1. What are the typical components of a node? sensor
   1. **Controller**   
      Processing of relevant data, executing arbitrary code)
   2. **Watchdog**:   
      Sensor controlled/time controlled/communication-controlled wakeup
      1. See how the IOT device handles an event while considering the WatchDog and the Controller (FLOW)
   3. **Wireless radio**

Short Range, low bit rate  
Types of Wireless Radio  
How does the controller control or access this component

* 1. **Sensor**Thermometer, cameras light, sensors, etc  
     What is the difference between a sensor and a sensor node
  2. **Power Supply**  
     batter, solar, vibration   
     The IoT device should be very energy efficient

1. **Which component of a sensor node have a major influence on its autonomous lifetime?**Wireless Radio: Data Exchange requires a lot of energy

Power supply: Bigger supply -> Longer operation time

Are there any other components that have a major influence on its autonomous life time

1. **What are the key differences between a wireless ad-hoc network and a conventional wired network?**

**Wireless Network are divided in two branches**

**1. IBCN  
2. M-AdHoc Network**

Wireless ad-hoc networks do not rely on a physical connection while wired networks do.

Wireless ad-hoc networks are decentralized where devices act both as router and end node.

Wireless ad-hoc network nodes can move freely within the network range

Wireless ad-hoc networks are less reliable because of the moving nodes

**Wireless Ad-hoc Network**

Physical Infrastructure: Wireless  
Topology: Decentralized/mesh  
Connectivity: Mobile  
Reliability: Less reliable

**Conventional Wired Network**

Physical Infrastructure: Cables or wires  
Topology: Centralized  
Connectivity: Stationary  
Reliability: More Reliable

**Aida**Wireless Ad-hoc Network:

Does not rely on pre-existing infrastructure such as routers or access points.

Each node can act as both a router and an end device, forming an infrastructure-less network.

The network topology is dynamic and can change frequently as nodes move, join, or leave the network.

Nodes must configure themselves and manage the routing of data.

The network can reconfigure itself in response to changes in the position of nodes.

Nodes are often battery-powered and require energy-efficient protocols to extend their operational life.

Energy constraints impact the design of routing and communication protocols.

Limited by the wireless transmission range of individual nodes.

Multi-hop communication is used to extend the effective range by relaying data through intermediate nodes.

Conventional Wired Network:

Relies on a fixed infrastructure with routers, switches, and cables to manage communication.

Has a static topology with fixed pathways for data transmission.

The network layout is usually planned and remains constant unless manually reconfigured.

Limited mobility due to physical cable connections.

Changes in the network layout require manual intervention and physical reconfiguration.

Scalability is constrained by the physical infrastructure and capacity of network devices.

Adding new nodes often requires additional hardware and reconfiguration.

Typically connected to a constant power source, so energy efficiency is less of a concern.

Network devices are powered through the wired connection.

Communication range is limited only by the length of cables and the reach of the network infrastructure.

Generally provides more stable and higher bandwidth connections

**Can the base station be wirelessly connected too unlike Tower**

1. **How does the power of a radio signal at the location of a transmitter relate to the power of the radio signal the location of the receiver?**The power must be equal to the number of hop times the distance to each device consideration of the signal attenuation

A screenshot of a computer

Description automatically generated

If there is a part in the answer which for example is causing limitations, should we also provide a valid justification of how that limitation can be solved

1. **Outline why multi-hop wireless communication is more energy- efficient than single -hop wireless communication**The Single Hop communication only uses two device and thus need more energy the further away the receiver is. The multi-hop always required less energy because the signal is transmitted by a chain of transmissions. The advantage can be calculated by Nα−1  
     
   Aida  
     
   A paper with writing on it

   Description automatically generated

(Rec )The larger the number of intermediate nodes, the less energy is required to forward a packet towards the destination. The denser the network, the less energy is needed to forward a packet to the destination.

(Chat + slide) Multi-hop wireless communication is more energy-efficient than single-hop wireless communication primarily due to the way signal strength decreases with distance. Here are the key points outlined in the provided slides:

Path Loss and Power Consumption:

In single-hop communication, the signal needs to cover a longer distance directly from the transmitter to the receiver. The power required for this transmission increases significantly with distance due to the path loss exponent (𝛼), which typically ranges between 2 and 5 depending on the environment.

For a single-hop transmission over a distance Nr, the power required (Psend(Nr)) can be approximated as (Nr)α⋅Preceive, where Preceive is the minimum receiving power required to maintain a certain transmission error rate.

