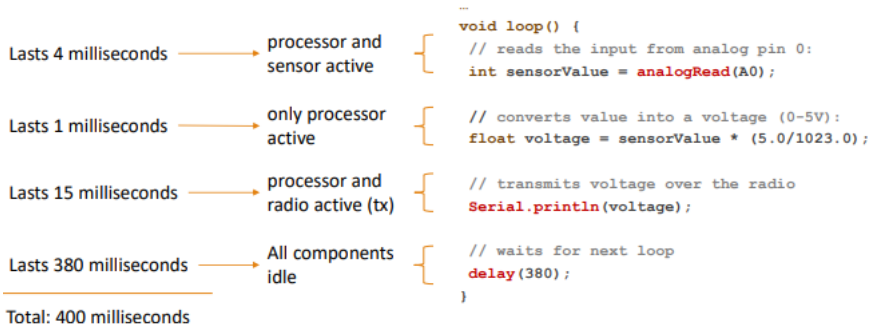

Figure 4.4: undefined undefined

```
void loop() {
// reads the input from analog pin 0:
turnOn(analogSensor);
int sensorValue = analogRead(A0);
turnOff(analogSensor);

// converts value into a voltage (0-5V):
float voltage = sensorValue * (5.0 / 1023.0);

// transmits voltage over the radio
turnOn(radioInterface);
Serial.println(voltage);
turnOff(radioInterface);

// waits for next loop
idle(380);
}
```

The `turnOn/turnOff` turn on/off a given components and the `idle` isntruction puts the processor in idle state with low power consumption for `y` ms. The `idle` instruction allows to do not power-off the processor but to obtain a duty cycle that by firing a call when the `y` ms are passed, the processor can return operative. The above code defines 3 different duty cycles, one for each component as is sketched in 4.5.

The duty cycles allows to define or compute the current absorbed by each component based on the general state of each component: the table in 4.6 define the specific value for each state of the considered component. As an example, the processor have two state (*full operation, sleep*) that consume different amount of current (*respectively 8 mA and 15 μ A*).

The battery gives a given amount of power but part of the energy is lost even without plugging it: it estimates to 3% per year on the total provided energy. This also have an impact on the charge of the

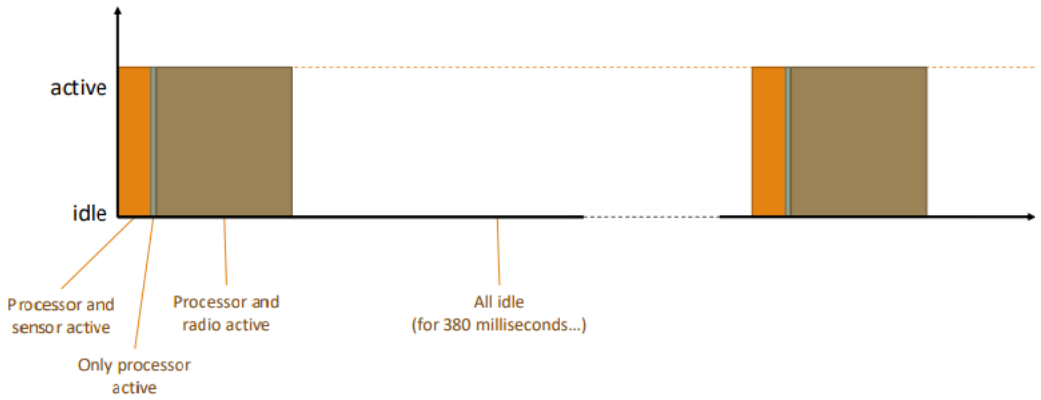

Figure 4.5: undefined undefined

	Value	units
Micro Processor (Atmega128L)		
current (full operation)	8	mA
current sleep	15	µA
Radio		
current in receive	19,7	mA
current xmit	17,4	mA
current sleep	20	µA
Logger (storage in the flash memory)		
write	15	mA
read	4	mA
sleep	2	µA
Sensor Board		
current (full operation)	5	mA
current sleep	5	µA
Battery Specifcations		
Capacity Loss/Yr	3	%

Figure 4.6: undefined undefined

battery that decrease overtime.

Example 2 The first model have a 100% DC so it's always fully active while the model 2 have a 5% DC as listed in 4.7, or, in a dirrent visualization as pictured in 4.8

	model 1: 100%DC	model 2: 5%DC	units
Micro Processor			
current (full operation)	100	5	%
current sleep	0	95	%
Radio			
current in receive	50	4	%
current xmit	50	1	%
current sleep	0	95	%
Logger			
write	1	1	%
read	2	2	%
sleep	97	97	%
Sensor Board			
current (full operation)	100	1	%
current sleep	0	99	%

Figure 4.7: undefined undefined

The sum of two different activity period form a duty cycle: in the figure 4.8 there is only indicated a single period.

4.0.2 Measuring energy

Energy and power are mesured in Joule J and Watt W, respectively $1J = 1W * sec$. In electromagnetism, $1W$ is the work performed when a current of $1A$ (Ampere) flows through an electrical potential difference of $1V$ volt, so: $1W = 1V * 1A$ Since we use direct current and the electrical potential difference is almost

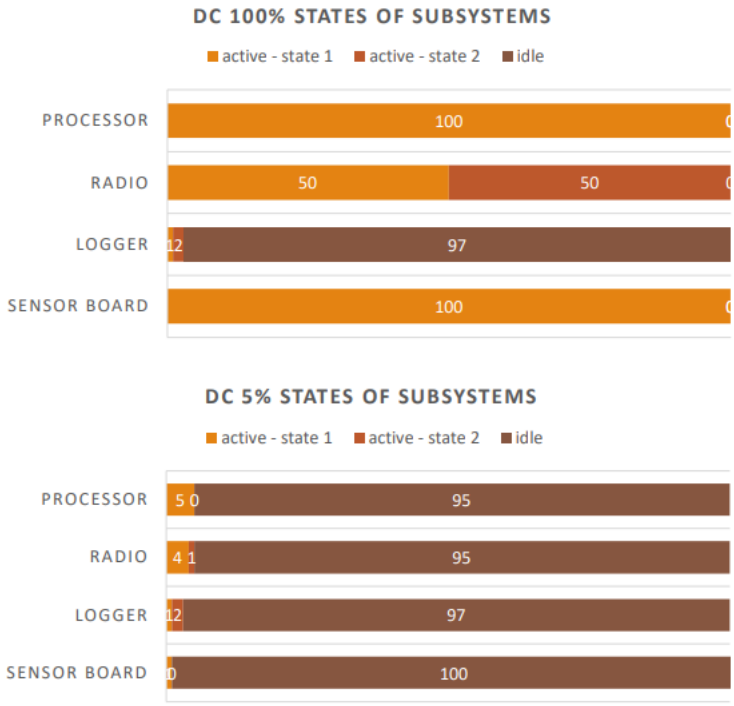

Figure 4.8: undefined undefined

constant, the power and the energy only depend on the current (*Ampere*). Hence we can express both the energy stored in a battery and the energy consumed in mAh (*milli-Ampere per hour*).

4.0.3 Example on computing consumption per duty cycle

Consider two different models in which, for each component, the Duty Cycle *(DC)* follows the percentage presented in 4.9:

Note that the first model have a Duty Cycle of 100% so it's an always up system while the second model present a Duty Cycle of 5%.

To compute the energy total cost we must first compute the energy cost of each component, considering its duty cycle period.

The **energy cost of a microprocessor** E_μ ***per cycle*** can be expressed as:

$$E_\mu = C_\mu^{full} * dc_\mu + C_\mu^{idle} * (1 - dc_\mu)$$

where:

- C_μ^{full} full energy cost microprocessor
- C_μ^{idle} idle energy cost microprocessor
- dc_μ

The C_μ^{idle} cannot be dropped because it's multiplied with a factor that is not neglectible $(1 - dc_\mu)$. The **energy cost radio** E_ρ ***per cycle*** can be expressed as:

$$E_\rho = C_\rho^T * dc_\rho^T + C_\rho^R * dc_\rho^R + C_\rho^{idle} * (1 - dc_\rho^T - dc_\rho^R)$$

where:

- C_ρ^T radio transmission energy cost
- C_ρ^Rradio receival energy cost
- C_ρ^{idle} idle energy cost
- dc_ρ^T percentage transmit duty cycle radio

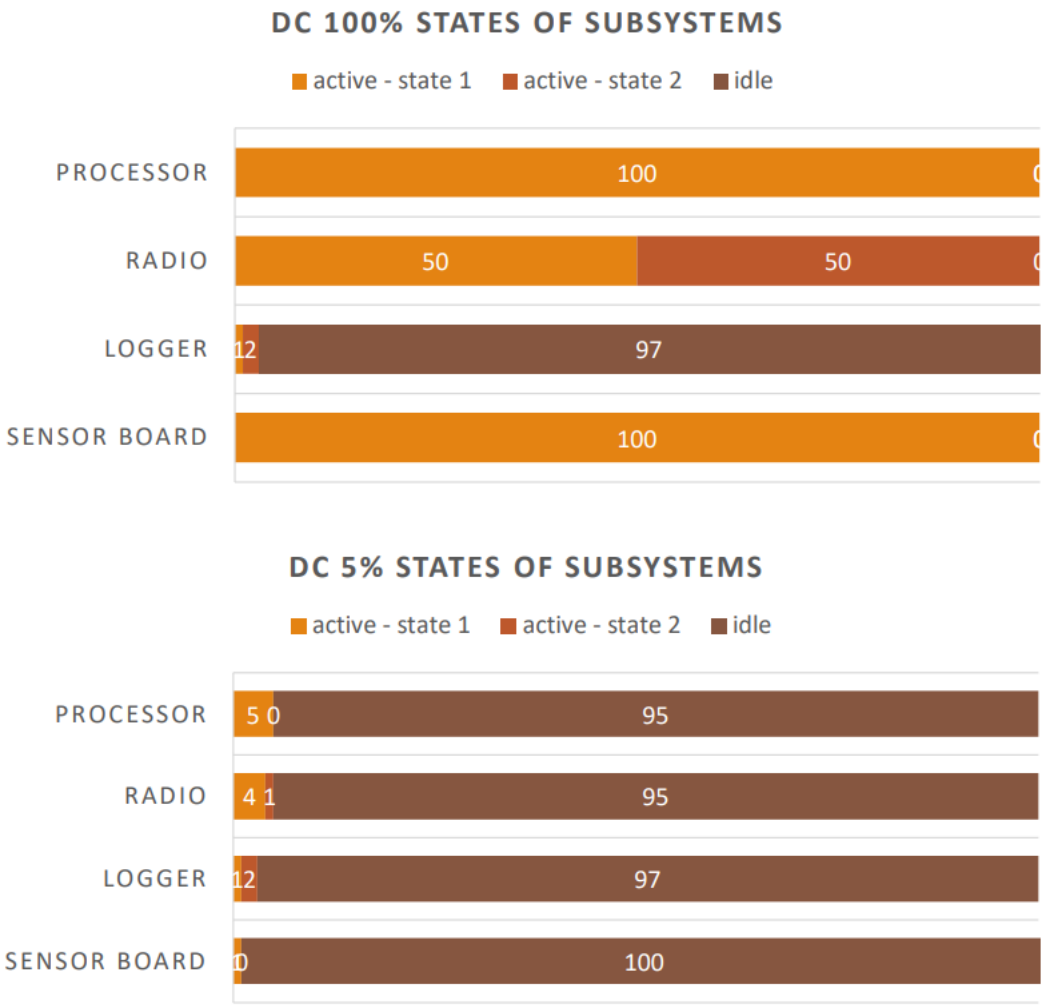

Figure 4.9: undefined undefined

- dc_ρ^R percentage receive duty cycle radio

The **energy cost logger E_λ and sensor board E_σ** can be calculated as before, obtaining the **total energy cost per duty cycle** of: $E = E_\mu + E_\rho + E_\lambda + E_\sigma$ .The **lifetime**, expressed in number of duty cycles, is:

$$Lifetime = \frac{B_0 - L}{E}$$

where B_0 is the initial battery charge and L is the battery charge lost during the lifetime due to **battery leaks** and E is the total cost of energy. Because L depends on lifetime and it's based on charge/loss cycle ϵ (*estimated to 3% yearly*), we can obtain a recurrence equation: $B_n = B_{n-1} * (1-\epsilon) - E$ where B_n is the battery charge at cycle n. By solving the recurrence equation we obtain that:

$$B_n = B_0 * (1-\epsilon)^{n-1} + \frac{E((1-\epsilon)^n)-1}{\epsilon}$$

The device lifetime is given by n, when $B_n = 0$.
Regarding the two models already cited, we can plot both the battery life (*expressed in months*) and the battery capacity (*expressed in milli-Ampere per hour*) as pictured in 4.10.

The previous graph (4.10) show how the second model that have 5% of DC have a longer battery life than the first-one, thanks to longer duration of idle state of its components. The graph, which in expressed in logarithmic scale, show how enlarging the battery capacity the correspondent battery life for the first model is limited to a couple of days while for the second model the battery life can last up to some months.

The second graph (4.11) show that given two devices, increase the DC corresponds to a decrease of the lifetime battery. To obtain an efficient device, it's necessary to work on the DC to minimize it and obtain a longer battery life.

To gain *energy efficiency* the solution is to reduce the *Duty Cycles (DC)*, however ***turning off the processor is a local decision*** so the node scheduler knows that are the activires and when the processor should run. Also, turning off the radio is a **global decision** that implies no communication, no reception of incoming messages/commands and the impossibility to act as a router in multihop network.

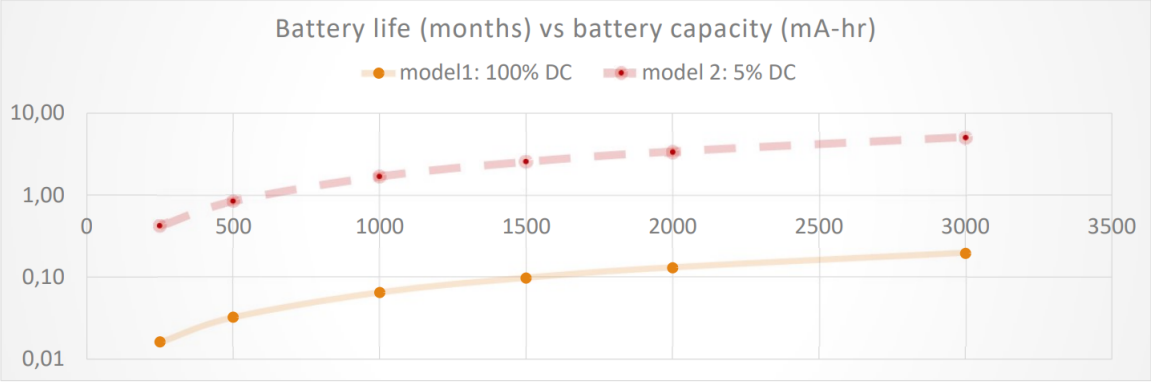

Figure 4.10: undefined undefined

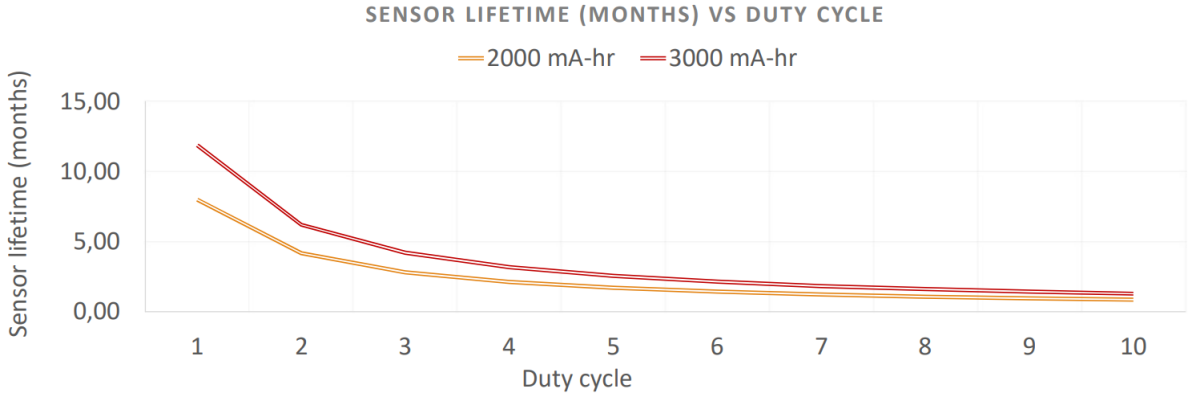

Figure 4.11: undefined undefined

At **MAC Layer**, MAC Protocols arbitrate the access to the shared communcation channel: in IoT also implements strategies for energy efficiency like **synchronize the devices** and **turn off the radio when it's not needed** (*so excluding the device from the network*).

4.0.4 Exercise

Consider this program and the table of energy consumption in the different states. Compute:

- the energy consumption of the device per single hour
- the expected lifetime of the device

```
...
void loop() {
 turnOn(analogSensor);
 int sensorValue = analogRead(A0);
 turnOff(analogSensor);

 float voltage = sensorValue*
         (5.0 / 1023.0);

 turnOn(radioInterface);
 Serial.println(voltage);
 turnOff(radioInterface);

 idle(380);
}
```

4 milliseconds
1 milliseconds
15 milliseconds
380 milliseconds

	value	units
Micro Processor (Atmega128L)		
current (full operation)	8	mA
current sleep	15	µA
Radio		
current xmit	1	mA
current sleep	20	µA
Sensor Board		
current (full operation)	5	mA
current sleep	5	µA
Battery Specifcations		
Capacity	2000	mAh

Figure 4.12: undefined undefined

energy consumption per hour:

- The processor has a duty cycle of _____
- The radio has a duty cycle of _____
- The sensor has a duty cycle of _____

Energy consumption in one hour: _____

Lifetime of the device: _____

	idle	Active
Processor		
Radio		
Sensor		
Total:		

	value	units
Micro Processor (Atmega128L)		
current (full operation)	8	mA
current sleep	15	µA
Radio		
current xmit	1	mA
current sleep	20	µA
Sensor Board		
current (full operation)	5	mA
current sleep	5	µA
Battery Specifcations		
Capacity	2000	mAh

Figure 4.13: undefined undefined

Answer

1. Compute the energy comsuption of the device per single hour todo
2. Compute the expected lifetime of the device todo

Exercise 2

Consider the sensor specification in the table pictured in 4.14
The device measures the hearth-rate (HR) of a person:

- Samples a photo-diode on the wrist at 20 Hz
  - sampling the sensor takes 0.5 ms
  - it requires both the processor and the sensor active
- HR is computed every 2 s (based on 40 samples)

	value	units
Micro Processor (Atmega128L)		
current (full operation)	8	mA
current sleep	15	µA
Radio		
current xmit	20	mA
current sleep	20	µA
Sensor Board		
current (full operation)	5	mA
current sleep	5	µA
Battery Specifications		
Capacity	2000	mAh

Figure 4.14: Exercise 2 - Parameters specification

- Transmit (from time to time... see below) a data packet to the server:
  - The average time required to transit is 2 ms
  - Requires both processor and radio active Compute the energy consumption and the lifetime of the device if it sends all the samples to a server:
- Stores 5 consecutive samples from the photodiode
- Transmits the stored 5 samples to the server
- The server computes HR (hence the device does not compute HR) Disregard battery leaks.

Solution 2

Duty cycle of sampling: DC of processor + sensors: 0.5 milliseconds (sampling time) / 0.05 seconds = (sampling period)= 0.01

Duty cycle of transmitting: DC of radio + processor: 2 milliseconds (transmit time) / 0,25 seconds (transmission period) = 0.008

- Sensor power consumption: $E_\mu = 5mAh * 0,01 + 5uAh * 0,99 = 0,05 + 0,005mAh = 0,055mAh$
- Processor power consumption: $E_\rho = 0,018 * 8mAh + 0,982 * 15uAh = 0,144 + 0,0147mAh = 0,1587mAh$
- Radio power compsution: $E_\lambda = 0,008*20mA+0,99992*20\mu A = 0.16mA+0.0198mA = 0.1799mAh$

$$E = E_\mu + E_\rho + E_\lambda$$
Battery specification $(in\ table)$ $B_0 = 2000mAh$
Power consumption: $2000mAh/0,3935mAh$ $(\frac{B_0}{E})$
obtaining **Lifetime**: $5082h$

Exercise 3

Consider the sensor specification in the table pictured in 4.15.
The device measures the hearth-rate (HR) of a person:

- Samples a photodiode on the wrist at 20 Hz
  - sampling the sensor takes 0.5 ms
  - it requires both the processor and the sensor active
- HR is computed every 2 s (based on 40 samples)
  - Computing HR in the device takes 5 ms

	value	units
Micro Processor (Atmega128L)		
current (full operation)	8	mA
current sleep	15	µA
Radio		
current xmit	20	mA
current sleep	20	µA
Sensor Board		
current (full operation)	5	mA
current sleep	5	µA
Battery Specifications		
Capacity	2000	mAh

Figure 4.15: Exercise 3 - Parameters specification

- Transmit a data packet to the server:
  - The average time required to transmit is 2 ms
  - Requires both processor and radio active Compute the energy consumption and the lifetime of the device if it computes HR itself:
- Transmits every 5 values of HR computed (1 packet every 10 seconds)

Disregard battery leaks.

Solution 3

- Duty cycle of sampling: $0,5ms$ (sampling time) / $0,05s$ (sampling period)= $0,01$
- Duty cycle of processing: 5 milliseconds / 2 seconds = 0,0025
- Duty cycle of transmitting: 2 milliseconds (transmit time) / 10 seconds (transmission period) = 0,0002
- Power consumption of sensor (in 1h):
  - $E_\mu = 5mAh * 0,01 + 5uAh * 0,99 = 0,05 + 0,005mAh = 0,055mAh$
- Power consumption of processor (1h):
  - $E_\rho = 8mAh * 0,0127 + 15uAh * 0,9873 = 0,1016 + 0,0148mAh = 0,1164mAh$
- Power consumtpion of radio (1h):
  - $E_\theta = 0,0002 * 20mAh + 0,9998 * 20uAh = 0,004 + 0,02mAh = 0,024mAh$

$$E = E_\mu + E_\rho + E_\theta = 0.1954mAh$$
Battery specification $(in\ table)$ $B_0 = 2000mAh$
Power consumption: $2000mAh/0,1954mAh$ $(\frac{B_0}{E})$
Lifetime: $2000mAh/0,1954mAh = 10.235h$

4.0.5 Exercise 4

Consider a Mote-class sensor with the parameters pictured in table 4.16.