Energy Efficiency of Multi-Hop Communication:

Multi-hop communication divides the overall distance into smaller segments (hops). Each hop covers a shorter distance rrr, requiring significantly less transmission power for each individual hop.

The power advantage ηrf of an N-hop transmission compared to a single-hop transmission is given by:

This equation shows that as the number of hops NNN increases, the power required per hop decreases, leading to overall energy savings.

Summary:

The reduction in transmission power per hop and the effective use of shorter communication distances in multi-hop communication lead to considerable energy savings.

This makes multi-hop communication more suitable for wireless sensor networks where energy efficiency is critical due to the limited power resources available in sensor nodes.  
A close-up of a page

Description automatically generated  
  
  
**Personal**

The more intermediate nodes there are in a network, the less energy is required to forward a packet to its destination. In other words, as the network becomes denser, the energy needed for packet transmission decreases. Therefore, multi-hop communication, which involves multiple intermediate nodes, requires less overall energy compared to single-hop communication when transferring data.

1. **What is meant by the term “Data centricity” in relation to sensor networks**Its not about the identity of the node but what data it can provide is the concept of data centricity. Neither sender nor receiver needs to know about their partner. The answer is not necessarily triggered directly by a question.

Data Centricity in relation to sensor networks refers to the focus on data itself rather than on the individual sensor nodes generating the data. This approach prioritizes the information being collected, processed, and queried over the specifics of which sensor node is providing the data. Here are the key points highlighted in the provided slides:

User Interest in Data:

Users are primarily interested in the data being generated, such as specific events or measurements, rather than the identity or location of the sensor nodes themselves.

Example: Searching for "car detection" rather than querying a specific sensor node ID.

Networking Based on Content:

Communication and networking within sensor networks can be structured around the content of the data rather than the nodes. This means routing decisions and data exchanges are made based on the nature of the data being processed.

Decoupling in Space and Time:

Decoupling in Space: Neither the sender nor the receiver needs to know each other's specific identity or location. Data can be published and subscribed to based on content.

Decoupling in Time: The communication can be asynchronous. An "answer" to a data query does not need to be directly and immediately triggered by the "question." This allows for more flexible and resilient data handling.

1. **What is the typical networking interaction paradigm in wireless sensor networks? Describe this concept!  
   Personal**   
   The typical networking interaction paradigm in wireless sensor networks is the publish/subscribe paradigm. In this model, publishers send data to a central software bus, which stores subscriptions from various subscribers. When the data changes, the software bus notifies the relevant subscribers based on their subscriptions, enabling efficient and decoupled communication between publishers and subscribers.  
   **A diagram of software bus

   Description automatically generated**
2. **What is meant by the term “Partial State Record”**A partial state record refers to the intermediate result of an aggregation process, where essential statistics such as the sum and count of previously aggregated values are maintained. This record is crucial for computing aggregates like averages, as it tracks the ongoing summation and the number of values included.   
   **The behaviour of partial state records varies**: they can be distributive, maintaining the same size as the final aggregate   
   *(e.g., MIN, MAX, SUM);*   
   algebraic, having constant size but not being full aggregates themselves   
   *(e.g., AVERAGE);*  
   holistic, requiring the retention of all measured values   
   *(e.g., MEDIAN);*   
   or content-sensitive, proportional to some data properties   
   *(e.g., HISTOGRAM).*   
   Aggregation functions with distributive and algebraic partial states are suitable for in-network aggregation, whereas content-sensitive and holistic functions present challenges for practical aggregation.

In the context of wireless sensor networks, a partial state record refers to an intermediate representation of the state of an aggregated value during in-network processing. This concept is crucial for efficiently computing aggregated values like averages, sums, or other statistical measures across distributed sensor nodes.

Purpose of Partial State Records:

Partial state records are used to store intermediate results of aggregation operations as data is collected and processed by sensor nodes.

These records enable the computation of final aggregated values without requiring all raw data to be transmitted to a central location, thus saving energy and bandwidth.

Types of Partial State Records:

Distributive: The size of the partial state is the same as that for the final aggregate (e.g., MIN, MAX, SUM).

Algebraic: The partial states are not themselves aggregates for the dataset but are of constant size (e.g., AVERAGE).

Holistic: The partial state needs to reflect all measured values (e.g., MEDIAN).

Content-sensitive: The partial state is proportional to some (statistical) properties of the data (e.g., HISTOGRAM).

Behavior and Use:

Distributive and algebraic partial states are well-suited for in-network aggregation because they can be efficiently computed and combined.

Holistic and content-sensitive aggregate functions may not be as practical for in-network aggregation due to their complexity and the amount of data they may req

1. **What are typical properties of aggregation function in relation to the unreliable character of wireless communication?**Aggregation functions  
   Given two partial state records <x> and <y> and aggregation function f computes a new state record <z> = f(<x>,<y>).

1. Duplicate sensitive aggregates are affected if the measured values of a node is used in computation more than once  
2. Exemplary aggregates return one or more representative values from the set of all values – MIN, MAX  
3. Summary aggregates compute some properties over all values

1. **Why is the sequential programming model unsuitable for programming sensor node?**In the case of sequential programming model, the sensor is being polled for available data**.** Also, the transceiver is being polled for any packets arrived or packets that must be transmitted. There are couple of risks with this programming model.  
   1. We can miss some data from the sensors while a packet is received or processed  
   2. We can miss a packet when the sensor information is being processed  
     
   Draw this in the exam   
   A diagram of a process flow

   Description automatically generated
2. **Why is the process-based programming model unsuitable for programming sensor nodes?**In case of process-based programming model, we have two separate processes or threads;  
   One for handling sensor.  
   Second is for the packets  
   This produces too much overhead because the following reasons:  
   1. A lot of memory i required, since each process requires its own stack space in the memory  
   2. The context switch between the processes has a high rate.  
   Draw this in the exam  
   A diagram of a process

   Description automatically generated
3. **Which programming model is suitable for sensor? Describe this programming paradigm!**The programming model suitable for sensor nodes is the event-based programming, the reason, in this case, a sensor node waits for any event to happen. This can be, for example, a timeout event, available data from a sensor, or an arrival of a packet. Once the event has occurred, it is then handled by a short sequence of instruction.  
     
   **Some Deep explanation**:  
   How it works  
    1. Types of Events:  
    Timeout Events: Triggered after a specified time interval  
    Sensor data availability: New data from a sensor becomes available  
    Packet arrival: A data packet arrives from another node  
    2. Event Handling:   
   i.When an event occurs, the sensor node transitions from idle or regular   
    processing to an event-specific handler  
    ii. Sensor Event Handler: Activated when a sensor-related event occurs,   
    such as new data availability.  
    iii. Radio Event Handler: Activated when a communication-related event   
    occurs, like the arrival of a data packet.  
    3. Instruction Execution:   
    i. Upon activation by an event, the node executes a short sequence of  
    instructions tailored to handle that specific event
4. **TinyOS applications are written in nesC. nesC uses two types of components. What functions do “ modules have and what functions do “Configurations” have in nesC application?**The nesC implementation consists of two parts:  
    A PowerupC modul and a PowerupApp C configuration  
    The module PowerupC contains the executable logic of the application  
   The module resides in the file Powerup.nc  
      
    **Modul Powerup C**{  
    uses interface Boot;  
    uses interface Leds;

Uses interface GPS;  
 } **//**This Module declares that it uses two interfaces “Boot” and “Leds” **implementation**{  
 event void Boot.booted(){  
 call Leds.led0On();  
 }

Event Void GPS.ON(){  
 }

}  
   
 //It implements an event handler for the Boot.boot event. When the system   
 boots, this event is triggered   
 //Inside this event handler, it calls the “LedOn” function from the Leds interface,   
 which turns on LED0  
  
 **Configuration** PowerupAppC {  
   
 }  
  
 **implementation{  
 components MainC, LedsC, PowerupC;  
 PowerupC.Boot -> MainC.Boot;  
 PowerupC.Leds -> LedsC.Leds;  
 }**  
 //This lists the components used in the application “MainC”, “LedsC”, and   
 “PoweupC”  
 So basically, it wires the interfaces together:  
 1. PowerupC.Boot is connected to “MainC.Boot” meaning PowerupC will   
 handle the boot event from MainC  
 2. PowerupC.Leds is connected to LedsC.Leds, meaning PowerupC will   
 use the LED functions provided by “LedsC”  
  
  
**Initialization:**components MainC, LedsC, PowerupC; initializes the components but does not specify their interactions.   
**Wiring:**PowerupC.Boot -> MainC.Boot; and PowerupC.Leds -> LedsC.Leds; connect the interfaces, defining how events and commands are handled.