Assume that the device performs a sensing task with the following parameters:

- The sensor board is activated with a rate of 0,1 Hz to perform the sampling; this operation takes 0.5 milliseconds. At the end the sensor board is put in sleep mode. During each sensing operation the processor is always active.

	value	units
Micro Processor (Atmega128L)		
current (full operation)	8	mA
current sleep	0,015	mA
Radio		
current xmit	1	mA
current sleep	0,02	mA
Sensor Board		
current (full operation)	5	mA
current sleep	0,005	mA
Battery Specifications		
Capacity	2000	mAh

Figure 4.16: Exercise 4 - Parameters specification

- After each sampling the processor performs a computation that takes 2 milliseconds.
- Then the processor activates the radio and transmits the data. The transmission takes 1 millisecond and, during it, the processor is active. At the end the radio and the processor are both set in sleep mode. Compute the duty cycle of each component (*sensor board, radio and processor*), and the lifetime of the device (*assuming that the sensor stops working when its battery charge becomes 0*).

Solution 4

- Sampling takes $0.5ms$ with a rate of $0,1Hz$
- Processing: $2ms$
- Transmitting: 1 ms
- **Duty cycle of sampling (processor and sensors)**: $0.5ms/10s = 0,00005$
- **Duty cycle of processing (only processor)**: $2ms/10s$ $(10000ms) = 0.0002$
- **Duty cycle of transmissions (radioprocessor)**: $1ms/10s = 0.0001$
- Power consumption of sensor (in 1h):
  - $E_\mu = 5mA * 0,00005 + 0,005mA * (1 - 0,00005) = 2.5 * 10^{-4}mA + 0.05mA = 0.00525mA$
- Power consumption of processor (1h):
  - $E_\rho = 8mA * 0.00035 + 0,015mA * (1 - 0.00035) = 0.0028mA + 0,015mA = 0,0178mAh$
- Power consumtpion of radio (1h):
  - $E_\theta = 0,0001 * 1mA + (1 - 0,0001) * 0,02mA = 100nA + 20\mu A = 0.0201mA$
    $E = E_\mu + E_\rho + E_\theta = 0.04315mAh$ Battery specification (*in table*) $B_0 = 2000mAh$ Power consumption: $2000mAh/0.04315mAh$ $(\frac{B_0}{E})$
- **Lifetime**: $2000mAh/0,04315mAh = 46.349h$

Chapter 5

Energy Harvesting in IoT

Chapter 6

Arduino Embedded Programming & Use Cases

Chapter 7

Wireless Networks

7.0.1 Wireless networks

Wireless allows to replace cables in communications in cyber-physical systems that can embed computers in physical objects. Wireless networks are networks of **hosts** connected by **wireless links**: we refer to host as end-system devices that run applicaiton (battery-powered, small or mobile). There are two modes of operations:

- **Infrastrucuture**: e.g. by providing an Access Point that can provide a link to a cabled infrastracture and offer various services
- **Ad hoc networking**: there is no centra coordinations so they must somehow coordinate themself and manage properly shared resources

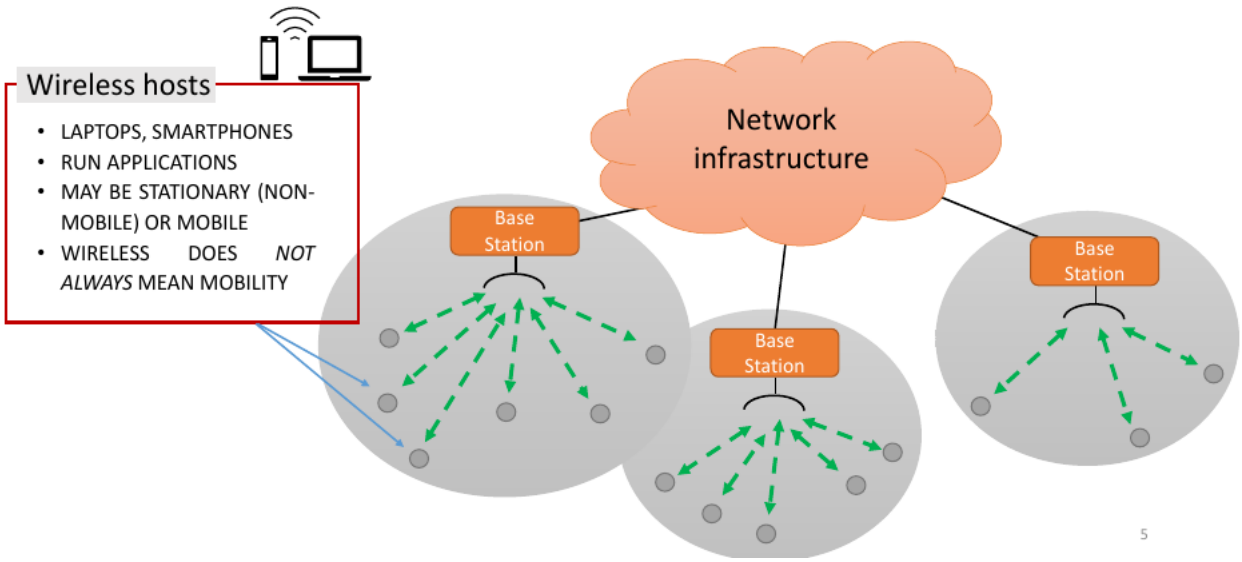

Figure 7.1: undefined undefined

The network infrastructure can be a mix of cabled and wireless infrastructure. The devices connected to the base station can be both mobile or not; the same can happen with the type of links that can be cabled or wireless, depending on the scenario. The **base station** have a revelent role in structured network: it's responsible to manage the resources of the *physical medium*. They usually are represented by cell towers or Access Point. The **wireless links** allows to connect mobile to base station and can be used as backbone links. In **infrastructure mode** base station connectmobiles into wired networks. Mobile can changes base station providing connection into wired network.

Charateristic of wireless links The category `802.11` refers to the Wifi family of protocols.

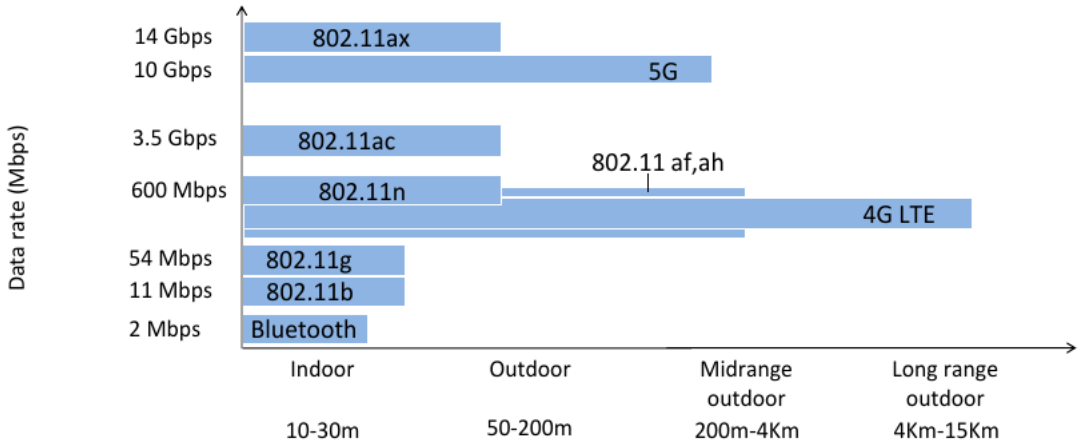

Figure 7.2: undefined undefined

7.0.2 Wireless Network taxonomy

We distingish the network type based on two category, as shown in the table 7.3.

An example of *infrastructure, single hop* is the mobile connection or throught the WiFi. The schema behing *no infrastructure, multiple hosts* is used when different devices needs to exchange information and makes routing decisions among them, without relying on a fixed infrastructure.

7.0.3 Charateristics of wireless communications

There are foundamental differences between wired and wireless links:

- **Decreased signal length**: radio signals attenuates as it propagates trhought matter causing **path loss**
- **Interference**: wireless frequency can be shared by other wireless devices by specification but engines or appliances may interfere as well.
- **Multipath propagation**: the reflection of the wave on context object can be distorted and change the origianl signal that it's necessary to rebuild

Communication Environment metrics

Some important metrics in **communication environment** are:

- **SNR - Signal to Noise Ratio**: measured at the received. The SNR *bit error rate* (**BER**) is the probability that a transmitted bit is received in error at the receiver. Based on different modulation technique we can achieve a lower or higher data rates, with different BER.

	Single hop	Multiple hops
infrastructure (e.g., APs)	host connects to base station (WiFi, WiMAX, cellular 3G,4G,5G) which connects to larger Internet	host may have to relay through several wireless nodes to connect to larger Internet: MESH networks
no infrastructure	no base station, not necessarily connection to larger Internet (e.g. Bluetooth)	no base station, no connection to larger Internet. May have to relay on other nodes to reach a given wireless node (ZigBee, ad hoc, VANET)

Figure 7.3: undefined undefined

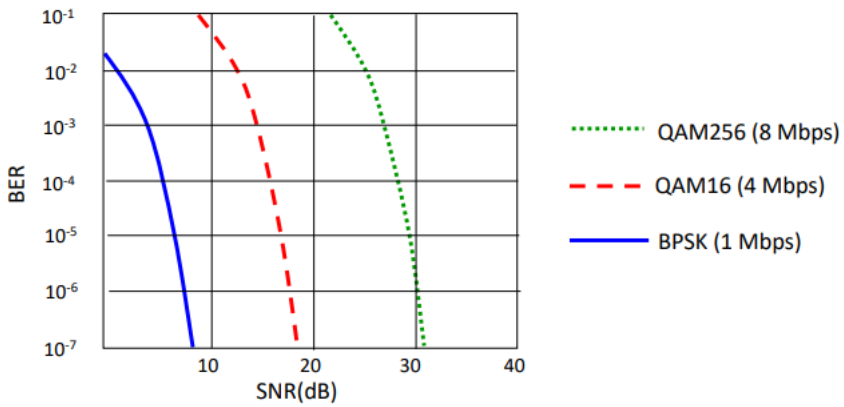

Figure 7.4: undefined undefined

-
- *BPSK* have a lower data rate and if we increase the SNR by increasing the *energy of the signal*, we lower the BER.
- For QAM16: you need more energy to get the same BER than BPSK at a certain level, as you send more information.

Obstacles

Signal attenuation or obstacles limit transmission ranges: the situation sensed by a node, given that there may be a wall, which prevents two devices to detect each other, while they can talk with a central host. According to how those host are positioned in distance:

- `A` has large signal strenght to `B`, but when we arrive at `B` the energy of the signal decrease and `B` is able to detect it.
- With higher distance `A` is not able to detect `C` signal. Also since `B` may receive two different signals at the same time, it may be difficult for it to reconstruct a signal. With the distance the amplitude of the signal decrease but despite that, the signal from C to B is received, the same from A to C but the intersection of receiving two signals from two different sources to B can be problematic.

The intended scenario is pictured in 7.5

Some of wireless network challenges are that nodes have **limited knowledge** regarding a terminal that cannot hear all the others or **hidden/exposed terminal** problems (*see later*). Another main problem is **Mobility or Failure of terminals** in which they move in the range of different base station. **Limited terminals** regards the battery life, memory and processing. In 7.6 is pictured the protocol stack that we'll describe.

Let's describe the **MAC layer** for wired network. We start from simple assumption like:

1. a **single channel** for all communications
2. all stations can transmit on it and receive,

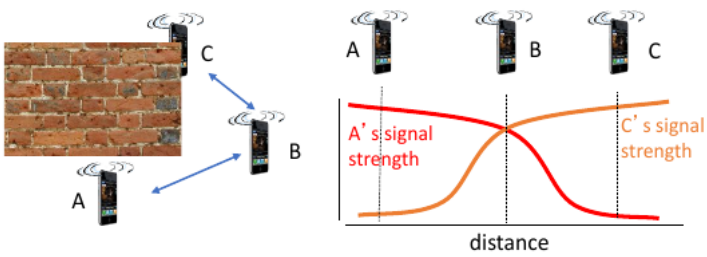

Figure 7.5: undefined undefined

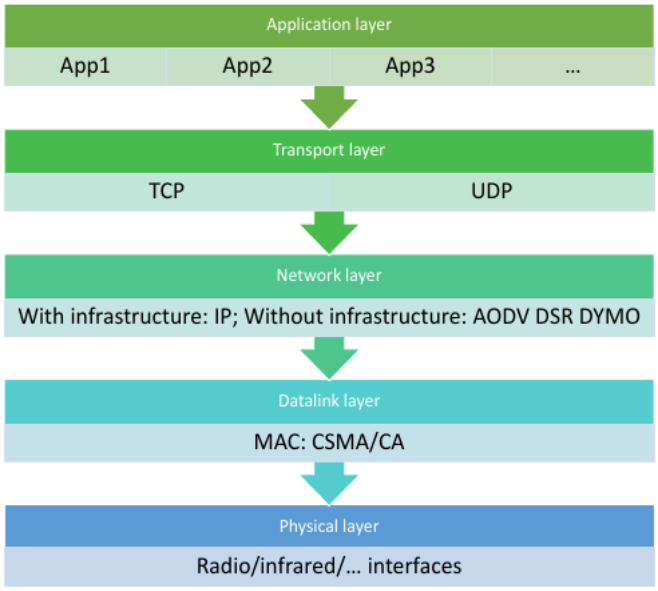

Figure 7.6: undefined undefined

3. if frames are sent simultaneously on the channel the resulting signal is *garbled* and a **collision** is generated, generally all stations can detect collisions. Different protocol are at this level, like: *ALOHA, slotted ALOHA, CSMA, etc.*

CMSA/CD - Carrier Sense Multiple Accesses with Collision Detection

The basic idea is that when a station has a frame to send, it first listens to the channel to see if anyone else is transmitting: if the channel is **busy**, the station waits until it becomes **idle**. When a channel is *idle*, the station transmits the frame: if a *collision* occurs the station waits a random amount of time and repeat procedures.

In the scenario shown in 7.7, the signal is transmitted at time t_0 because the channel was *idle* but at time t_1 another signal (*the red one*) is transmitted by generating a collision on the channel. So in CSMA/CD a station **abort its transmission as soon it detects a collision**: if two stations sense the channel idle simultaneously and start transmitting, they quickly abort the frame as soon collision is detected. It's widely used on LAN In MAC sub-layer like *IEEE 802.3 Ethernet*.

The CSMA/CD behaviour can be descriped by defining:

- `T` is the *time required to reach the fartest station*

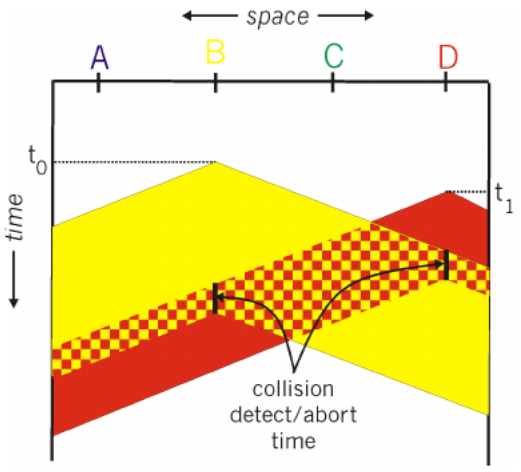

Figure 7.7: undefined undefined

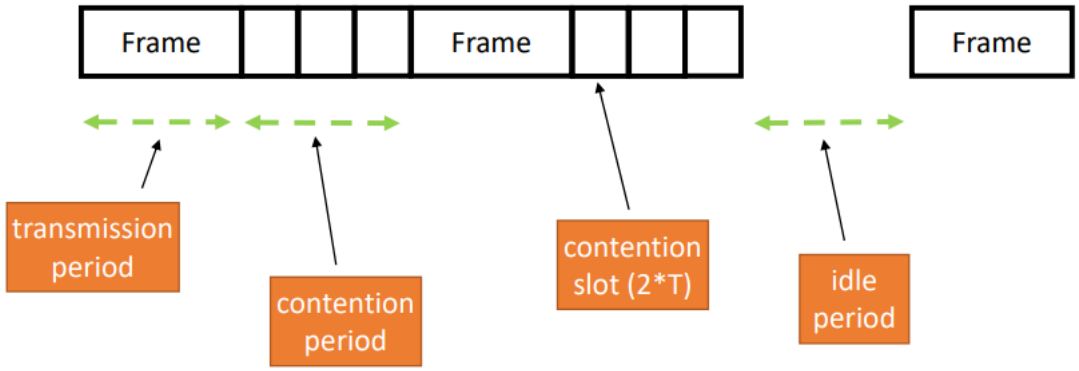

Figure 7.8: undefined undefined

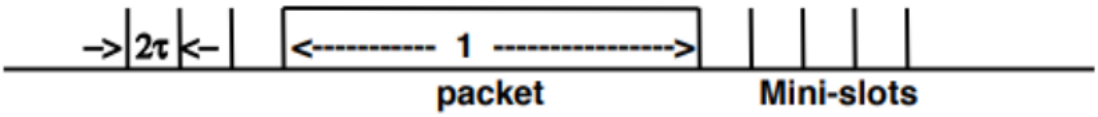

Figure 7.9: undefined undefined

- It takes a minimum of RTT time (`2*T`) to detect a collision

So we can consider a **slotted system** 7.9 with *mini-slots* of duration 2τ, so if a node starts transmission at the beginning of a mini-slot, by the end on the slot either two things can happen:

1. No collision occurred and the rest of the transmission will be uninterrupted (because now the channel is occupied and this condition is detectable by other nodes)
2. A collision occured but by the end of the mini-slot the channel would be idle again

So a collision at most affect *one* mini-slot.

Binary Exponential Backoff The algorithm used by CMSA/CD to retry retransmission is called **Binary Exponential Backoff** and defines how much time the station must wait before trying transmitting again.

The time after a collision is divided in **contention slots**: the lenght of a slot is equal to the *worst case round propagation time* (*so if T is the time to reach the most distant station, the length will be $2T$*). After the first collision each station waits 0 or 1 slot before trying again: so after collision i the nodes chooses x at random in the interval $[0, 2^i - 1]$ so that it skips x slots of time before retrying the trasmission. Number at hands, after:

- 10 collisions, interval is frozen at $[0, 1023]$
- 16 collisions, failure is reported to upper levels (*retrasmission is aborted*).

Why does this approach not work with wireless network? The algorithm described is suitable in **wired networks** but not in **wireless scenarios**. The main problems are:

1. The transmitted sides is sensing if there is another sending a signal in the channel. In wireless is not possible to both transmit and receive (*at least with 1 antenna*), the problem is that you transmit your signal with high energy, the one you receive has low energy, so it happens that you do not recognized the other. With wireless transreceiver you cannot detect collision.
2. If a node is out of range, it may not note a collision, this problem is called **hidden terminal problem**
3. Exposed terminal problem:

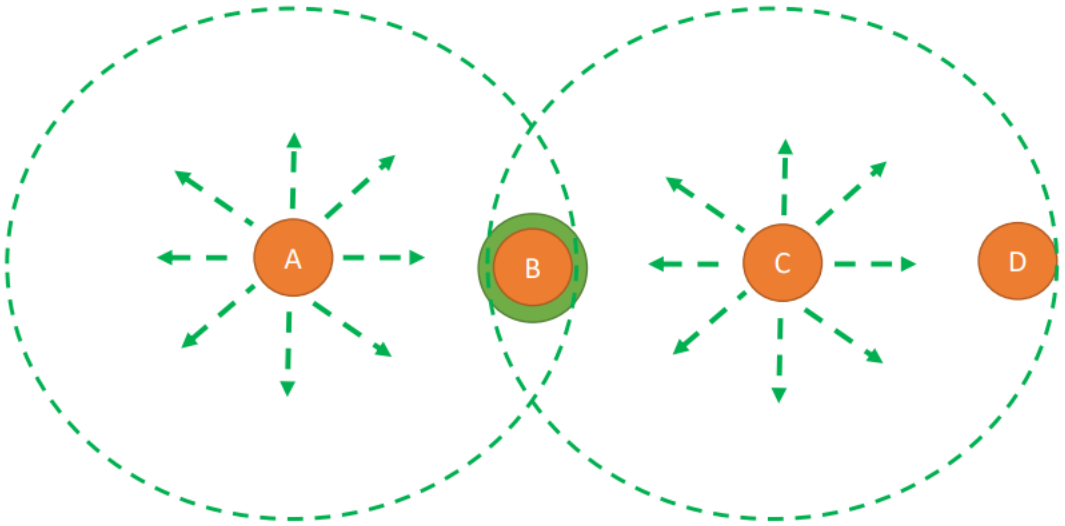

Figure 7.10: undefined undefined

7.0.4 Hidden terminal problem

The problem is sketched in 7.10.

The node A can receive and transmit to B because B it's in his radio range. The same happen to C: can transmit and receive to/from B and D because they are in C's radio range. The problem arise when A is sending to B and C senses the medium: *it will not hear A because its out of C's range* so if C start to transmit to anybody (*either B or D*) this *generate a collision at B*.

- **Hidden terminal problem**: C is not able to detect a potential competitor because it is out of range: a collision happens at B (*the receiver*). For the same reason A does not detect the collision so **C is hidden** with respect to the communication from A to B. Usually this problem arises when *two or more stations which are out of range of each other transmit simultaneously to a common recipient.*

7.0.5 Exposed terminal problem

The **Exposed terminal problem** is sketched in figure 7.11.

The node B is transmitting to A: the node C wants to transmit to D so C senses the medium and because detect the B $\rightarrow$ A transmittion it concludes that cannot trasmit to D. But, **the two transmission can actually happen in parallel**.

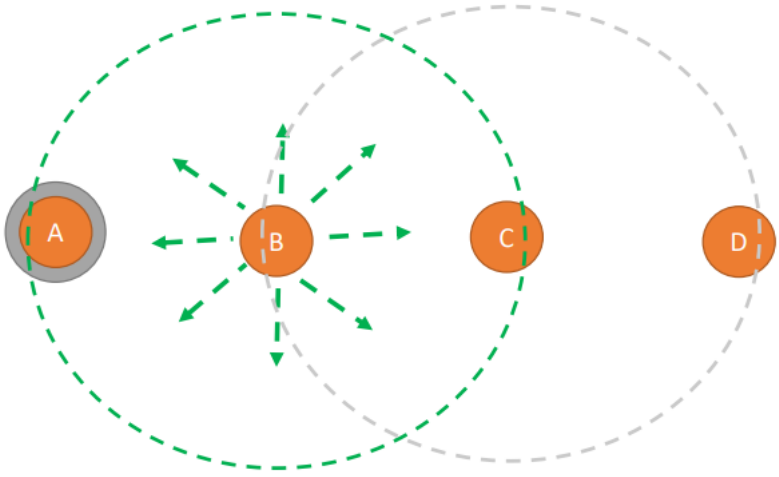

Figure 7.11: undefined undefined

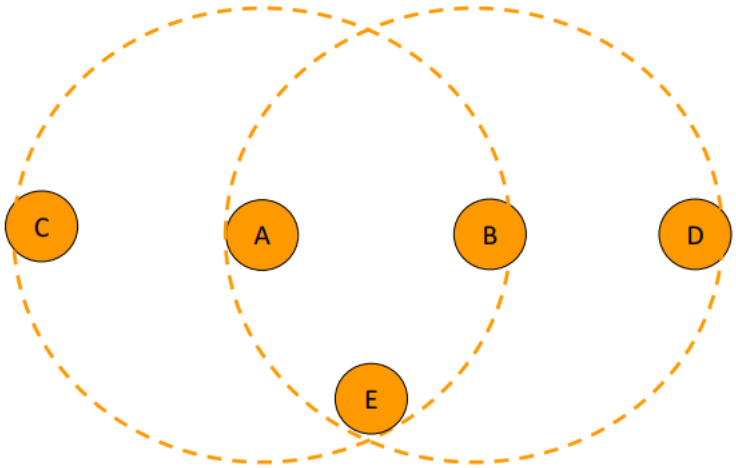

Figure 7.12: undefined undefined

- **Exposed terminal problem**: C hears a transmission. C does not send to D despite its transmission would be Ok. So **C is exposed** with respect to the communication from B to A. Usually this problem arises when *a transmitting station is prevented from sending frames due to interference with another trasmitting station.*

The solution on wireless network is represented by **MACA Protocol**.

7.0.6 MACA - Multiple Access with Collision Avoidance

The basic udea is to stimulate the receiver into transmitting a short frame first and then transmit a long data frame. So stations hearing the short frame **refrain from transmitting** during the trasmission of the subsequent data frame.

The scenario is pictured in 7.12 and it's here described:

- C is **within** range of A and **out** of range of B and D
- D is **within** range of B and **out** of range of A and C
- E is **within** range of both A and B

The protocol flows:

- **Step 1**: A wants to transmit to B so send a **RTS - Request To Send** packet to B: *the short frame* sended include also the length of the data frame that will eventually follow.
-
- **Step 2**: B, C and E receive the **RTS** from A (*because they're in A's radio range*). If B wants to receive the message, it replies with a **Clear To Send - CTS** that is a short frame with data length copied from RTS sended by A.
- **Step 3**: the **CTS** sended by B is received by A, D, E
- **Step 4**: upon reception of the **CTS** frame by B, A (*whose the only interested in receiving*) start to transmit the data frame

Problems:

1. **C hears the RTS from A but not the CTS from B** so it is free to transmit (*may be interested in transmitting).* So **C is exposed**. Refer the figure 7.17.
2. **D hears CTS from B nut not RTS from A** so should stay silent until data framr transmission completes. So **D is hidden**. Refer the figure 7.18.

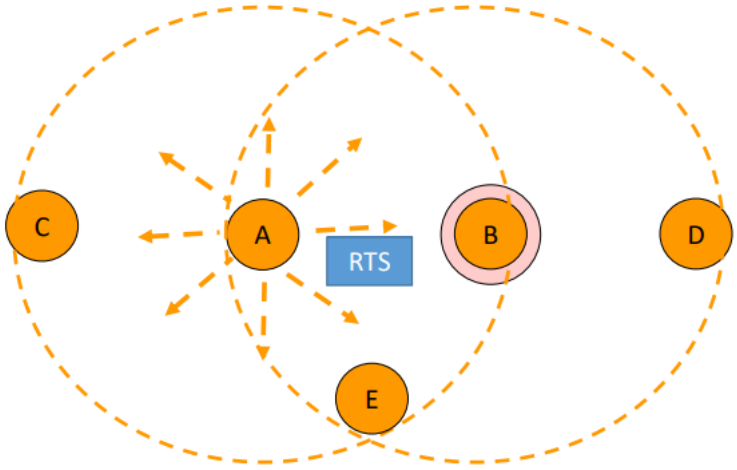

Figure 7.13: **Step 1**: A wants to transmit to B so send a **RTS - Request To Send** packet to B: *the short frame* sended include also the length of the data frame that will eventually follow.

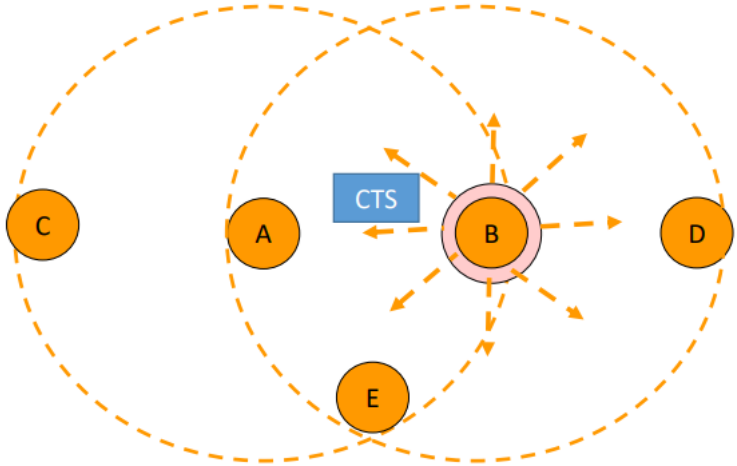

Figure 7.14: **Step 2**: B, C and E receive the **RTS** from A (*because they're in A's radio range*). If B wants to receive the message, it replies with a **Clear To Send - CTS** that is a short frame with data length copied from RTS sended by A.

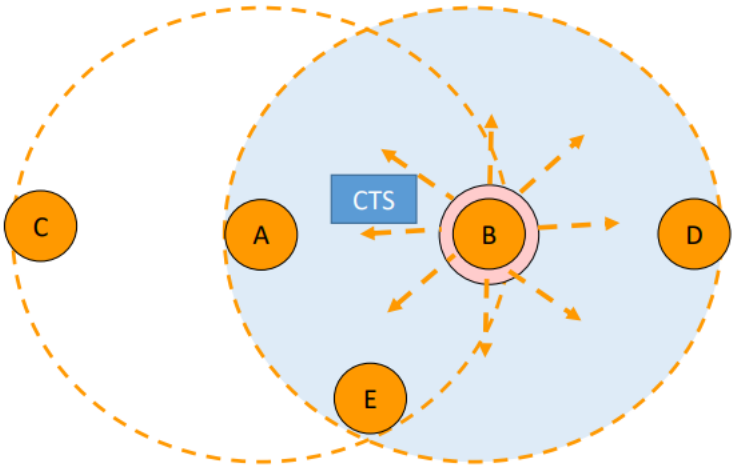

Figure 7.15: **Step 3**: the **CTS** sended by B is received by A, D, E

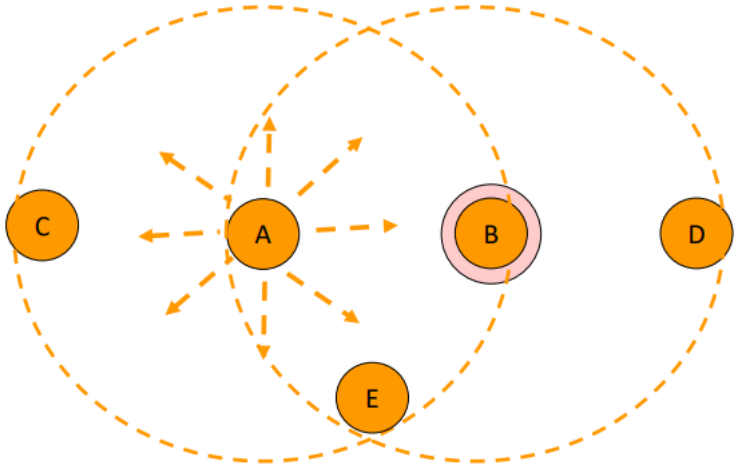

Figure 7.16: **Step 4**: upon reception of the **CTS** frame by B, A (*whose the only interested in receiving*) start to transmit the data frame

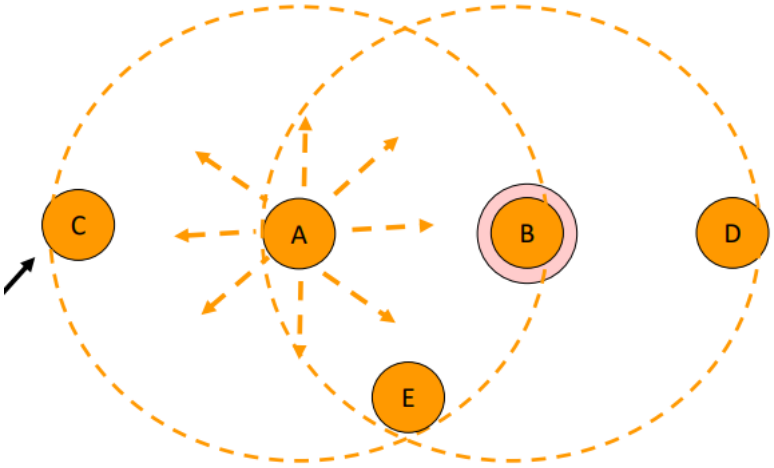

Figure 7.17: **C hears the RTS from A but not the CTS from B** so it is free to transmit (*may be interested in transmitting)*. So **C is exposed**

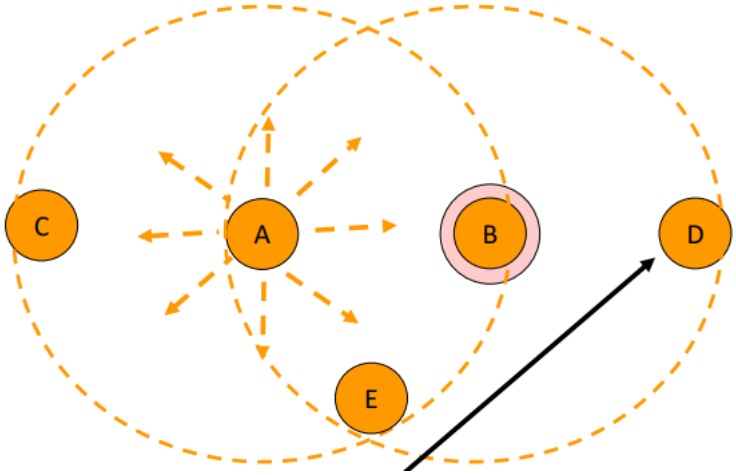

Figure 7.18: **D hears CTS from B nut not RTS from A** so should stay silent until data framr transmission completes. So **D is hidden**

MACA collisions Given the previous scenario, *what happen if anoter node out of the range of A and B wish to trasmit a message to node E?*

- C and B sends *RTS* simultaneosly to A: **the two RTS messages collide so no CTS message is generated**.
- C and B use *Binary Exponential Backofff* to retry send the RTS.

MACAW: MACA for Wireless Network

Fine tunes MACA to improve performance:

- introduces an **ACK frame** to acknowledge a successful data frame
- added **Carrier Sensing** to keep a station from transmitting RTS when a nearby station is also transmitting an RTS to the same destination
- exponential backoff is run for each separate pair source/destination and not for the single station
- mechanisms to exchange information among stations and recognize temporary congestion problems
- CSMA/CA used in IEEE 802.11 is based on MACAW

Questions (2)

- Exposed
- Hidden
- Collision

7.1 IEEE 802.11

The IEEE 802.11 identify a protocol family: each release is identified by a subsequent letter. The firstone was the legacy mode and rarely used: was at 1-2 Mbps data rate implemented via Infrared signals with a radio frequencies in the `2.4 GHz` band (*as frequency range*). Subsequent releases were:

- `802.11a` : operate at 5GHz band with a throughtput of 23 Mbps and data rate of 54 Mbps.
- `802.11b` : operate at 2.4GHz but poses a problem of interference with other appliances (*cordless telephones, microwave, etc*). Had a throughput of 4.3 Mbps and data rate maximum of 11 Mbps.
- `802.11g` :
- `802.11n` : operate at 2.4GHz band and 5GHz It support **MIMO** technologies for using multiple antennas at the transmitter and receiver
- `801.11ac - WiFi 5`
- `801.11ax - WiFi 6` The table in 7.19 provide a short summary.

  All of those use *CSMA/CS* for Multiple Access and have base-station (*infrastructure*) and ad-hoc network versions (*implementing function to access the medium, see previous).* With lower frequencies we have longer wave length: it suitable for IoT sensors. Despite the max dara rate is not huge remains suitable for transmitting data.

We ca have also different implementation at **data link layer** of both **MAC sublayer** and *Logical Link control* as shown in **??**.

Architecture A group of stations operating under a coordination function may or may not use a *base station* or *Access Point (AP)*. If using AP a station communicates with another by channeling all the traffic through a centralized AP: so AP provide connectivty with other APs and other groups of stations via fixed infrastructure. It can also suppoer *ad hoc networks* by introducing a *group of stations that are under the direct control of a single coordination function without the aid of an infrastructure network.*

IEEE 802.11 standard	Year	Max data rate	Range	Frequency
802.11b	1999	11 Mbps	30 m	2.4 Ghz
802.11g	2003	54 Mbps	30m	2.4 Ghz
802.11n (WiFi 4)	2009	600	70m	2.4, 5 Ghz
802.11ac (WiFi 5)	2013	3.47Gpbs	70m	5 Ghz
802.11ax (WiFi 6)	2020 (exp.)	14 Gbps	70m	2.4, 5 Ghz
802.11af	2014	35 – 560 Mbps	1 Km	unused TV bands (54-790 MHz)
802.11ah	2017	347Mbps	1 Km	900 Mhz

Figure 7.19: undefined undefined

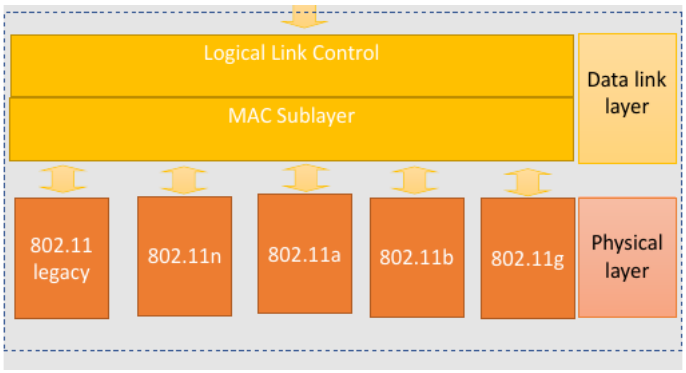

Figure 7.20: undefined undefined
??

Channel association

The spectrum of a mean is divided into channel at different frequencies: the AP admin chooses the frequency for AP. At this level the interference is possible because channels can be same as that chosen by neigboruing AP.

A new host must be associated with an **AP**: first it scan the channel frequencies and listen for *beacon frames* (*like a sort of hello frame, advertise device on the device on the same frequency and sended by the AP*). The beacon frame contains the name of the network (SSID) and MAC address of AP (BSSID). The node select one AP to associate and perform authentication (if needed) and then run the DHCP protocol to get IP address in the AP's subnet.

Active/Passive scanning AP periodically sends beacons frames on their frequency: the node when arrived detect beacons frame and select AP. Nodes select the AP and send the association request. The choose of the AP made by nodes is not mandatorial descripted in the specific: usually is selected the AP with the best **signal strenght** that also is indicative of the number of communication opened by AP with other nodes so is a measure of quality/disponibility of resources. The **passive scanning** flows as following:

1. Beacon frames sent from APs
2. association request frame sent so *H1* to selected AP
3. Association Response frame sent: selected AP to H1

The **Active scanning** flows:

1. Probe Request frame broadcast from H1
2. Probe Response frames sent from APs
3. Association request frame sent: H1 to selected AP
4. Association Response frame sent from selected AP to H1

MAC Sublayer A sketched version of the MAC sublayer is presented in 7.21.

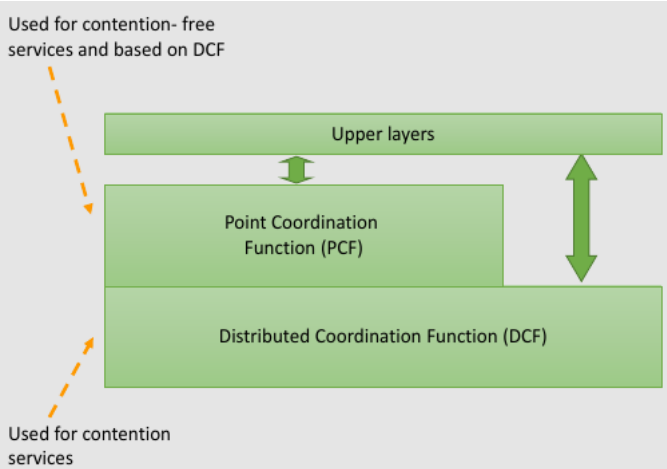

Figure 7.21: undefined undefined

- **Point Coordination Function (PCF)** : uses base station to control all activity in its cell. AP poolls stations for transmission. It's based on DCF.
- **Distributed Coordination Function (DCF)**: used for contention services, is based on MACA.

DCF must be implemnted by all stations: it's completely decentralized and use a best affort asyncronous traffic. Stations must contend for the channel at each frame using CSMA/CA. *Carrier sensing* is performed at two levels:

- *Physical*: by checking the frequency to determine whether the medium is in use or not. It detect an incoming signal and any activty in the channels due to other sources.
- *Virtual*: is performed by seding a duration information in the header of an RTS, CTS and data frame. So keeps the channel *virtually busy* up to the end of data frame transmission. A channel is marked **busy** if either the physical or the virtual carrier sensing indicate busyness.

Priority access to the medium is controlled through the use of **interframe space (IFS)** time intervals. The IFS is a mandatory periods of idle time on the transmission medium.
Three IFS specified by the standard:

- *Short (SIFS)*:
- *Point Coordination Function (PIFS)*:
- *Distributed Coordination Function IFS (DIFS)*: The hierarchical order based on priority is given by: `SIFS < PIFS < DIFS`

  The exchange of RTS and CTS between sender and receiver allows the other nodes to calculate the *NAV (Network Access Vector)* that is an **estimation of duration** of communciation between sender and receiver (*a sort of period of silence to avoid illegal accesses to the mean*), as shown in 7.22.

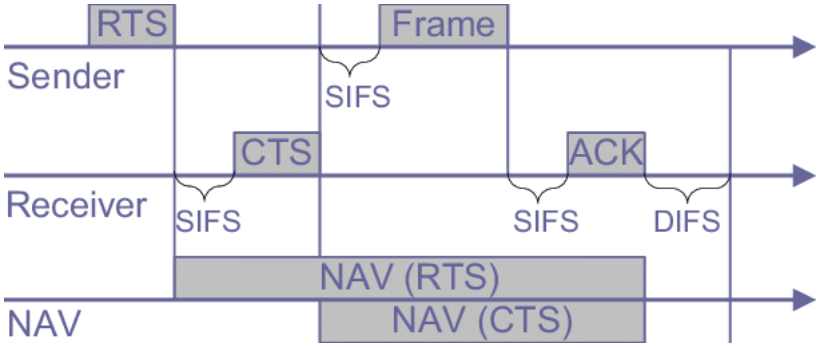

Figure 7.22: Exchange flow to determine the *NAV*

7.1.1 Mobile Networks

Nowadays there are more mobile-broadband-connected deviced than fixed-broadband. the 4G-5G cellular network now embrace internet protocol stack, including SDN. There are two important different challenges:

- *Wireless*:
- *Mobility*: handling the mobile user who changes point of attachment to network or by the provider that wants advertise a service It also poses authentication and authorization problems towards user requested services.

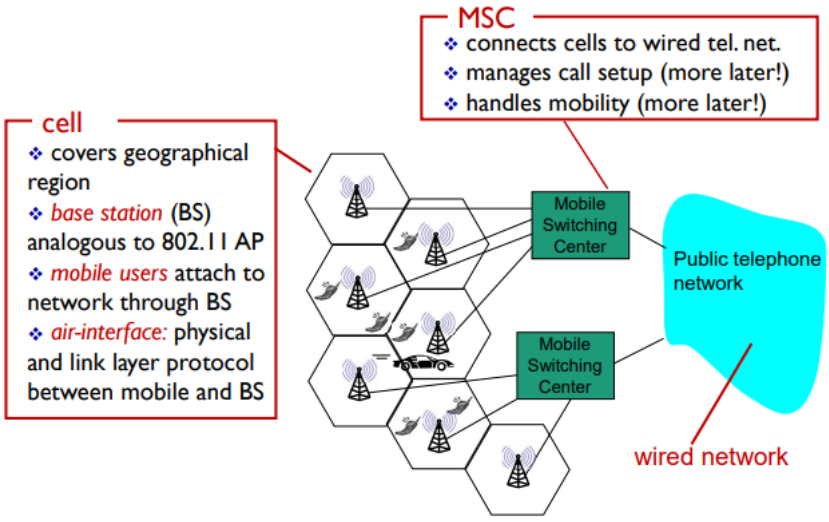

Figure 7.23: undefined undefined

Components of cellular network architecture The *MSC* intermediate the connection betwwen different cells sets of networks, it also manage the billing information by tracking times and call setup. MSC can manage more than 4 cells or *base station*: some MSC can be also used as gateway to the *wired-telephone network*.

From 4G to 5G we have a change in the architecture, integrating software engineering choices in new components like supporting orchestration and function virtualization (*More at https://doi.org/10.3390/en12112140*).

With the evolution of mobile network technology both evolves with the *access techniques*, as the time diagram show in 7.24.