**Execution:**The Boot.booted event triggered by MainC is handled by PowerupC, which then calls Leds.led0On() in LedsC.  
  
  
**Steps:**  
 1. MainC gets initialized  
 2. MainC Triggers the Boot Event  
 3. PowerupC handles the BootEvent  
 4. LED Turned On

**[TinyOS Startup]**

**|**

**v**

**[MainC Initialization]**

**|**

**v**

**[MainC triggers Boot.booted event]**

**|**

**v**

**[PowerupC handles Boot.booted event]**

**|**

**v**

**[PowerupC calls Leds.led0On()]**

**|**

**V  
[LedsC turns on LED 0]  
  
  
  
  
A screenshot of a computer screen

Description automatically generated**

**Modules** are the components that contain the executable logic of the application, they define how the application behaves by implementing the interfaces. They either use or provide interfaces, which means they either require or offers certain functionality of the other components  
  
**Configurations** are the components that wire other components together. They define how different modules and other configurations are connected to form complete application. It specifies which components specify which components are used and how their interfaces are connected

Writing the code in the exam would give us points or?

1. **What does the term “wiring” mean in relation to nesC application?**The term “wiring” means binding an interface user to an interface provider. In other words, configuration consists of a signature and an implementation block. The implementation block can bind an interface user to a provider using  
    the -> or <- operator  
    This process is called wiring  
   We achieve and promote the modularity and reusability mechanism  
   -> operator connects a user of an interface to a provider  
   <- operator can also be used but is less common. It reverses the direction of the wiring  
     
   Example is given in question 13
2. **The functional relationship between different nesC components is defined by interfaces. Which functions are implemented by interface providers and which functions are implemented by interface users?**  
   Personal  
   In nesC programming, the functional relationship between components is defined by interfaces, where interface users and providers have distinct roles.   
     
   **Interface users** can call commands provided by the interface and must implement the events signaled by the providers. On the other hand, interface providers are responsible for implementing the commands that users can call and can signal events that the users must handle.   
     
   This clear division of responsibilities ensures structured and modular interaction between different components in nesC, enhancing the overall functionality and maintainability of the system.
3. **Many TinyOS interface are “Split phase”. What does this term mean?**Split phase a design pattern,   
   This means that the user of the interface sends a command  
   The provider of the interface performs the associated operation  
   The provider signals an event that indicates the completion of the operations  
   Draw this in the exam  
   A diagram of a user interface

   Description automatically generated  
     
   Nonblocking concepts
4. **The execution model of TinyOS uses “tasks”. Which special functions do these tasks have?**Tasks are very particular functions; they are an important building block for the execution model. Tasks look like C functions, but:  
    1. They can’t have a return value  
    2. They don’t take any parameters  
    3. They can be executed only within the naming scope of a component  
   4. Tasks schedule a function to be called later by postponing it using the POST   
    keyword  
    5. Task do not interrupt one another  
    6. Tasks can be seen as time-flexible background jobs
5. **Describe the execution model implemented by TinyOS! How does Tiny OS’s schedular work?** 1. **Split-Phase Interface**  
    Operations like sending packets or sampling sensor are handled in two   
    phases:  
    1. Command initiation: A non-blocking command starts the   
    operations and immediately returns.  
    2. Interrupt Handling: The hardware component issues an   
    interrupt when the operations is complete  
    3. An event handler processes the completion and can post a   
    new tasks  
    2. **Execution Model:**  
    1. Energy saving: The mote stays asleep most of the time to save   
    energy  
    2. Computation Start: Computation is initiated by the hardware   
    interrupts. (The post operations adds tasks to an internal task   
    queue)  
    3. **Schedular Operations:**  
    1. Tasks Queue Processing: The scheduler processes tasks in   
    First In, First Out (FIFO)  
    Interrupts can interrupt tasks and post additional tasks  
     