FDMA/TDMA

Frequency Division Multiplexing Access/Time Division Multiplexing Access are two techniques used for **sharing mobile-to-BS (*base station*) radio spectrum**. They can be combined in one by *dividing the spectrum in frequency channels and divide each channel into time slots* as pictured in 7.25

Another methods is *CMDA - Code Division Multiple Access*: is a digital cellular technology that allows multiple users to share the same frequency band simultaneously. It works by **assigning a unique code to each user, which is used to modulate the user's signal.**

CDMA uses spread spectrum technology, which spreads the user's signal across a wide bandwidth. This makes it possible for multiple users to transmit their signals at the same time and on the same frequency band without interfering with each other.

The **unique code assigned to each user acts as a key to unlock the signal at the receiving end**. The receiver is able to identify and separate the different users' signals based on their unique codes, allowing each user to receive their own signal without interference from other users, as pictured in 7.26

Even in case of different generated signal, if the receiver is able to known the code used by the sender, even if the received signal is the overlap of different codes of different sender the receiver will be able to reconstruct the data. The specific code used by the sender is negoatiated beforehand with the receiver.

2G Network Architecture (voice)

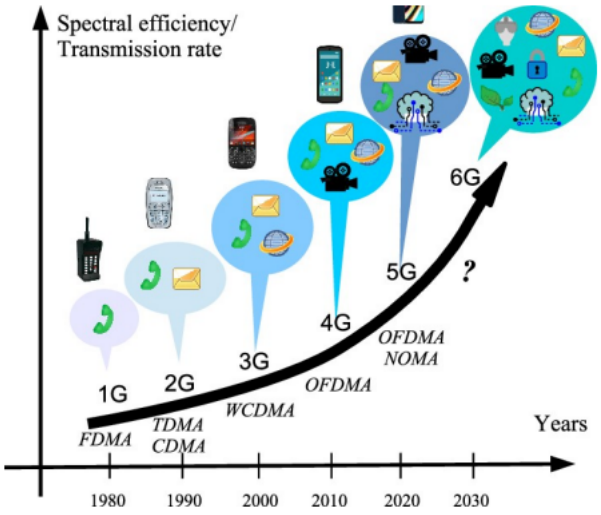

Figure 7.24: undefined undefined

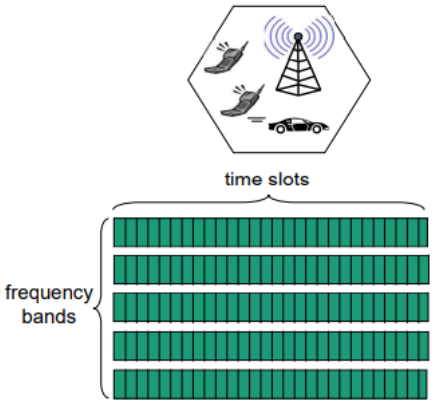

Figure 7.25: undefined undefined

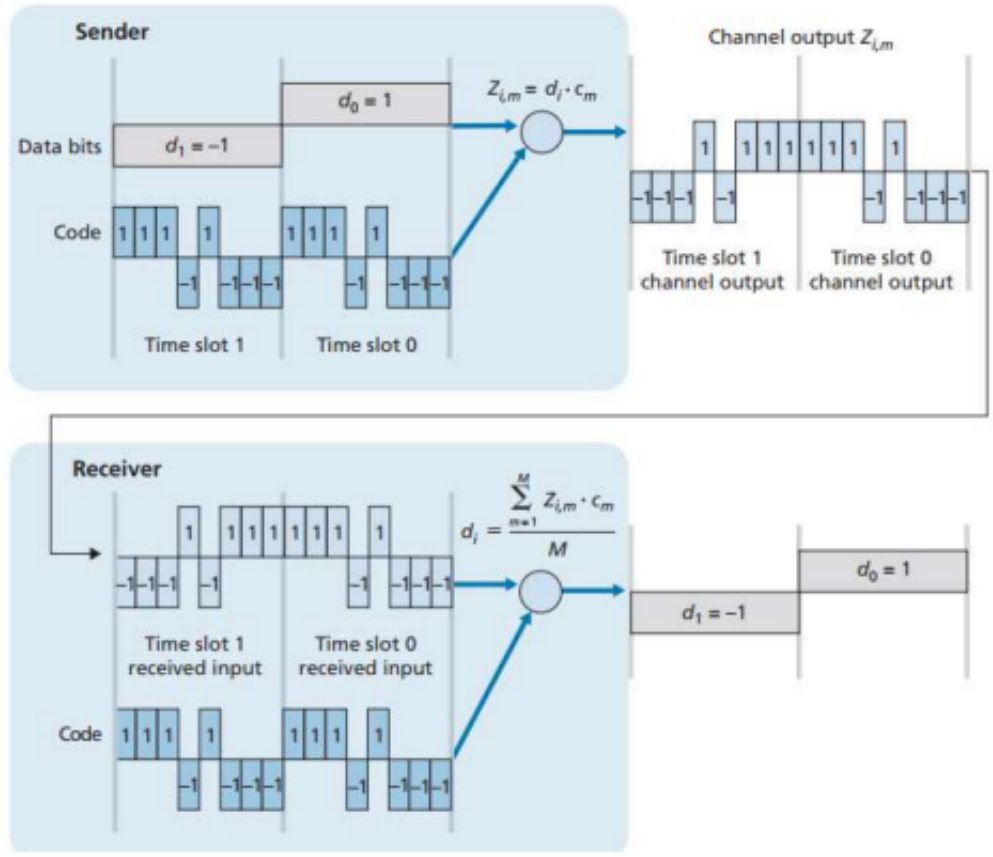

Figure 7.26: undefined undefined

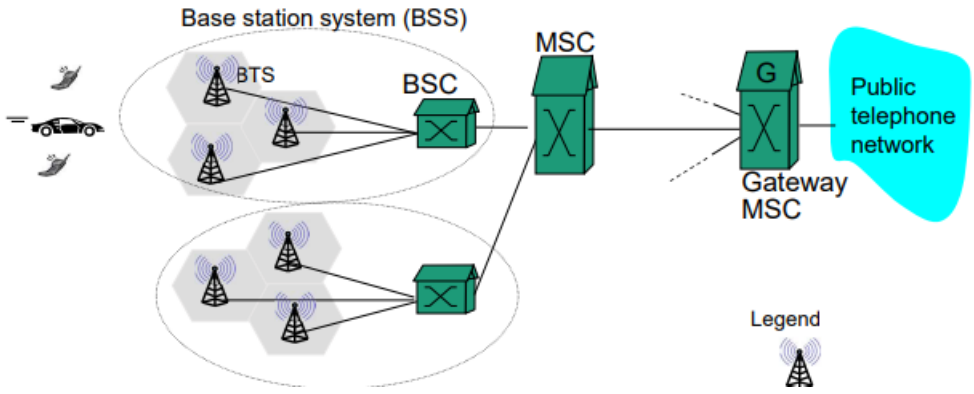

Figure 7.27: 2G network architecture for voice communication

Chapter 8

Mobile Networks

3G Network Architecture (voice+data) New cellular data network operate in parallel (*except at the edge*) with existing cellular voice network. The voice network is unchanged in core and operates *circuit switching* while *data network * 2G Network Architecture (voice)ates in parallel by operating **packet switching**. The general schema is pictured in 8.2

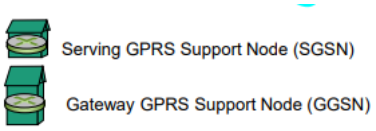

Figure 8.1: Symbol reference

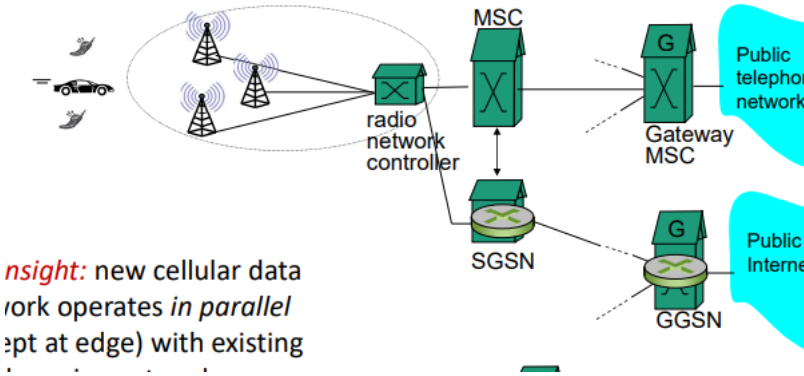

Figure 8.2: 3G Network Architecture

8.0.1 3G vs. 4G LTE (Long Term Evolution) network architecture

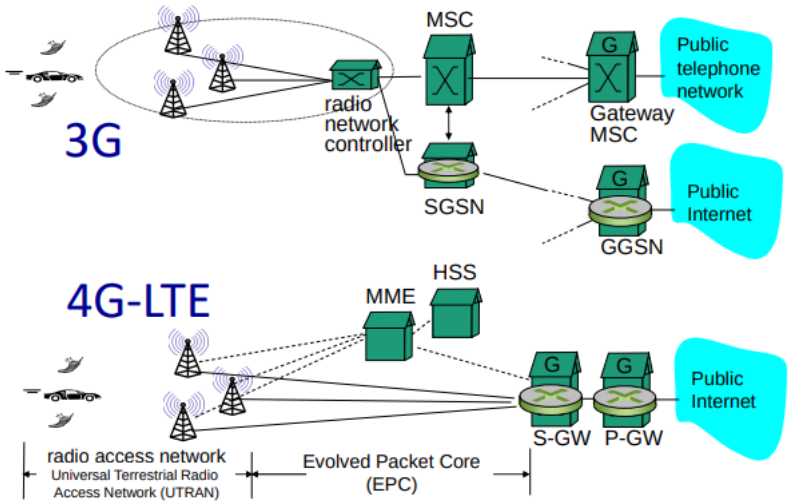

Figure 8.3: 3G vs 4G LTE Network Architecture

The changes between the 3G and 4G netowkr architecture are in the core network, like:

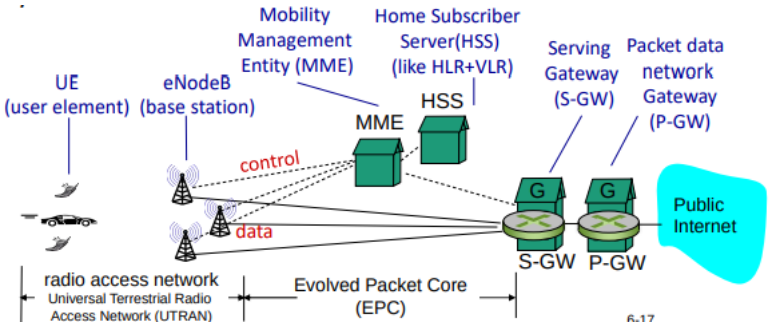

Figure 8.4: 4G-LTE Network Architecture

- **All-IP core**: IP packets are tunneled from base station to gateway and there is no separation between voice and data so all traffic is carried over IP core to gateway. The architecture is pictured in 8.4

8.0.2 4G/5G cellular networks

There are some similarities to *wired internet* like:

- edge/core distinction where we have application services (edge) and core (e.g. services to authenticate devices)
- Global cellular network: operators agree on a network of networks among all geographical countries
- Use of protocols like HTTP, DNS, TCP, UDP, DHCP
- Mobile networks are interconnected to internet to be able to use services

Between differences from wired internet there are:

- Mobility as a first class service
- User *identity* via SIM card
- Business models: user subscribe to a cellular provider called "home network" that works differently when roaming on visited nets. The global access is granted thanks to authentication infrastructure and inter-carrier settlements

Architecture

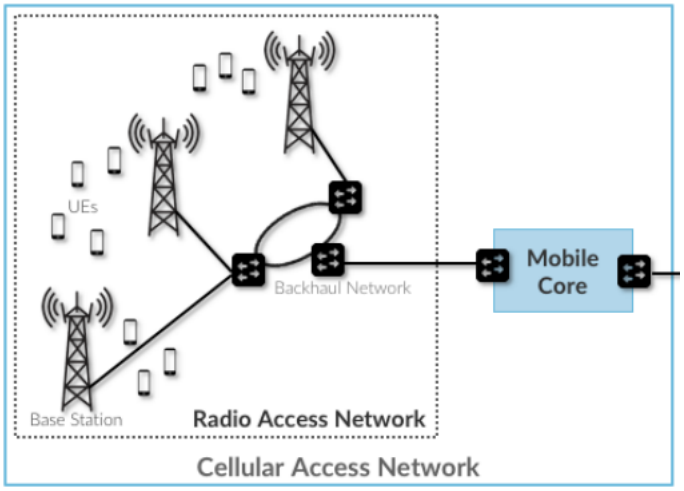

Figure 8.5: undefined undefined

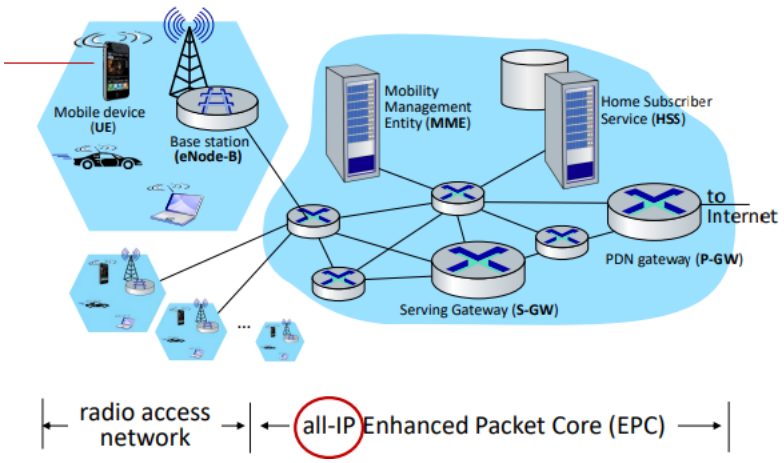

Figure 8.6: undefined undefined

Pasted image 20230308164331.png

Figure 8.7: Mobile core architecture

- **Mobile Core**: provides IP (*internet*) connectivity for both data and voice servics. Also ensure QoS requirements and allows to tracks user mobility to ensure uninterrupted service, providing *billing* and *charging*.
- **Blackhaul Network**: allows to connect base station to mobile core (*interconnecting the RAN*) by using optics or wireless solution (*IAB - Integrated Access Blackhaul*).
- **Radio Access Network (RAN)**: it's a distributed connection of Base Stations (BSs) and allows to manage the radio spectrum.

Elements of 4G architecture

In the **radio access network** the mobile device can be a large series of devices like *laptop, ioT* devices with 4G LTE radio: they are identified by 64 bit *International Mobile Subscriber Identity (IMSI)* stored on the **SIM - Subscriber Identity Module**.

The **core network** is composed by ***Home Subscriber Service - HSS*** that contains information about subscribers with their identifiers and allows to manage the authentication phase. It mantains the information about the device's "*home network*".To be able to authenticate, it works with ***Mobile Management Entity - MME***.

The **Mobile Management Entity** perform the device authentication, coordinated with mobile home network HSS. The *MME* is also involved when changing the base station and allows to track the device and page the device location (*ask the base station for the presence of this device*). For data packets transfer we need to establish a *data path* from the mobile device to the ***Serving Gateway (S-GW)*** (the *tunneling* and the data path is explained later). Both the *Serving Gateway (S-GW) and PDN Gateway (P-GW)* are on the path from mobile to/from internet network: the S-GW forwards IP packets to and from the RAN, the P-GW is a gatway to mobile cellular network that looks like any other internet gateway router. It provides *NAT services* and support access-related functions like *policy enforcement, traffic shaping and charging*. When devices move to one base station to another,also the S-GW gateway is involved in this process.

A simplified view of the *mobile core* architecture is pictured in 8.7

For example, a single **MME/PGW** pair might serve a metropolitan area, with SGWs deployed across 10 edge sites spread throughout the city, each of which serves 100 base stations.

In 4G there is a strong separation between **data plane** and **control plane** that allows the mobility management and the boostrapping of the data path:

The **base station** forwards both control and user plane packets between the mobile core and the user. The separations concerns:

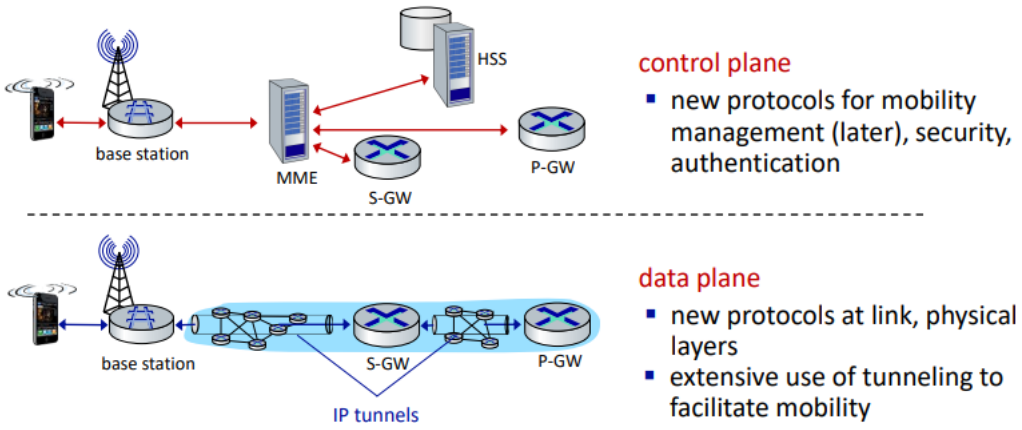

Figure 8.8: undefined undefined

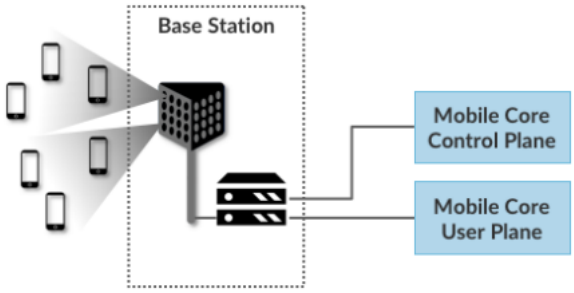

Figure 8.9: undefined undefined

- ***Control plane***: the packts are tunneled over SCTP/IP (Stream Control Transport Protocol) that is an altenative reliable transport to TCP, tailored to *carry signaling (control)*. It allows to provide user's device authentication, registration and mobility tracking.
- ***User plane***: packets are tunneled over **GTP (General Packet Radio Service) Tunneling Protocol.

The Long Term Evolution (LTE) **data plane protocol stack** in the *core network* adopt, as already mentioned, the **tunneling mechanism** (*as pictured in 8.10*): mobile datagrams are encapsulated using GPT and sent inside a UDP packet to the Serving Gateway (S-GW). The S-GW re-tunnels the UDP datagrams to P-GW, allowing reaching internet networks. The tunneling mechanism described allow supporting *mobility*: only tunneling endpoints change when mobile user moves. In this context the S-GW represent a it's a fixed point both from the device to internet and viceversa so if the device change base station, the S-GW is able to *retrieve* the new base station of the device and change the IP address to deliver the packet to the correct base station in which the device is connected. This mechanism allows to implement mobility across base stations.

LTE: Link Layer Protocol The **first hop LTE Link Layer protocols** (8.11) in the IP range defines:

- **Packet Data Convergence**: contains header for compression, decompression, encryption and decryption.
- **Radio Link Control (RLC)**: implement fragmentation/re assembly by ensuring reliable data trasnfer.
- **Medium Access**: requesting and use of radio *transmission slots*

Between the *LTE radio access network* we have several mechanism like:

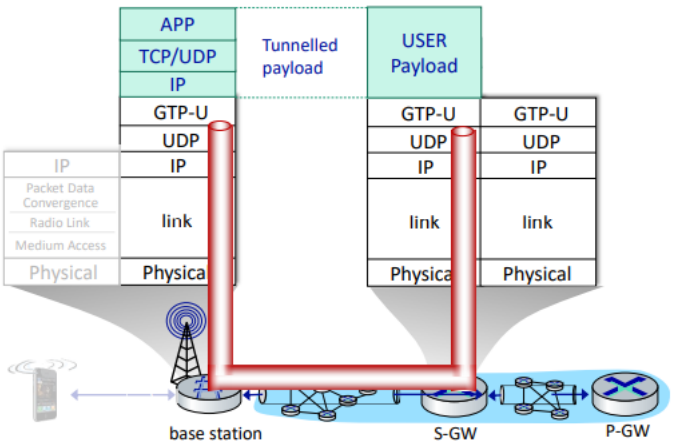

Figure 8.10: undefined undefined

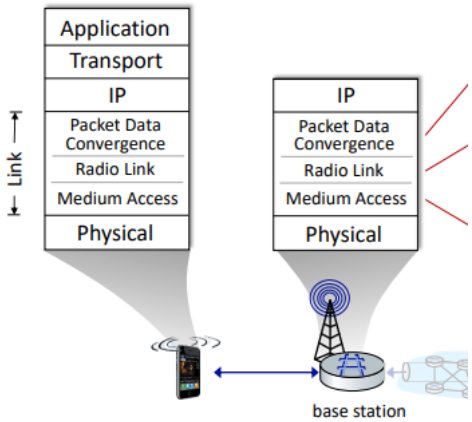

Figure 8.11: undefined undefined

- **Downstream channel**: combination of FDM and TDM within frequency channel (*OFDM - Orthogonal Frequency Division Multiplexing: orthogonal refer to minimal interference between channels.*)
- **Upstream**: FDM, TDM similar to OFDM. Each **active mobile device allocated two or more 0.5 ms timeslots in or more channel frequencies**. The scheduling algorithms is not standardized so it's up to the operator and allows up to 100's Mbps per device.

Associating with a base station

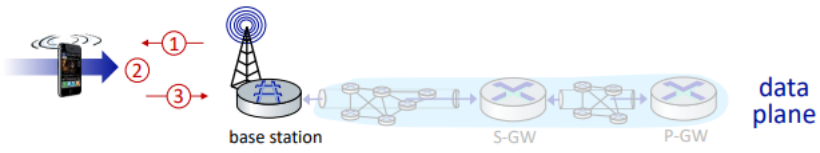

Figure 8.12: Process of association with a base station

1. BS broadcast primary synchronization signal every 5 ms on all frequency
2. Mobile finds a primary synch signal and locate the second synch signal on this frequency: mobile then finds info broadcast by BS like channel bandwidth and configuration. Mobile may get informations from multiple base stations, multiple cellular networks.
3. Mobile selects which BS to associated with (*e.g. prefererence for home carrier*)

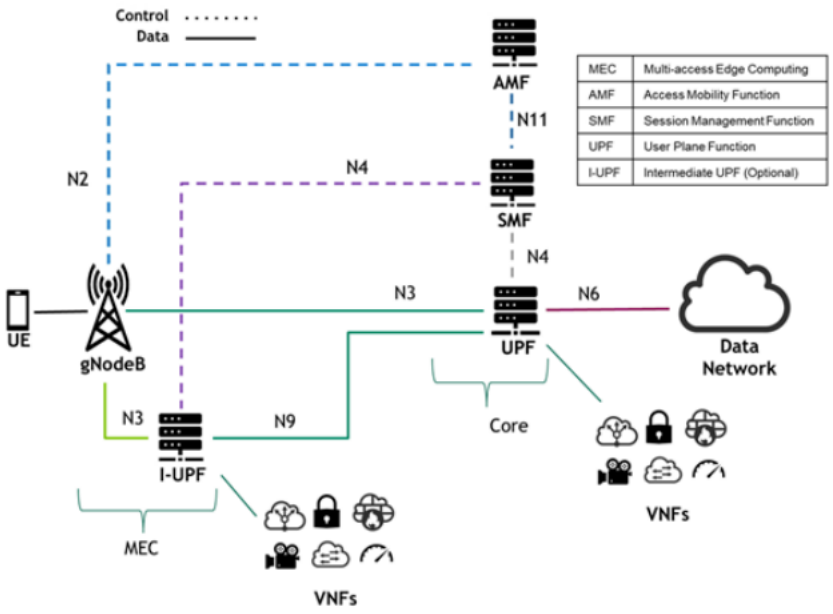

Figure 8.13: undefined undefined

4. More steps till needed to authenticated, establish state and set up control plane.

The **sleep modes** of LTE allows to put radio to *sleep* to conserve battery:

1. *Light Sleep*: afet 100 millisecond of inactivity: need to wake up periodically to check downstrema transmissions.
2. *Deep Sleep*: after 5-10 seconds of inactivity then the mobile may change cells while deep sleeping so need to re-establish association

8.0.3 5G architecture

The main goal of 5G is to icnrease in peak bitrate, 10x decrease in latency and 100x increase in traffic capacity over 4G. THe **5G New Radio Frequencies** have two fruquency bands:

- *FR1*: 450MHz - 6 GHz
- FR2: 24GHz - 52 GHz The 5G is not backwards-compatible with 4G: the *millimiter wave frequencies* have much higher data rates but over shorter distances because uses **pico-cells** that have a diameter of 10 to 100m. This implies a dense deployment of *new base station.*

The underlying idea is to realize the functions of authentication, tracing and so on by means of **microservices** implemented throught ***Network Function Virtualization***: this allows the deploy of those function is both close to the core network and base stations so the path between the mobile device and the function/service is really close, reducing the latency and load balancing the traffic. This also avoid traffic directly flow into the core network and manage the traffic at the edge.

5G User Plane

The 5G user plane is sketched in 8.13.

The **UPF - User Plane Function** allows to forward traffic between RAN and internet, corresponding to the S-GW/P-GW combination into the **EPC**. It's also responsible for:

- Packet inspection and application detection
- QoS management
- Traffic usage reporting This component allows flexibility to deliver user plane functionality at the edge as well as the network core because UPF can be co-located with edge and central data centers at both locations, implementing laso **MEC - Multi Access Edge Computing** (see next lecture).

5G Control Plane

The basic 5G architecture schema is sketched in 8.16.

- ***AMF (Core Access and Mobility Management Function)***: connection and reachability management, mobility management, access authentication and authorization, and location services
- ***SMF (Session Management Function)***: manages each UE session, including IP address allocation, selection of associated UP function, control aspects of QoS, and control aspects of UP routing.
- ***PCF (Policy Control Function)***: manages the policy rules that other CP functions then enforce.
- ***UDM (Unified Data Management)***: Manages user identity, including the generation of authentication credentials.
- ***AUSF (Authentication Server Function)***: Essentially an authentication server.
- ***SDSF (Structured Data Storage Network Function)***: a "helper" service used to store structured data.
- ***UDSF (Unstructured Data Storage Network Function)***: A "helper" service used to store unstructured data.
- ***NEF (Network Exposure Function)***: A means to expose select capabilities to third-party services
- ***NRF (NF Repository Function)***: A means to discover available services.
- ***NSSF (Network Slicing Selector Function)***: A means to select a Network Slice to serve a given UE. Network slices are essentially a way to partition network resources in order to differentiate service given to different users

Deployment options

Two main deployment options are considered (*as shown in 8.15*):

1. **Non standalone (4G+5G) over 4G's EPC**: first schema in which 5G base stations are deployed alongside existing 4G base stations ina given geography to proide a data-rate and capacity boost. In NSA, the control plane traffic between user equipment and the 4G Mobile Core utilizess 4G base stations and the 5G base stations are usd only to carry user traffic.
2. **Non standalone (4G+5G) over 5G's NG-Core**: second schema

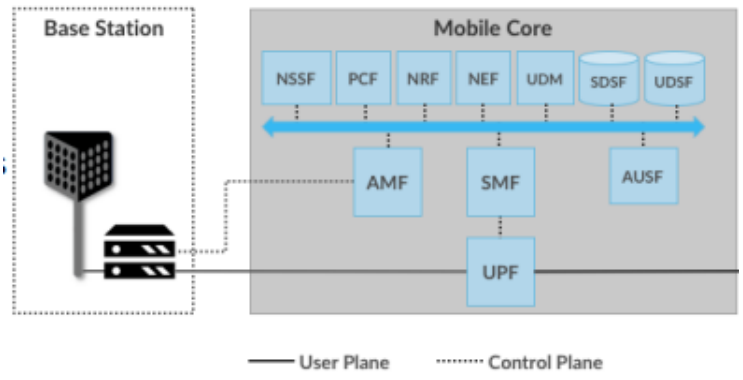

Figure 8.14: 5G Control Plane

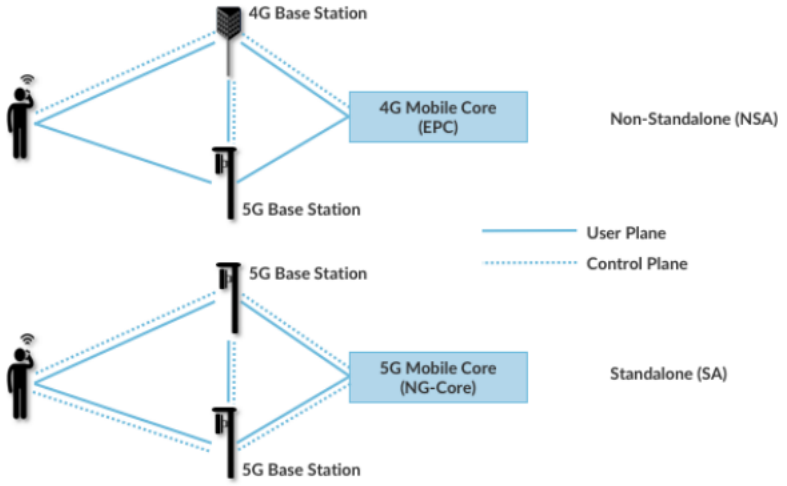

Figure 8.15: Non standalone (4G+5G) over 5G's EPC vs over 5Gs NG-Core

8.0.4 Mobility

From the network perspective, the spectrum of mobility have a wire area of scenarios and application, as shown in 8.17.

The **home networks** allows to have a service plan with a cellular provider (*Verizon, Orange*) where

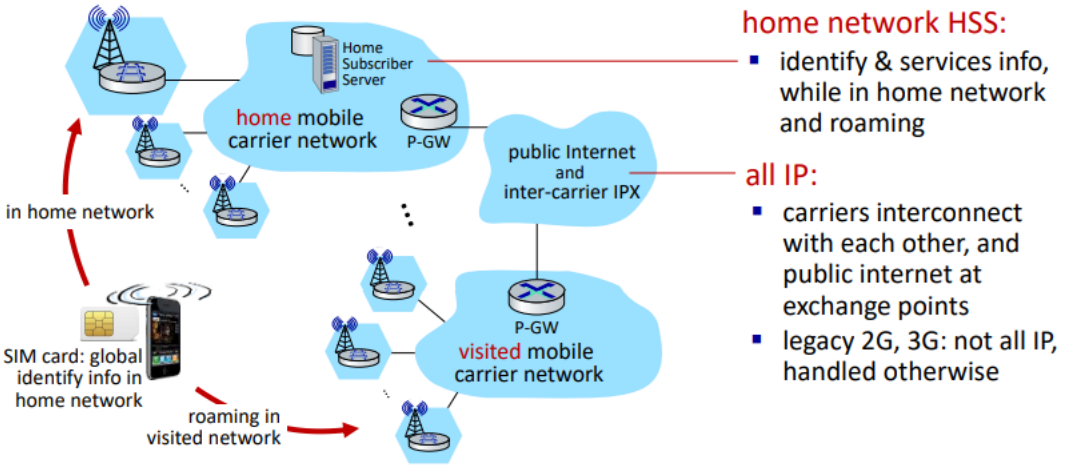

Figure 8.16: Mobility scenario

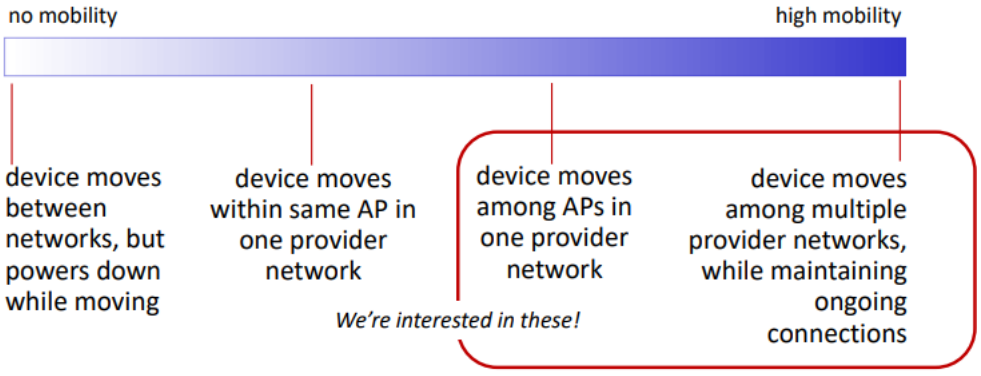

Figure 8.17: Mobility spectrum

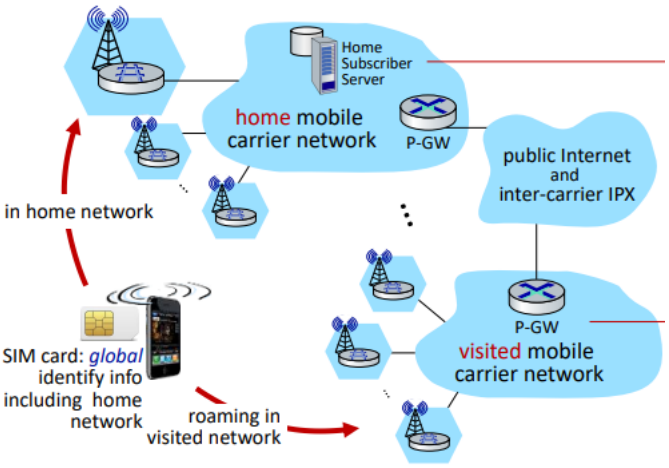

Figure 8.18: Flow between home network and host network for mobility purposes

the home network HSS stores identity and service informations of the device. The **visited network** is any network other than your home network and by service agreement with other networks allows to provide access to *visiting mobile*, implementing mobility. The described scenario is pictured in 8.18.

Wifi/ISP does not have a concept of Home network where the user informations are stored but they are stored on the device or with the user (eg. username, password), different networks different credentials (*Think of eduroam*).

To implement mobility and have a correspondence between the mobile device and the home network/visited network we have two different approaches:

1. **Network router handle it**: the network router handle the management of the mobility. The routers advertise well-known name, addresses (*e.g. permanent 32-bit IP address*) or number (*phone cell*) of visting mobile node via usual *routing table exchange*. Internet rotung could do this already without changes: routing tables indicate where each mobile is located by using **Longest Prefix Match**. This approach is not scalable to a billions of mobiles: we could have routing tables too large to expand and retrieve data (*coconsidering 32 bit IP addresses*).

2. **End-system handle it**: the funcionality is provided at the *edge*. There are two types of routing:

   (a) ***Indirect routing***: communication from correspondent to mobile goes through home network and then forwrded to remote mobile (*connected to a thid-party visited network*). The correspondant, known the home network gateway send the request and thanks to the HSS - Home Subscriber Server that contains the updated address of the device, is able to forward the request to the Serving Gateway of the *visited network*. Refer the schema in 8.19.

   (b) ***Direct routing***: correspondents get foreign address of mobile, sending directly to mobile device. The *IMSI* is used as a sort of *MAC address* of the device. Refer the schema in 8.20.

In the ***indirect routing*** a mobile-device-to-visited network association protocol is needed: the mobile device will need to assicate with the visited network and will similarly need to disassociate when leaving the visited network. There is also the need of a *visited-network-to-home-network-HSS registration* to allows visited network to register mobile device's location with the HSS In the home network. A datagram tunneling protocol between the home network gateway and the visited network gateway router is needed to allows home gateway performs encapsulation and forwarding of the correspondents original datagram and, on the receiving side, the gateway router performs decapsulation, NAT translation and forwarding of the original datagram to the mobile device.

In the ***direct routing*** the process is less transparent o the correspondent because it must get care of the address from home agent. In case the mobile changes visited network the request sending can be handled but with additional complexity by allowing the HSS to be queried by the correspondent only at the beginning of the session.

This last approach require a **registration** procedure to be sure that the home network known there the device is, as pictured in 8.21.

The mobile device needs an IP address in the visited network:

1. permanent address associated with the mobile device's home network

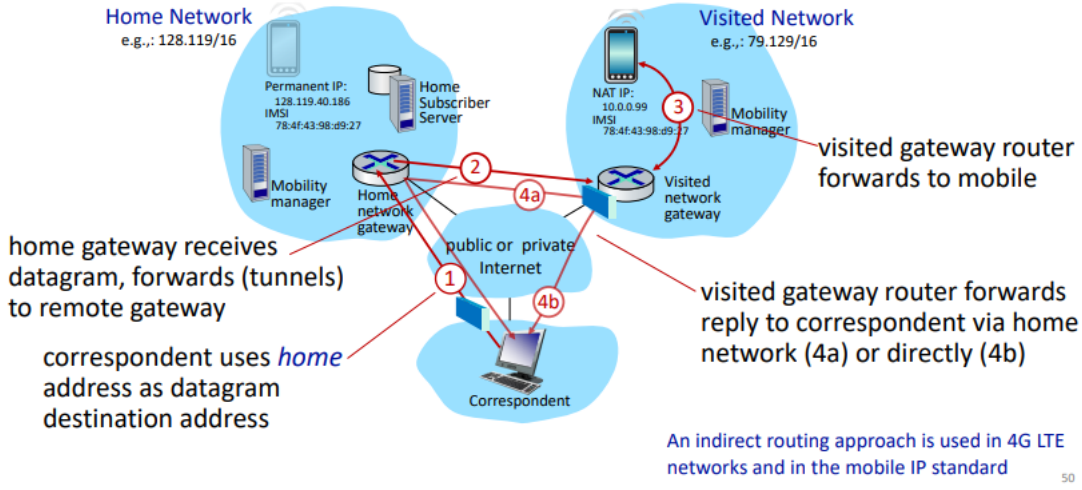

Figure 8.19: Indirect routing schema

2. New address in the address range of the visited network
3. IP address via NAT The end result is that the visited mobility manager known about the mobile and the home HSS knowns the location of mobile.

In case on **indirect routing**, the scenario is different:

1. The correspondent use `home` address as datagram destination address
2. The home gateway receives datagrams and forwards (*by tunneling*) to remote gateway
3. Visited gateway router forwards to mobile
4. The visited gateway riyter forwards reply to correspondent via home network **(a)** or directly **(b)**. The main requiremenets for mobility in *indirect routing* are:
5. *Mobile-device-to-visited-network association protocol*: the mobile device will need to associate with the visited network and will similarly need to dissociate when leaving the visited network.
6. *Visited-network-to-home-network HSS registration*: The visited network will need to registrer the mobile device's locatio with the HSS in the home network
7. *A datagram tunneling protocol between home network gateway and visisted network gateway router*: the home gateway perform encapslation and forwarding of the correspondent's original datagram.

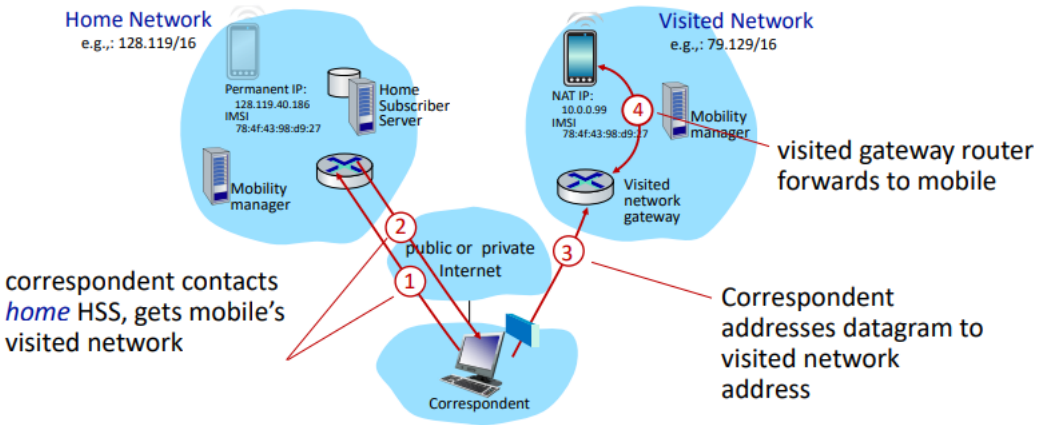

Figure 8.20: Direct routing schema

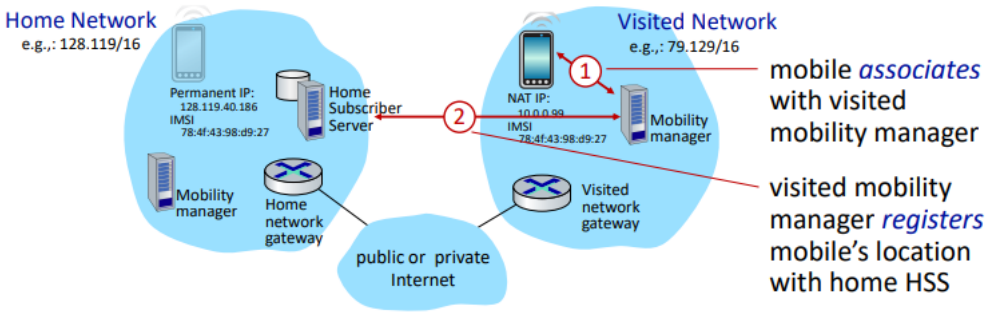

Figure 8.21: undefined undefined

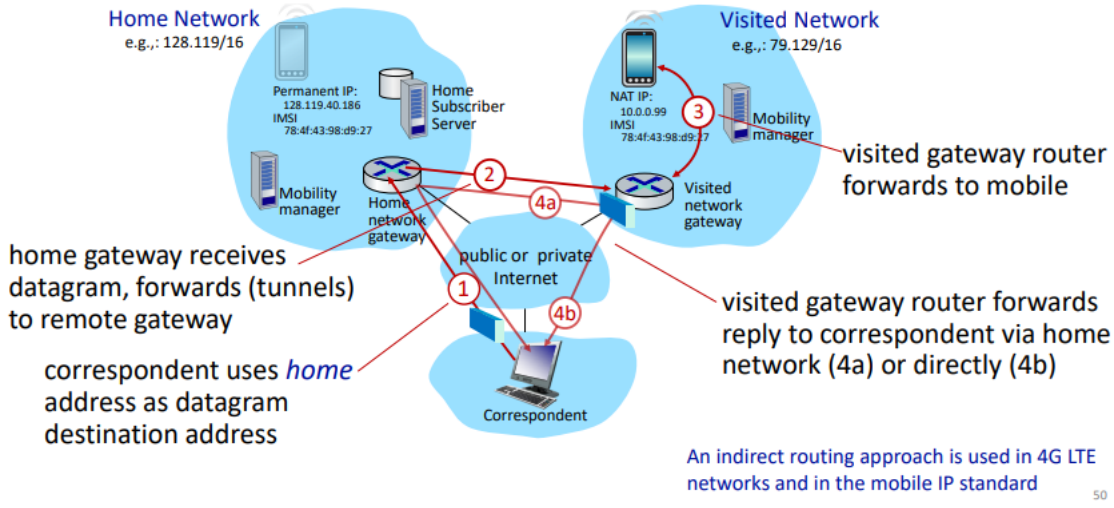

Figure 8.22: undefined undefined

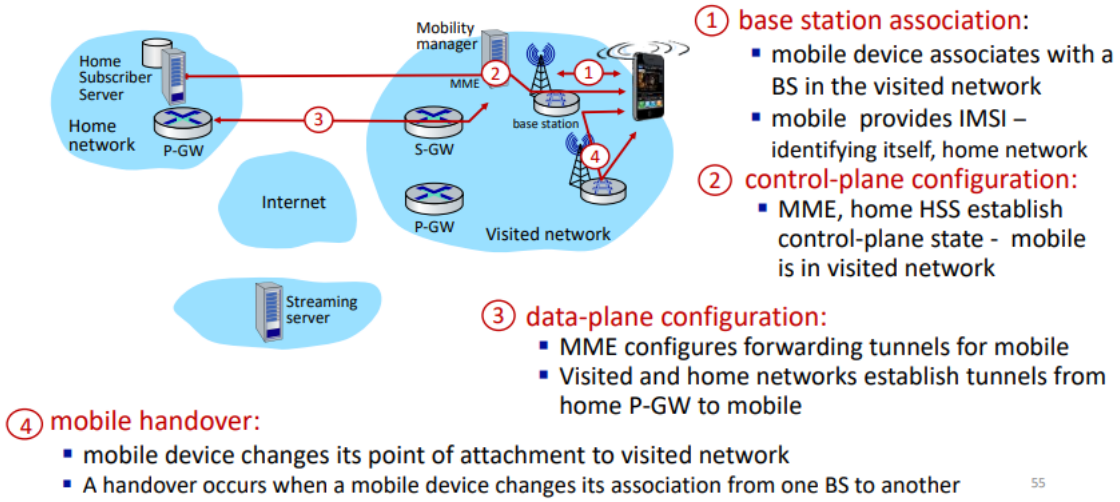

Figure 8.23: undefined undefined

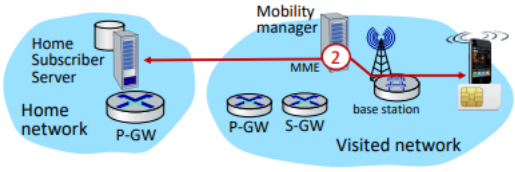

Figure 8.24: undefined undefined

On the receiving side, the gateway router performs ***decapsulation, NAT translation and forwarding*** of the original datagram to the mobile device. Some considerations about ***indirect routing*** evaluate the problem of **triangle routing** that results in an inefficient model when correspondent and mobile are in the same network (*due to the problem of forwarding the messages to the home network and re-routing them to the correspondent's original network which correspond to the visisted network of the destination device*). The mobile device can see an interrupted flow of datagrams as it moves between network, losing some datagrams but the on-going TCP connection between correspondet and mobile can be mantained. The following schema describe the steps. So if the mobile changes visited network it's necessary to update the HSS in the home netowkr, changing the tunnel endpoint to terminate at the gateway router of the new visited network.

Regarding the mobility with ***direct routing***, it overcomes triangle routing ineffiencies at cost of be *non-trasparent to correspondent* because he need to get care of the address from home agent. For this type of routing, if the mobile changes visited network, the HSS is queried by the correspondent only at the beginning of the session, requiring additional complexity.

Mobility in 4G

allows the mobile to communicate witht he local MME via a BS control-plane channel. The MME uses mobile's IMSI information to contact mobile's home HSS to retrieve authentication, encryption and network service information and, in return, the home HSS knowns the mobile now reside in the visited network. The BS and mobile select parameters for BS-mobile and data-plane radio channels.

The MME configures the data plane for the mobile device so all traffic to/from the mobile device will be *tunneled* through the device's home network. Two different tunnels are established: 1. **S-GW to BS tunnel**: when the mobile changes base station, simply change the endpoint IP addres of tunnel 2. **S-GW to home P-GW tunnel**: to support the indirect rotuingg

The **tunneling via GTP (*GPRS tunneling protocol*)*** allows the mobile's datagram directed to the streaming server to be encapsulated using GTP.

The ***handover (also known as handoff)*** is the process of transferring an ongoing call or data session from one base station (BS) to another base station while maintaining the continuity of the communication. Handovers are necessary when the mobile device moves from one cell to another, and the signal strength of the current base station becomes too weak to maintain the connection.

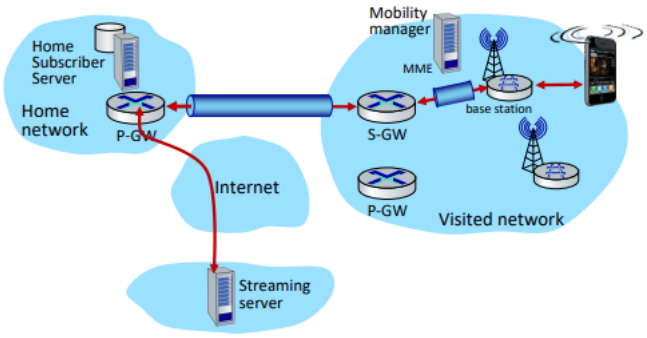

Figure 8.25: undefined undefined

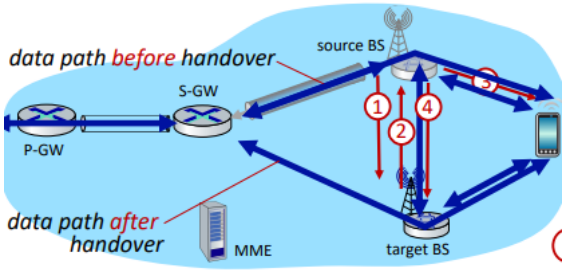

Figure 8.26: undefined undefined

The steps are:

1. Current source BS chooses to select a handover: selects target BS, sends `Handover Request message` to target BS
2. Target BS pre-allocates radio time slosts, responds with `HR ACK` with information for mobile
3. Source BS informs mobile of new BS: mobile can now send via new BS. The handover looks complete to the mobile
4. Source BS stops sending datagrams to mobile, instead forwards to new BS (*who forwards to mobile over radio channel*).
5. Target BS informs MME that it is new BS for mobile: MME instructs the S-GW to change tnnel ednpoint to be new target BS
6. target BS ACKs back to source BS: handover compelte, source BS can release resourcees
7. Mobile's datagram now flow through new tunnel from target BS to S-GW

Mobile IP

The mobile IP architecture use *indirect routing* to node via home networks and using tunnels. Operates by two agents:

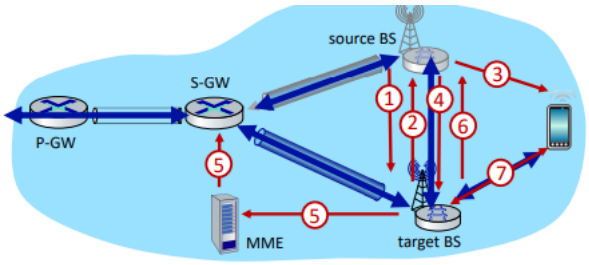

Figure 8.27: undefined undefined

1. *Mobile IP Home Agent*: combined roles of 4G HSS and home P-GW
2. *Mobile IP Foreign Agent*: combined roles of 4G MME and S-GW The protocols for agent discovery in visited network, registration of visited location in home network via ICMP extensions.

Chapter 9

SDN - Software Defined Networking

The **network layer** have two main functions to be carried out:

- **Forwarding**: move packets from router's input to appropaite route output
- **Routing**: determine route takne by packet from source to destination The forwarding activity is implemented by the ***data plane***, the routing is implemented by the ***control plane***. Respectively, both planes operates on fast timescale (*per-packet in case of forwarding*) and slow time scale (*per control event in case of routing*).

Two main approaches exists to structuring the network control plane:

- ***Per-router***: is the traditional one
- ***Logically centralized control***: also known as **Software Defined Networking**.

Per-router control plane

Individual routing algorithm components in each and every router interact with the control plane to compute the forwarding tables as pictured in 9.1. In traditional IP networks, the control and data plae are ***tightly coupled***, embedded in the same networking devices and the whole structure is highly *decentralized*. The *network resilience* was a crucial design goal in the early days of Internet.

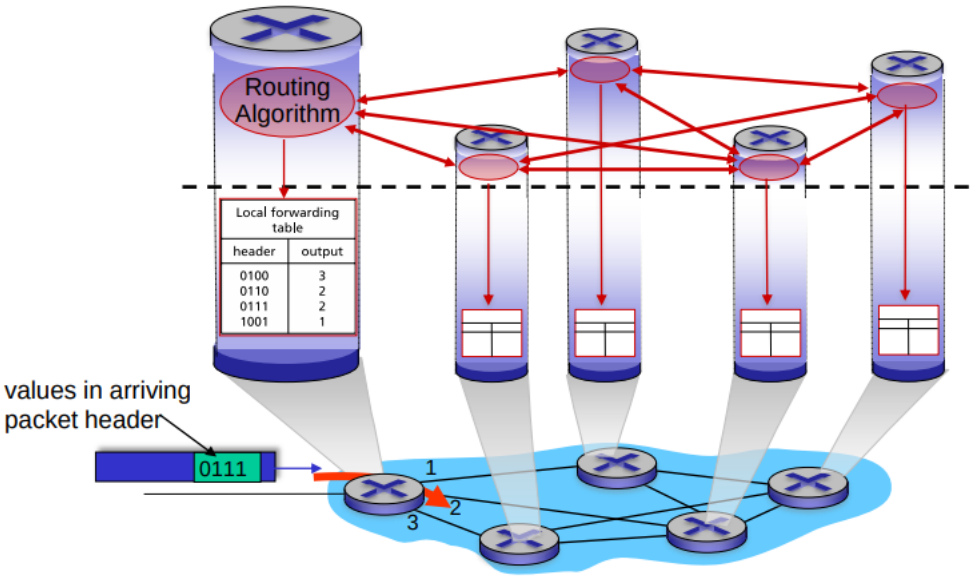

Figure 9.1: undefined undefined

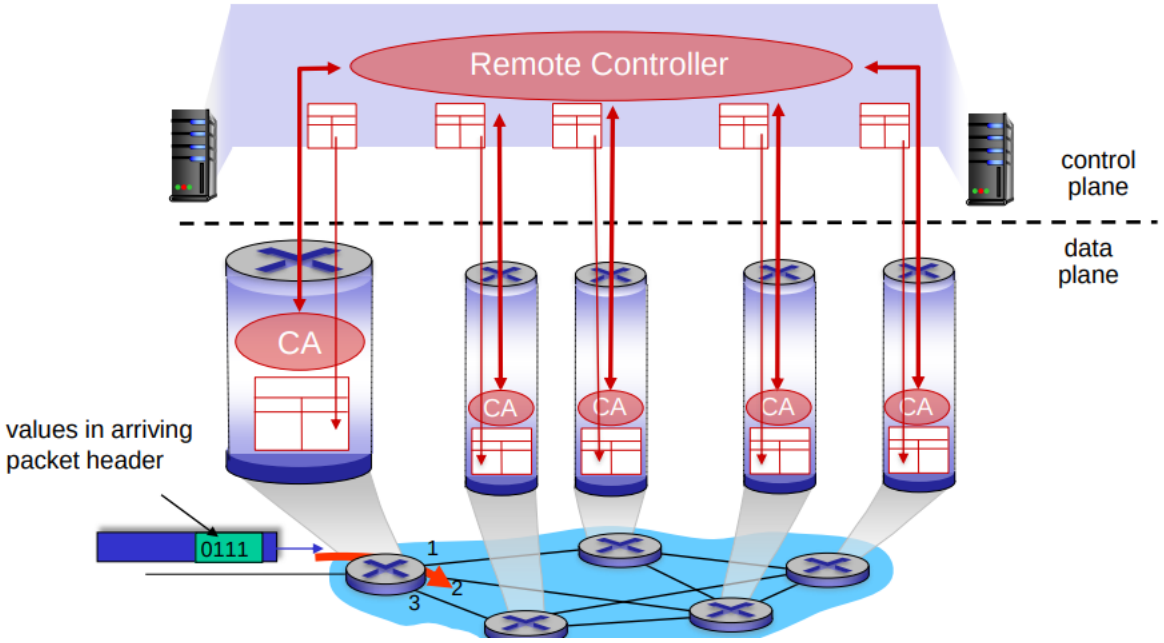

Figure 9.2: undefined undefined

Software-Defined Networking control plane

The ***remote controller*** computes and installs the forwarding tables in routers as pictured in 9.2.

The rationale behind a **logically centralized** control plane reside in the fact that is easier to carry out management network functions by *avoiding router misconfiguration* and gaining greater flexibility of traffic flows. The so called *table-based forwarding* (*e..g OpenFlow API*) allows programming routers in different ways:

1. Centralized programming easier: compute tables centrally and distribute
2. Distributed programming more difficult: compute tables as a rsult of distributed algorithm implemented in each and every router

There are some difficult regarding the **traffic engineering** with a traditional router. For example, let's consider those scenarios and relatives implications

- *What if the network operator want* `u-to-z` *traffic to flow along* `uvwz` *rather than* `uxyz`*?* There is the need to re-define link weights so traffic routing algorithm computes routes accordingly: the tuning of the weight does not provide enough control on the routing itself. Refer the diagram in 9.3

- *what if network operator wants to split* `u-to-z` *traffic along* `uvwz` *and* `uxyz` *(load balancing)?* It's not possible or at lest require a new routing algorithm. Refer the diagram in 9.4.

- *what if w wants to route blue and red traffic differently from w to z?* Not possible. Refer the diagram in 9.5.

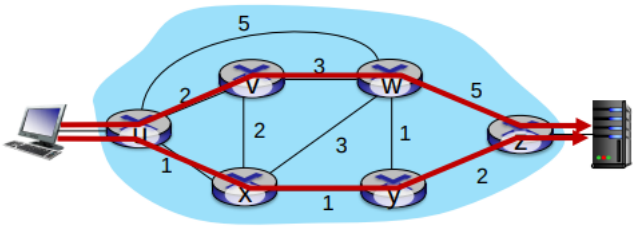

Figure 9.3: undefined undefined

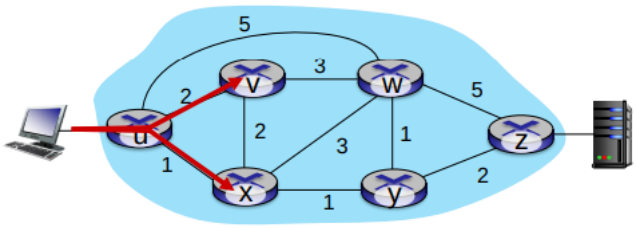

Figure 9.4: undefined undefined

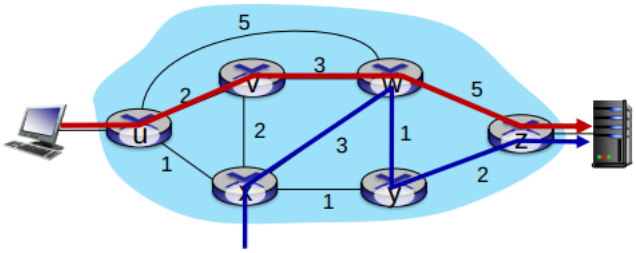

Figure 9.5: undefined undefined

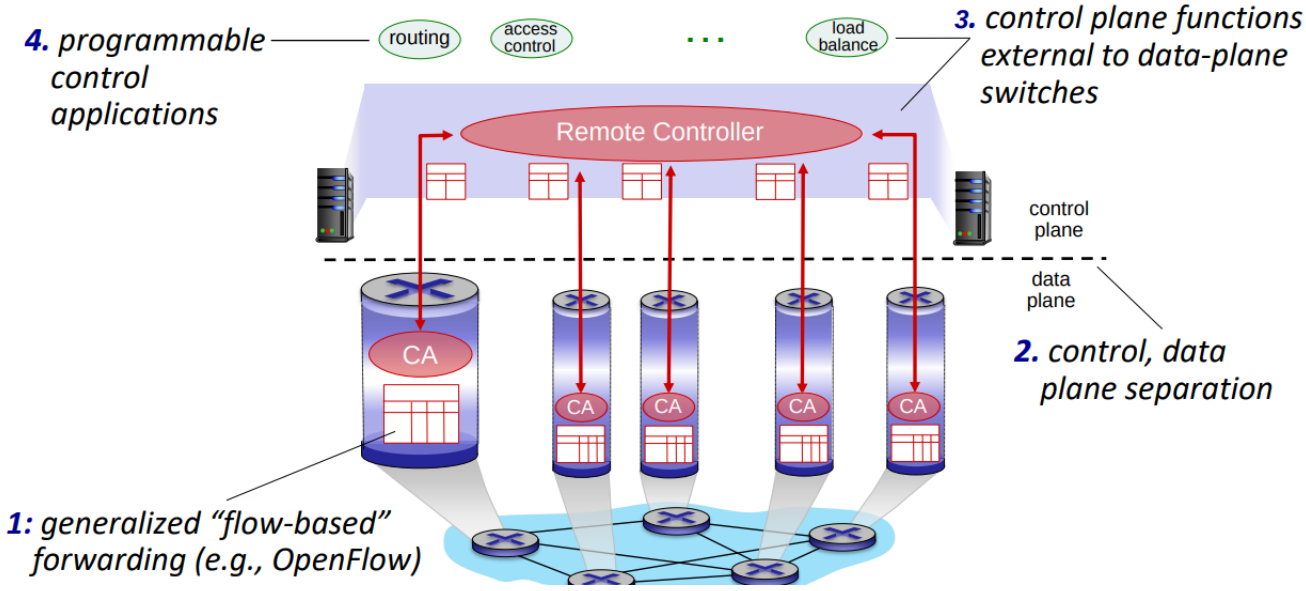

Figure 9.6: Overview of logically centralized control plane features

9.0.1 SDN Architecture

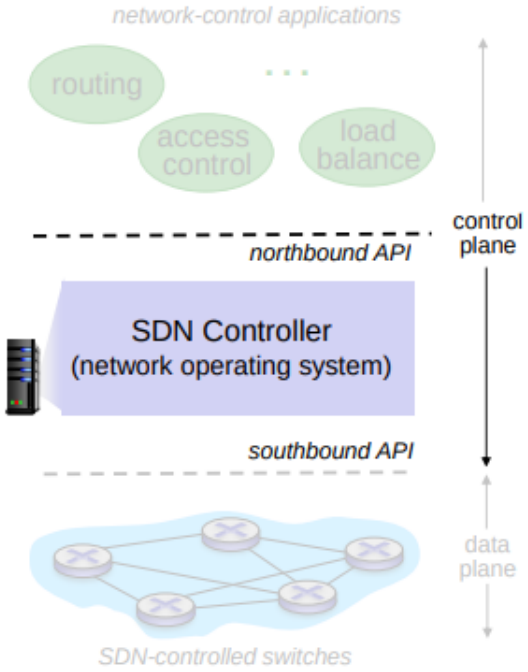

Figure 9.7: undefined undefined

The main three components are:

1. **Data plane switches**: they implement genralized data-plane forwarding in hardware. The forwarding table is already computed, installed under the supervision of the controller. They provide also API for table-based switch control (*like OpenFlow*) and provide a protocol for communicating with controller. They are *dummy* devices, without added complex and routing mechanisms to be computed.

2. **SDN Controller**: mantain the network state infromation and interact with the network control application above via the *northbound* API and the network switches via the *southbound API*. It's implemeted as distributed system to guarantee scalability, performance and fault-tolerance. It separates the control and the data plane (*as opposed to classical routing*): it represent the Network Operating System because perform routing decision based on different criteria.

3. **Network control apps**: they implement the control function (*like routing, access control and load balancing*) using lower-level services, API provided by the SDN Controller. This layer can also be *unbundled* so provided bi third-party supplier. Can have different routing vendor or SDN controller, customizable thanks to the northbound/southbound API exposure. Specifically, the northbound API provides an deeper abstraction respect to southbound API (*mainly OpenFlow*).

9.1 SDN Data Plane

The SDN data plane represent the resource or infrastructure layer made by forwarding devices: it allows to perform the transport and proessing of data according to decisions made by the SDN control plane. It perform those functions without embedding software, implementing an autonomous decision mechanism. The data plane can have virtual switches or physical switches, as sketched in 9.8.

The main function of the SDN data plane is to forward packets between network devices based on instructions provided by the SDN controller. The controller communicates with the data plane switches using a standardized protocol called OpenFlow, which allows the controller to program the switches to perform specific actions based on network conditions.

Some of the key functions of the SDN data plane include:

1. **Packet forwarding**: The data plane switches are responsible for forwarding packets between network devices based on the instructions provided by the controller. They also need to be able

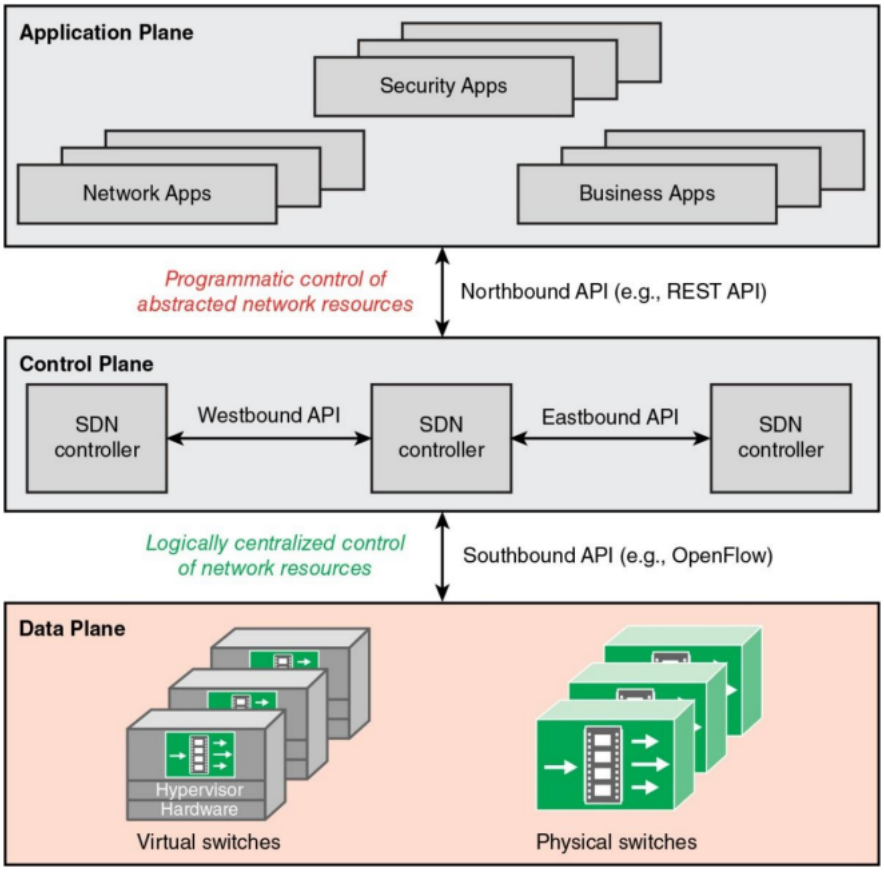

Figure 9.8: undefined undefined

to forward packets at high speeds, without causing bottlenecks or delays. It accepts incoming data flows from other network devices and end-systems: forwards them along the data forwarding paths computed and established according to the rules defined by the SDN applications. The *control support function* include the interaction with the SDN controller for management of forwarding rules (*the reference standard specification is mainly* ***OpenFlow Switch Protocol***).

2. **Traffic shaping**: The data plane can implement traffic shaping policies to manage the flow of traffic through the network. For example, it can prioritize certain types of traffic, such as voice or video, over other types of traffic to ensure that they are delivered without delay.

3. **Security**: The data plane can implement security policies to prevent unauthorized access to the network. For example, it can filter out packets that do not meet certain criteria, such as those coming from a known malicious source.

4. **Quality of Service (QoS)**: The data plane can implement QoS policies to ensure that certain types of traffic are given priority over others. For example, it can ensure that real-time applications like voice and video are given higher priority over other types of traffic.

5. **Load balancing**: The data plane can implement load balancing policies to distribute traffic across multiple network paths. This helps to prevent congestion and ensure that traffic is delivered efficiently.

The **Generalized forwarding** operated by the data plane consists in providing each router a **forwarding table (*flow table*)** that use the *"match plus abstraction"*: match bits in arriving packets and take the corresponding action. Two main forwarding techniques are used:

1. **Destination-based forwarding**: forward based mostly on destination IP address

2. **Generalized forwarding (*followed by OpenFlow*)**: many header fields (*e.g. MAC Address, destination port, etc*) can determine the action and many actions are possible like `drop/copy/log packet/modify`. It generalize both matching information and action to be taken.

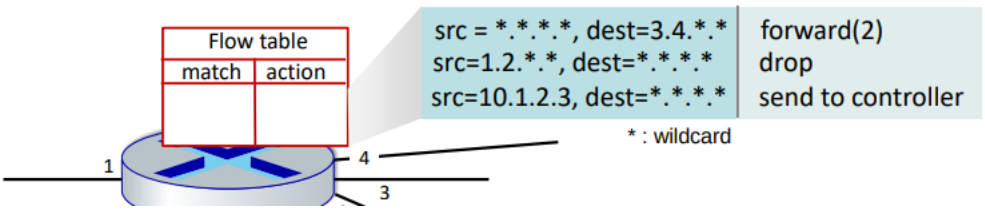

Figure 9.9: undefined undefined

The **flow** is defined by the header fields values (*respectively in link-,network-, transport-layer fields*): the generalized forwarding define simple packet handling rules (9.9):

- `match`: pattern values in packet header fields
- `actions`: for matched packet can `drop, forward, modify` or send matched packets to controller to decide the action to be performed.
- `priority`: disambiguate overlapping patterns (*flows can match more than one entry so needs disambigution*)
- `counters`: allows to count the number of bytes and packets, mainly for statistics. Those are helpful to identify patterns and derive rules based on traffic volumes and parameters, updating the controller knowledge on the network.

The packet can be sent to the controller only once: the first time a packet is send back to the switches with informations for future patterns (*e.g. as in the last picture*).

OpenFlow: Flow table entries

The following figure in 9.10 present the information that can be used at different layer to set the matching rules in the flow tables.

Examples As pictured in 9.11 and, as an example of switch implementation in 9.12.

We can also implement the behavior of a switch:

OpenFlow abstraction The *match plus action* abstraction allows to unifies different kinds of devices, based on their behavior:

- **Router**: the `match` is the longest IP prefix, the `action` forward out a link
- **Switch**: the `match` is the destination MAC address, the `action` forward or floow
- **Firewall**: the `match` is the IP address and the TCP/UDP port numbers, the `action` is permit or deny

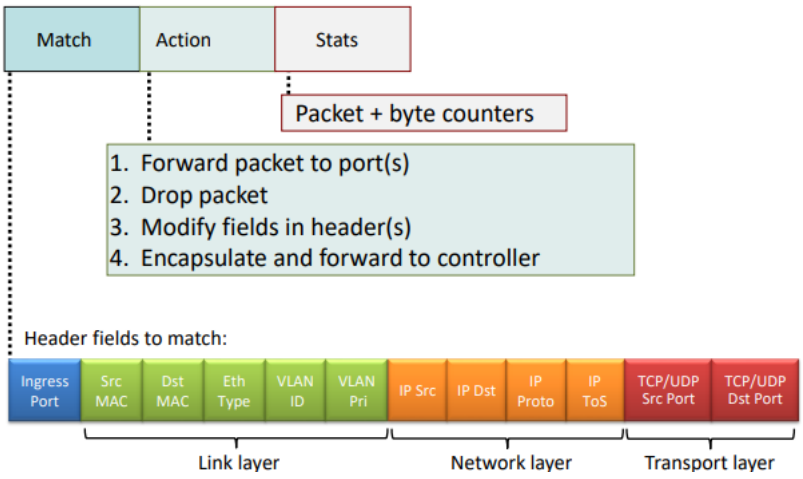

Figure 9.10: undefined undefined

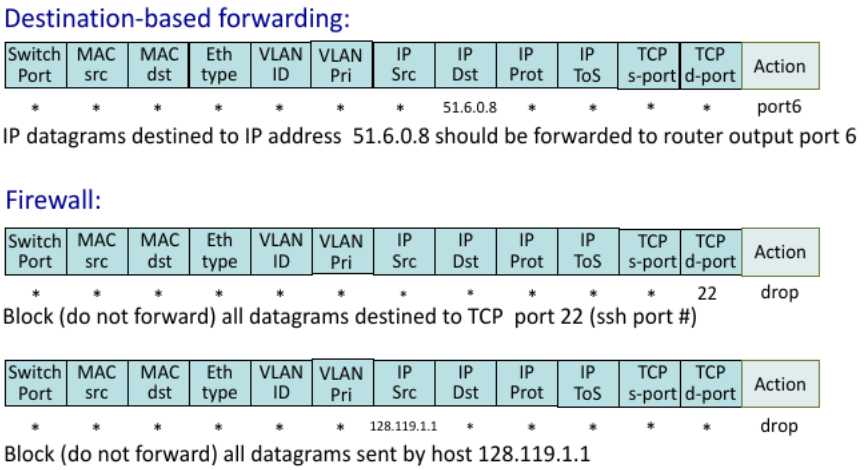

Figure 9.11: undefined undefined

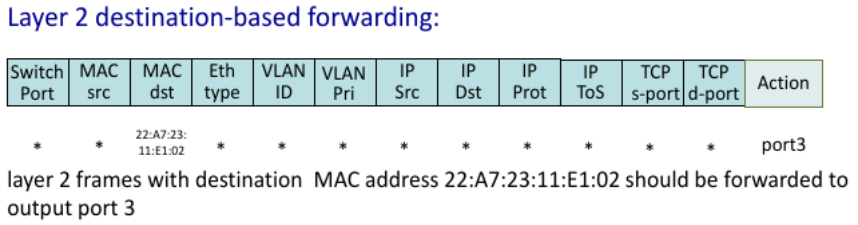

Figure 9.12: undefined undefined

- **NAT**: the `match` is IP address and port, the `action` is to substitute a private IP address with a public IP address. So the boundaries between switches and router's behaviour is blurring.

Example 2 The idea is that the OpenFlow controller have a complete overview of the network, providing the forwarding tables to the switches without them implementing and re-compute the algorithms itself. In the example picture in 9.13, the datagram from host `h5` and `h6` are destined to `h3` or `h4` and should be forwarded via `s1` and from there to `s2`. The routing tables are configured to forward the traffic by passing from `s1`.

OpenFlow Switch

In a OpenFlow switch there is more than a one flow tables, usually organized in pipeline as showed in 9.14. The group table specify a more complex behavior that refer to a group of flows, specifing rules for group of flows based on their attribute. Both those two type of table specify the data plane. The control channel allows to communcate to the control plane: typically can have a connection with multiple controllers as they are implemented in a distributed way. The simplest configuration is the master-slave with only one controller while the complex one can include active-passive controller redundancy.

Flow table pipeline The packet entering the switch are analyzed according to those mentioned tables as sketched in 9.15

The passage through different table accumulate the actions associated with a given match, executing them only as final step before the packet is sent out as stated in 9.16.

If there is a match on one or more entry in the table, the match is defined to be witht the highest priorioty matching entry. If there is a match only on a table-miss entry, the table entry may contain instructions as with any other entry. In practice, te tablemiss entry specifies one of three actions:

1. Send packet to controller: this will enable the controller to define anew flow for this and similar packet or decide to drop the packet

2. Direct packet to another flow table further down the pipeline

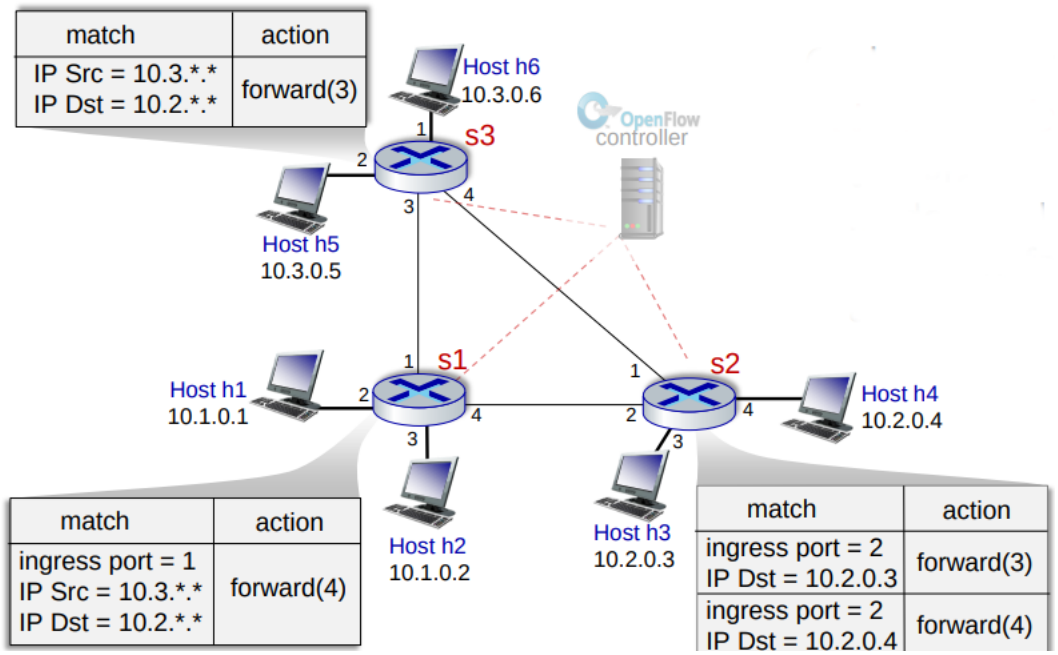

Figure 9.13: undefined undefined

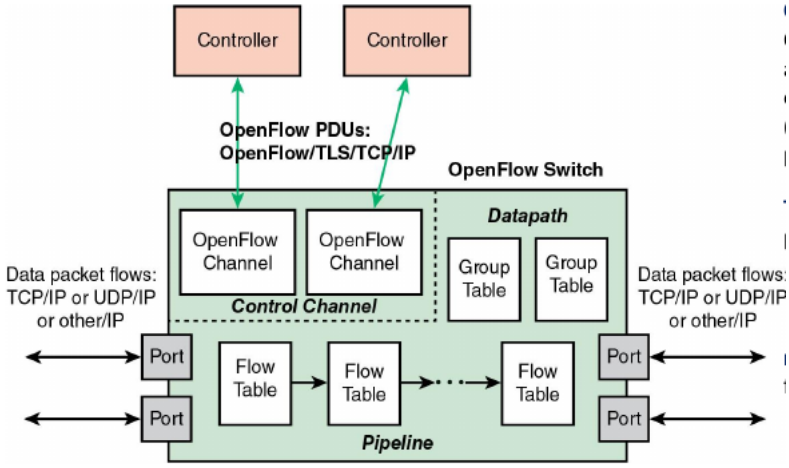

Figure 9.14: undefined undefined

3. Drop the packet If there is no match on any entry and there is no table-miss entry, the packet is dropped.

9.2 SDN Control plane

The general structure of the SDN Control plane is sketched in 9.19.

We may distinguish three layers:

1. **Communciation layer**: in the bottom one there is the OpenFlow implementation and the SNMP protocol.

2. **Network-wide state management**: the central part is the core of the controller. It collect and manage the infromation about the underluing infrastructure (*host information, how many switches, how many active ports are active*) and according to those information build the general network overview and extract statistics and manage dynamically flow tables.

3. **Interface latyer to network control apps**: In the upper layer there are components that handle the interaction with application like *network graph, RESTful API* to expose the network topology

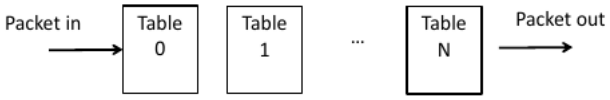

Figure 9.15: undefined undefined

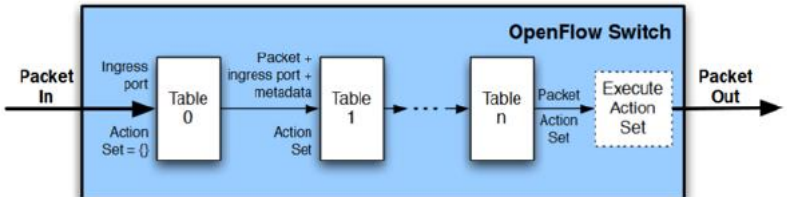

Figure 9.16: undefined undefined

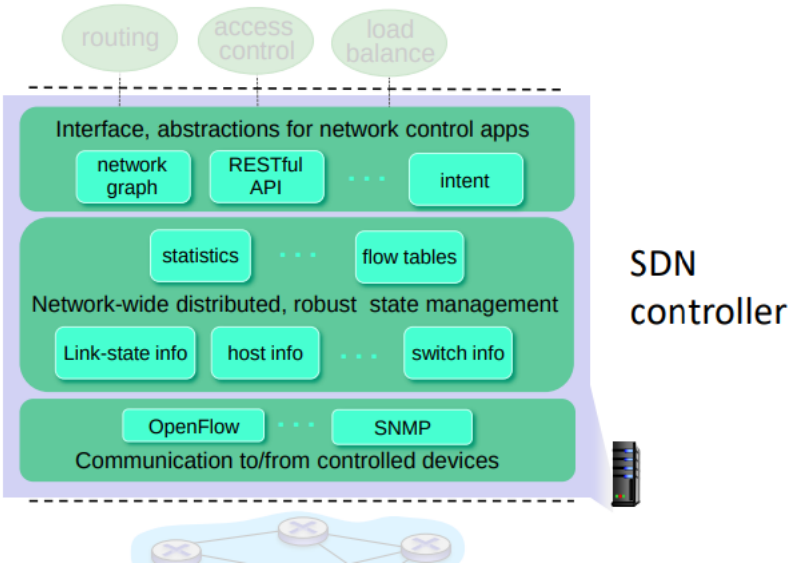

Figure 9.17: undefined undefined

to the application or *intent* that, for example, state at high level *"drop the malicious traffic pattern"* without specifying how: the SDN controller elaborates this intent and traduce this intent in a set of rules to be installated based on the current network topology.

OpenFlow protocol

It operates between controller and switches: use TCP to exchange messages (*suggested with TLS*). There are three classes of messages:

1. **Controller to switch**: These OpenFlow messages are initiated by the controller and sent to the switch to manage its behavior. Examples of controller to switch messages include packet-out messages to send packets to a specific port or set of ports, flow-mod messages to add, modify or delete flow entries in the switch's flow table, and barrier-request messages to synchronize the switch's processing with the controller.

2. **Asynchronous (Switch to controller)**: These OpenFlow messages are initiated by the switch and sent to the controller to notify it of events that occur on the switch. Examples of asynchronous messages include packet-in messages that inform the controller of packets that match a flow entry with the "send to controller" action, port-status messages that notify the controller of changes in the state of switch ports, and error messages that report errors encountered by the switch.

3. **Symmetric (both directions)**: These OpenFlow messages can be initiated by either the controller or the switch and are sent in both directions to maintain the state of the OpenFlow channel.

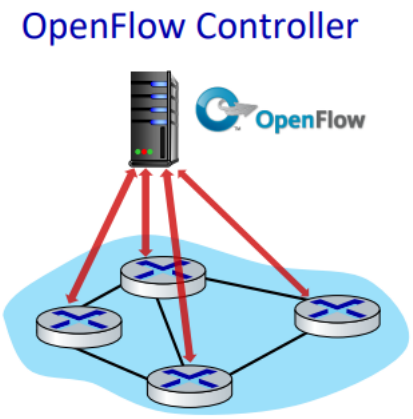

Figure 9.18: OpenFlow controller schema

Examples of symmetric messages include echo-request and echo-reply messages to test the connectivity and responsiveness of the channel, and features-request and features-reply messages to exchange information about the switch's capabilities and configuration.

The main *controller-to-switch* messages can be of the following type:

- `features`: controller queries switch features, switch replies
- `configure`: controller queries/sets switch configuration parameters
- `modify-state` (*FlowMod*): the controller ask the switch to add, modify, delete an entry in the routing rule table
- `packet-out`: controller can send this packet out of specific switch port. The controller instruct the switch to implement a specific action for a specific pattern.

The main *switch-to-controller* messages can be:

- `packet-in`: transfer packet to controller. For this packet the switch delegate the control to the controller.
- `flow-removed`: the switch inform the controller that has deleted an entry in its flow table
- `port-status`: inform the controller of a change on a port

Usually network oeprators don't *program* switches by creating/sending OpenFlow message directlu but use higher-level abstraction at controller.

SDN: Control/data plane interaction example The interaction between the SDN control plane and data plane consists of 6 steps, as pictured in 9.19.

1. S1 is experiencing link failure so uses OpenFlow `port-status` message to notify the controller
2. SDN Controller receive the message and updates link status info
3. Dijkstra's roting algorithm application has previosuly registered to be called whenever link status change so it's called
4. Dijkstra's routing algorithm access the network graph information, link state information in the controller and computes new routes
5. Link state routing app interacts with flow-table-computation component in the SDN controller, which omputes new flow tables needed
6. Controller uses Openflow to install new tables in swtiches that need updating

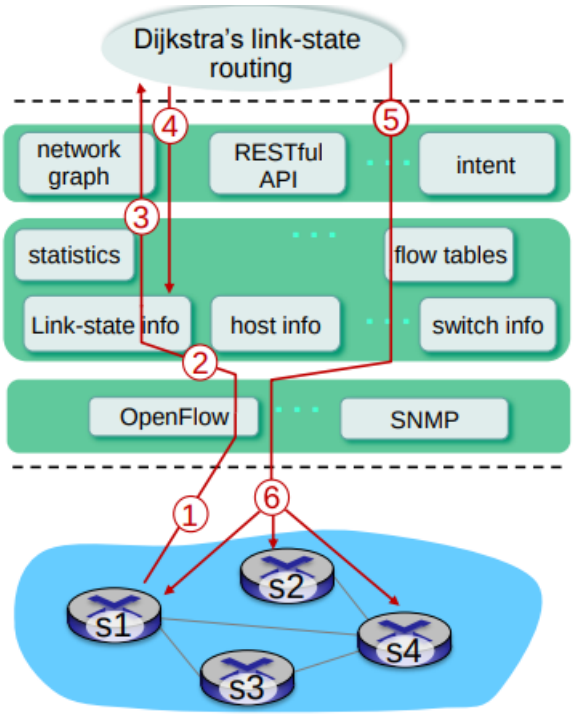

Figure 9.19: undefined undefined

9.3 Topology discovery and forwarding in SDNs

Traditonally the **routing** function is distributed among the routers in a netowrk: in an **SDN controlled network** it make sense to centralize the routing unction within the SDN controller. The controller can develop a *consistent view* of the network state for calculating shortest paths and can implement application-aware routing policies. **Data plane switches** are relieved of the processing and storage burden associated with routing, leading to improved performances because **centralized routing** application performs two different functions:

1. ***Link/Topology discovery***: routing function needs to be aware of the links between data plane switches. The topology discovery in OpenFlow is not fully standardized.

2. ***Topology manager***: mantains the topology information for the network, computing routes in the network like the shortest path between two data plane nodes or between a data plane node and a host.

The **Topology Discovery** is implemented by exploiting two major *initial configuration* features of Openflow (*OF*) switches:

1. every OF switch has initially set the IP address and TCP port of a controller to establisha connection as soon the device is turned on

2. switches have pre-defined behavior which support the implementation the support of the topology discover. It's supported by preinstalled flow rules to route directly to the controoler via a Packet-in message any message of the **Link Layer Discovery Protocol (LLDP)**.

9.3.1 LLDP - Link Layer Discovery Protocol

It's a **neighbor discovery protocol** of a single jump: it advertises its identity and capabilities and receive the same information from the adjacent switches. It's a vendor neutral protocol (*defined by IEEE*) ad operates at **layer 2** of OSI Model. It can be used also in traditional network and it's used by switches that support active configuration. In OpenFlow network, switches send the LLDP messages to discover the underluing topology by request the controller.

LLDP uses *Ethernet* as its transport protocol, thus the Ethernet type for LLDP Is `0x88cc`. An LLDP frame contains the following fields:

- **Chassis ID (Type 1)**: contains the identifier of the switch that sends the LLDP packet.

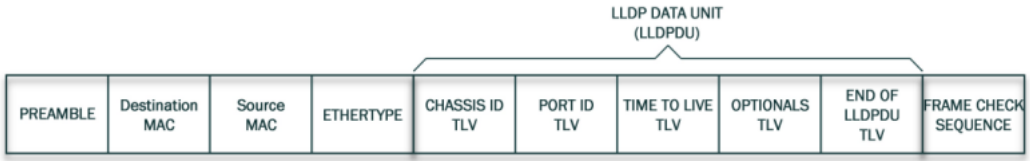

Figure 9.20: undefined undefined

- **Port ID (Type 2)**: contains the identifier of the port through which the LLDP packet is sent.
- **Time to Live (Type 3)**: time in seconds during which the information received in the LLDP packet is going to be valid.
- **End of LLDPDU (Type 4)**: indicates the end of the payload in the LLDP frame

Switch initialization

The initialization, for the pictured scenario in 9.21, implies the need of the controller to understand the **circular topology** of the switches.

When the switch is initialized:

- Establish a connection with the controller
- The controller send a message *FEATURE REQUEST MESSAGE* to the switch
- The switch responds with a message *FEATURE REPLY MESSAGE*
- It informs the controller of relevant parameter for the discovery of the links like the `Switch ID` and a list of `active ports` with their respective MAC associate, among other. At the end of the initial handshake, the controller knows the exact number of active ports on the OF switches **but** it doesn't know the physical connections between switches so it needs to build it by performing *Topology Discovery*

The **topology discovery** it's based on the information obtained from the initial handshake so the controller known the exact number of active ports:

1. The controller generates a **Packet-Out** message per active port on each switch discovered on the network and encapsulates a **LLDP packet** inside each generated message.
2. When an OF switch receives a LLDP message sent by the controller, it **forwards the message by the appropriate Port** ID included in the message to adjacent switches
3. Upon receiving the messages by a port that are not the controller port, **the adjacent switches encapsulate the packet within a Packet-In message addressed to the controller**. Metadata is included in the message such as `Switch ID`, `Port ID` where the LLDP packet is received, among others
4. When the packet comes back to the controller from an adjacent switch, the controller extracts this information, and then it knows a link exists between those switches

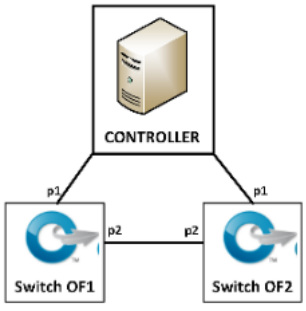

Figure 9.21: undefined undefined

The controller at the end known the `Switch ID` and `Port ID` in the LLDP message and `Switch ID` and `Port ID` in metadata (*from which the switch received the LLDP message*). This process is repeated for every OF switch on the network: the entire process of discovery is performed periodically. In a network of S switches interconnected by a set of links L, the total of packet-out message that the controller sends out to the network to discover all existing link between OF switches with P active port is:

$$Total_{packet-out} = \sum_{i=1}^{S} P_i$$

Let's clarify with an example pictured in 9.22:

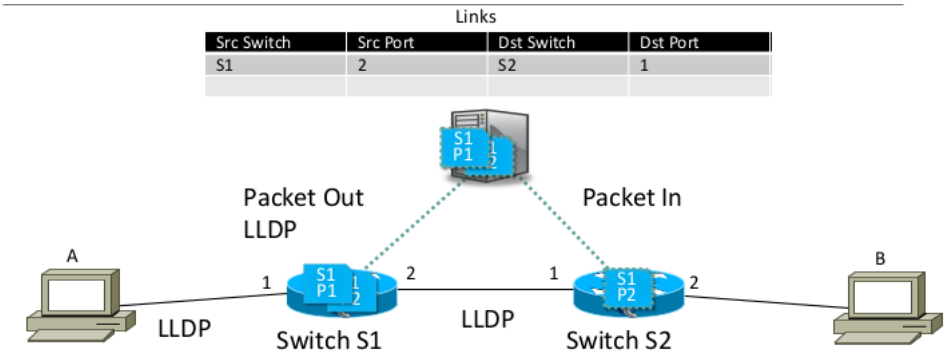

Figure 9.22: Topology Discovery Scenario

The controller send a **Packet-Out** to $S1$: the switch send the packet $S1-P1$ to host A which it's not an LLDP device so did not answer, while send the LLDP packet $S1-P2$ to switch $S2$. The switch $S2$ add its metadata information like `Switch ID, Port ID` and send the **Packet-In** message to the controller that now knows that $S1$ and $S2$ are adjacent because the packet sended by $S2$ contains the information that the initial LLDP packet was received from $S1$ via port 1_{S2} .

The topology discovery itself does not able both the controller and the switches to identify which hosts are reachable. The controller can learn that there is a **reachable host** in the network (*that is not an OF switch, so do not reply to OF packets*): the discovery is triggered by **unknown traffic entering** the controller's network domain either from an *attached host* or from a *neighboring router*. (*For unknown we mainly refer to not LLDP packets*).

For the **path computation**, initially flow tables on all switches are *empty*, but assume that Host table and Topology at thecontroller are fully populated with the necessary knowledge of the network and host. Refer the scenario in picture 9.23: when the switch receive a packet from the host `A` send the packet to the controller so it can compute the path and send the `Flow Mod` to update the flow table of the switches along the path from `A` to `B`. Based on the assumption that the controller already have all the kwnolesge about the host table and topology, the $S1$ and $S2$ flow table are updated as in figure.

These Flow Mod packets (*both from the controller to $S2$ and $S1$*) will be sent to each switch in **reverse order**, from the switch closest to the destination and so on until the source switch: the rationale behind this order is to **avoid the scenario in which multiple switches trigger the path computation to the controller** while there is the flow table updating operation ongoing by the controller.

All future packets from this flow will match directly at each switch and no longer need to take a trip up to the controller.

Exercise Describe the source, destination and semantics of the following OF messages:

- packet-in
- packet-out
- FlowMod:

9.3.2 Application of SDN

The SDNs's early adopters were companies that manages datacenter, for several use cases:

- Switching fabrics - SDN within data center: from black box to white box switches, server interconnected by software

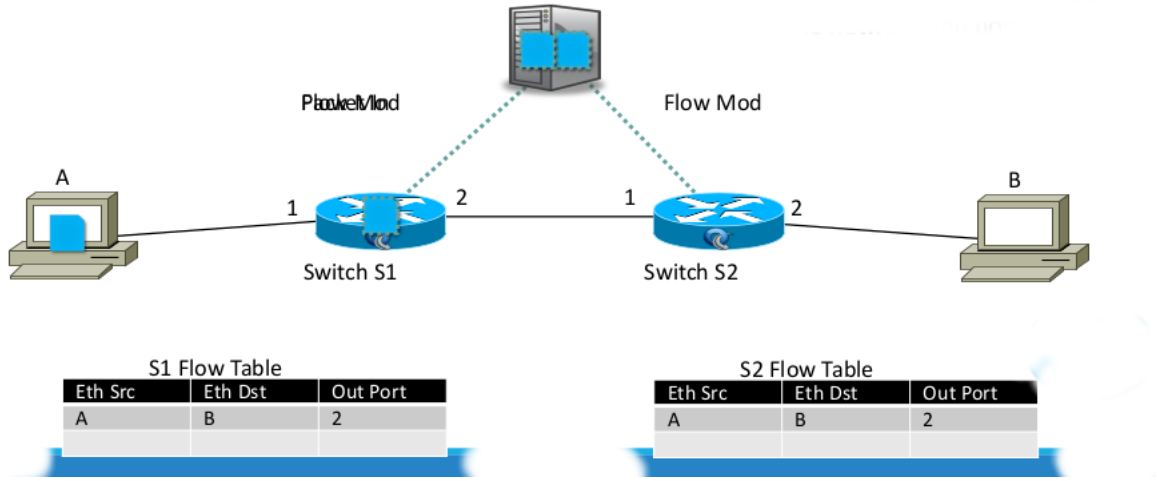

Figure 9.23: LLDP Path Computation

- Virtual networks - SDN to setup, manage and shutdown
- Wide area networks - as a traffic engineering methods, optimizing the traffic between different datacenters (*e.g. Google WAN for inter-datacenter traffic*).

Here we analyze the **B4 - Google Use case** of SDN inside Google Datacenters.

Google WAN

There are two separate backbones:

- **B2**: carries internet facing traffic. It have a growing traffic than the internet.
- **B4** 9.24: carries inter-datacenter traffic (*sort of horizontal traffic*). It have more traffic than B2 that also grows faster than B2

Google have its own globally deployed WAN, private and connected to all its datacenter. Data center are deployed across the world (as shown in 9.25) and need to ensure **multi-tenancy availability zone and redundancy** to guarantee availability and reliability of its service. The content are served based on geographic locality and replicated for fault tolerance among few dozens of datacenter around the world.

This network it's not on the public internet and must be cost-effective for high volume of traffic, performing load balancing among the DC.

Three main *type of flow types* are consired:

1. **User data copies** to remote data center for availability/reliability (*lowest volume, most latency intesnvie, highest priority*)

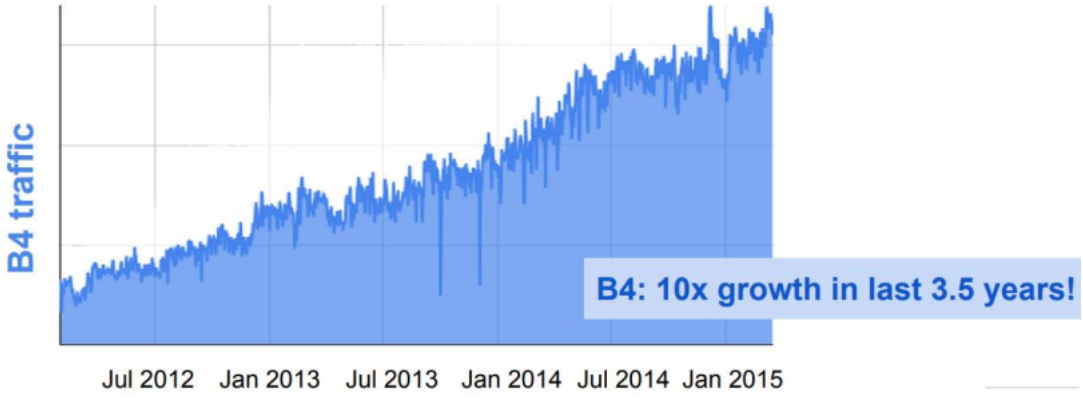

Figure 9.24: undefined undefined

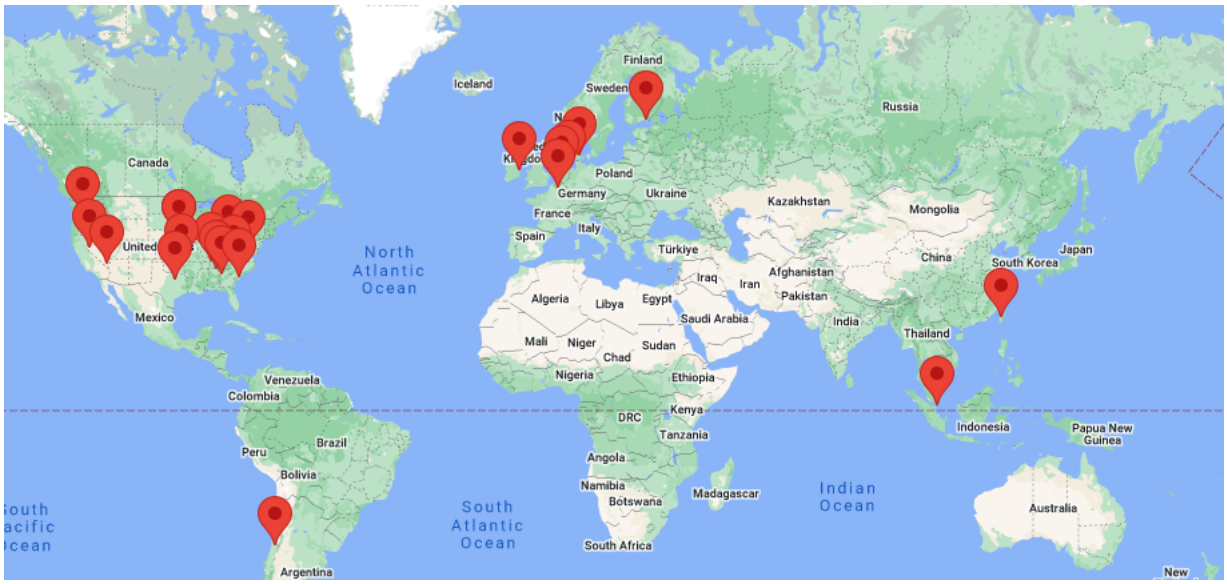

Figure 9.25: Google Datacenter Locations

2. **Remote storage access** for computation over distributed data sources
3. Large-Scale data push **syncrhonizing state across multiple data center** (*highest volume, least latency intensive, lower priority*)

The main problem in traditional WAN routing is that all the flows are treated with the same behavior (*so there is no QoS*) , all the links are used on average with 30% to 40% with some peaks, downgrading overall performance in a small timeframe.

The **backbone B4** has been built considering some requirements:

- *Elastic bandwidth demand*: applications can tolerate periodic failures or temporary bandwidth reductions
- *Moderate number of sites*: few dozens of sites, no need for large routing tables
- *End application protocol*: Google has the control of the application so can optimize the schedule of the data transfer, using a fine-grained application protocol. This avoid the need for link provsioning.
- *Cost sensivity*: private intercontinental links cost, so it needs to be used at its full capacity

The main *design decisions* applied on B4 backbone are related to:

1. **Customize HW and SW**: customize routing and monitoring protocols to requirements with a rapid custom protocol development
2. **Low cost switching hardware**: edge application control limits need for large buffers but the limited number of sites avoid the need to manage large forwarding tables
3. **Centralize traffic engineering (TE)** : allows more optimal and transfer traffic engineering than distributed control routing. It allows also to share bandwidth among competing applications. Usually the **TE server** is an application that works on top on the SDN controller.
4. **Increase link utilization**: allows an efficient use of expensive long haul transport. With **Traffic Engineering** usually we refer as a set of techniques to optimize and control traffic flows ina telecommunication network in order to ensure a maximum throughput and a sufficient QoS level.

B4 Architecture Overview

The **B4 architecture overview** is pictured in 9.26.

Each site have an *OFA - OpenFlow Agent* while the *site controller* it's based on a distributed *NCS - Network Controller System* to avoid a single point of failure. The *switch hardware* forwards traffic without using any complex control software and builded by using Google custom design with commodity

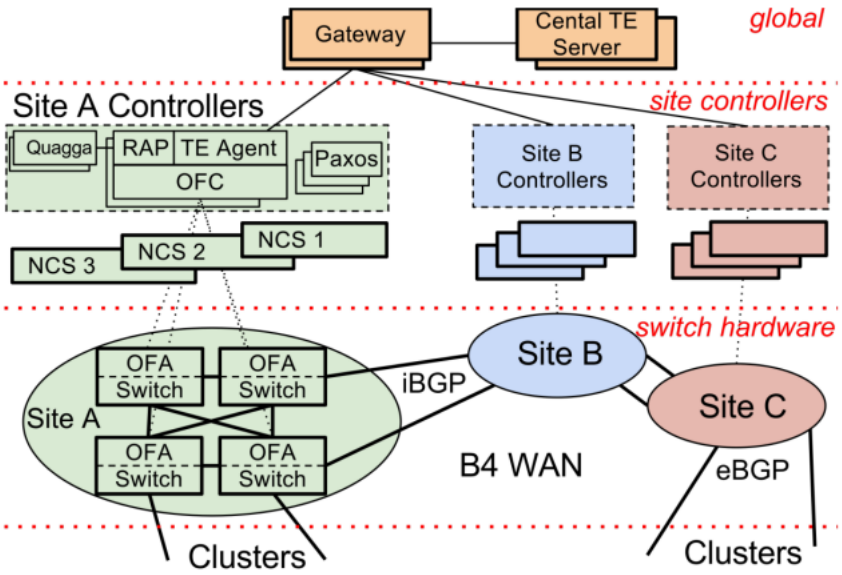

Figure 9.26: B4 Backbone General Architecture

sillicon.
The **control layer** is composed by *NCS* hosting both OpenFlow controllers (*OFC*) and Network Control Application *(NCAs)*: the OFCs mantain network state based on NCA directives and switches events but also instruct the switches to set forwarding table entries based on this changing network state.
The **global layer** is composed by a logically centralized applications like the *SDN Gateway* and a *Central TE Server*: the SDN gateway abstracts details of OpenFLow and switches hardware from the central TE server. Each B4 site consits of multiple switches with potentially hundreds of individual ports.

Each cluster contains a set of BGP router that peer with B4 switches at each WAN site: the rationale behind this choice was made due to the familiarity of the operator with the BGP protocol and mantain protocols retro-compatiblity, enabling also a gradual rollout.

The **traffic engineering** components in the *global layer* are pictured in 9.27:

The figure shows an overview of our **TE architecture**. The TE Server operates over the following state:

- The **Network Topology** graph represents sites as vertices and site to site connectivity as edges. The *SDN Gateway* consolidates topology events from multiple sites and individual switches to TE. TE aggregates trunks to compute site-site edges. This abstraction significantly reduces the size of the graph input to the *TE Optimization Algorithm*.
- **Flow Group (FG)**: For scalability, TE cannot operate at the granularity of individual applications. Therefore, we aggregate applications to a Flow Group defined as {`source site, dest site, QoS`} tuple.
- A **Tunnel (T)** represents a site-level path in the network, e.g., a sequence of sites (`A → B → C`). B4 implements tunnels using IP in IP encapsulation.
- A **Tunnel Group (TG)** maps FGs to a set of tunnels and corresponding weights. The weight specifies the fraction of FG traffic to be forwarded along each tunnel.

TE Server outputs the Tunnel Groups and, by reference, Tunnels and Flow Groups to the SDN Gateway. Thee Gateway forwards these Tunnels and Flow Groups to OFCs that in turn install them in switches using OpenFlow.
Another main component is the **Bandwidth Enforcer**: to capture relative priority, we associate a *bandwidth function* with every application, effectively a contract between an application and B4. This function specifies the bandwidth allocation to an application given the flow's *relative priority* on an arbitrary, dimensionless scale, which we call its **fair share**. We derive these functions from administrator-specified *static weights* (the slope of the function) specifying relative application priority. In the example pictured in 9.28, *App*1, *App*2, and *App*3 have weights $10, 1$, and 0.5, respectively. Bandwidth functions are configured, measured and provided to TE via Bandwidth Enforcer.

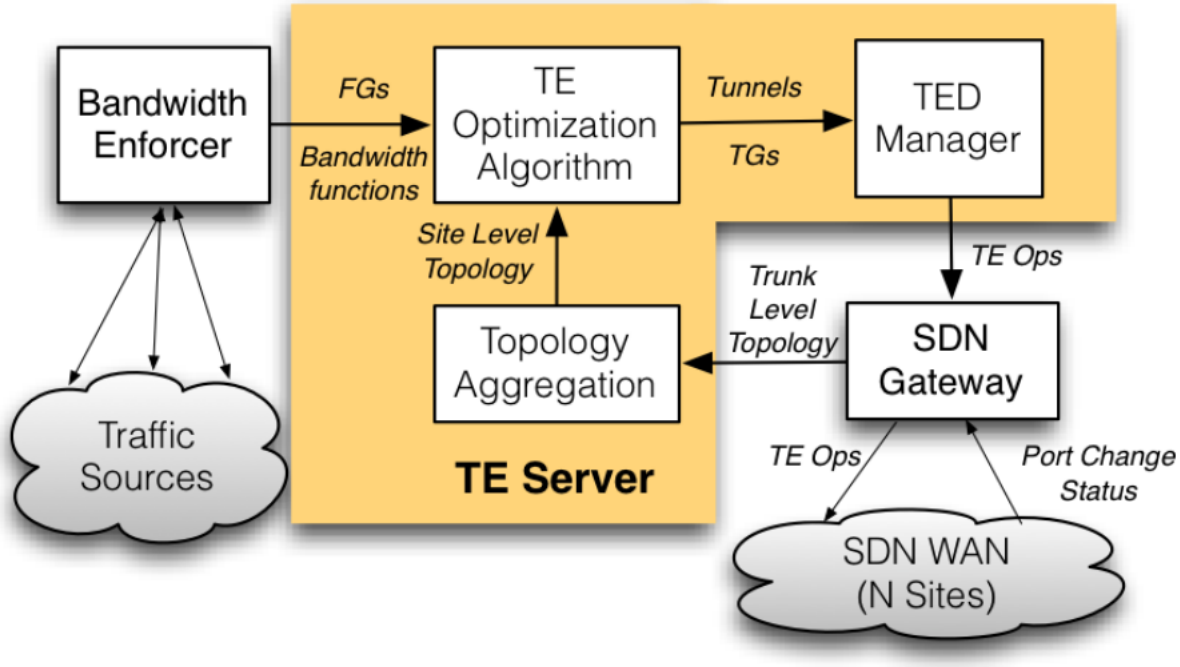

Figure 9.27: undefined undefined

Improvements results OpenFlow has helped Google improving B4 backbone (*referece 1*):

- Utilization near 100
- Mirror production events for testing on virtualized switches
- Reduced complexity
- Reduced costs With follow-up improvements (*referece 2*):
- hierarchical topology
- decentralized TE algorithms

  For further information about B4:

1. *Sushant Jain et al. "B4: experience with a globally-deployed software defined wan". In Proc. of the ACM SIGCOMM 2013 ACM, New York, NY, USA, 3-14*
2. *Hong CYY et al.,. B4 and after: managing hierarchy, partitioning, and asymmetry for availability and scale in google's software-defined WAN. In Proc. Proc. of the ACM SIGCOMM 2018 Aug 7 (pp. 74-87).*

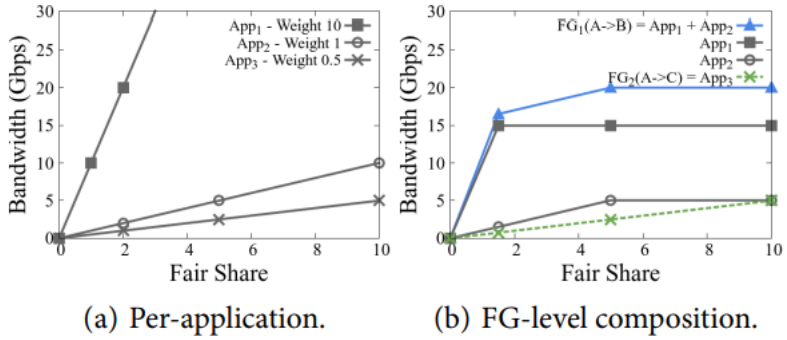

Figure 6: Example bandwidth functions.

Figure 9.28: undefined undefined

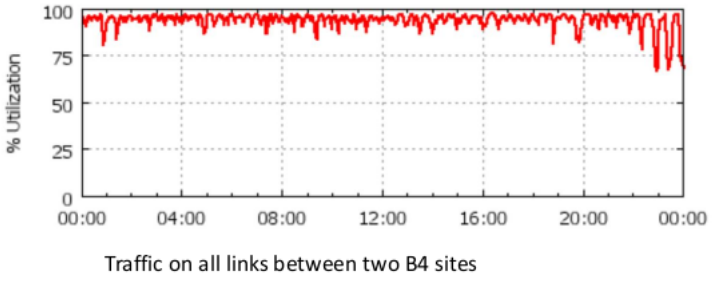

Figure 9.29: undefined undefined