   A white text on a black background

   Description automatically generated  
   A white text on a white background

   Description automatically generated

**Chapter 3  
Personal**

1. **Describe the 5-Layer model of the IOT**  
   There is no single consensus on architecture for IoT, but we can have a starting point.
   1. **Client/External Communication Layer**
      1. This layer handles the interaction between the IoT system and the external entities. Provides web protocols, dashboards, and user interfaces / web interface to communicate with the other IoT systems. So the other IoT system are able the services of this IoT
   2. **Event Processing and analytics layer**
      1. As each IoT devices generate a lot of data, so this layer manages and processes the vast amounts of data or events generated by IoT devices. An example of event can be a sensor reading, the data gets generated very fast, that why we need this layer, or we lose the value of the data. Functions of this layer are that it uses techniques to analyse data in real-time, employing pattern matching algorithms to detect significant events (e.g., identifying dangerous individuals at an airport, a reactive system with a sensor node for temperature in a building for fire detection.)
   3. **Aggregation/Bus layer**
      1. Facilitates communication between IoT devices and protocols. Functions of this layer is to translate between different application protocols, such as HTTTP CoAP and MQTT. It provides security processes like authorization and verification to ensure data stream integrity for a specific data is accessed from specific data source. Furthermore, we have a RESTful interface in this layer so other systems can interact with this device and process data vice versa.
   4. **Communication Layer**
      1. Deals with the protocols and methods for data transmission between devices. Function of this layer is to support various communication method. For example: Radio modules, Bluetooth, long-range modules) and includes protocols like HTTP, MQTT and CoAP to enable efficient data exchange
   5. **Device layer**
      1. Comprises the actual IoT devices, which can include various sensors and actuators such as thermostat. Function of this layer is to act as an embedded web server, accessible through HTTP polling or more efficient method like WebSockets for real-time, bidirectional communication
   6. Additional Management blocks: Device, Identity, and access management. The following components are suggested to be used in all of the functional layers
      1. **Device Management:** Deploys software, manages version control, automates updates, monitor device state and availability, and can block or wipe compromised devices
      2. **Identity and Access Management**: Implements authentication and authorization to ensure secure data transfer and function access between devices

A screenshot of a computer

Description automatically generated

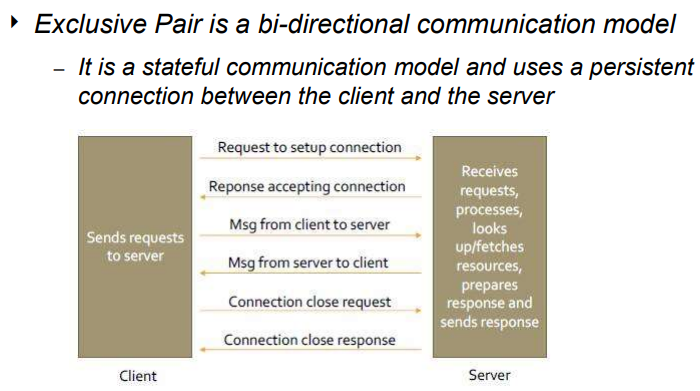
1. **What are the characteristics of the “request-response” communication model?**Its a stateless communication model. Each request-response pair is independent of each other   
   A diagram of a server

   Description automatically generated  
   Clients sends a request to a sever, and the server responds to the request, now when the server receive the request, it decided how to respond fetches the required data from the database, prepares the response in an appropriate represents like in JSON and sends the response to the client. As this a stateless mode of communication so, so all the requests are independent to each other  
   This communication is good in the context when clients are making of requests from time to time.  
   Example: HTTP
2. **What are the characteristics of the “publish-subscribe” communication model**A diagram of a subscriber

   Description automatically generated

The Publish-Subscribe communication model involves three main instances: **publishers, consumers, and brokers**, it is suitable in event driven applications  
Publishers act as sources of data, sending messages to specific sever called the broker, the broker handles the routing distributions of the message, to the various subscribers/consumers.   
Consumers subscribe to these topics to receive relevant data.   
Brokers manage the topics and facilitate the forwarding of messages from publishers to the appropriate consumers.   
This model decouples the producers and consumers of data, enhancing scalability and flexibility. For example, a publisher can send messages to topics such as Topic 1 and Topic 2, and brokers ensure these messages are delivered to all consumers subscribed to those topics, allowing Consumer 1, Consumer 2, and Consumer 3 to receive only the data they are interested in.  
  
Example MQTT

1. **What are the characteristics of the “push-pull” communication model?**A diagram of a company

   Description automatically generatedThe "Push-Pull" communication model uses queues to decouple messaging between producers and consumers. Producers send messages to queues, which act as buffers to handle inconsistent data rates between producers and consumers. This decoupling allows producers and consumers to operate independently, with queues managing the flow of messages to ensure consumers can process them at their own pace. This model enhances flexibility and scalability, as it allows the system to handle varying speeds of message production and consumption effectively.  
     
   This is used to have consistency with data rate between Publisher and Consumers
2. **What are the characteristics of the “exclusive pair” communication model?**The "Exclusive Pair" communication model is a bi-directional and stateful communication method that maintains a persistent connection between a client and a server. This model allows both the client and server to continuously exchange messages over the same connection until it is explicitly closed. It begins with the client requesting to set up the connection, followed by the server's response accepting it. Messages are then sent back and forth, and the connection is closed upon request and confirmation from both sides. This approach ensures efficient and continuous communication without the need to repeatedly establish new connections.
3. **What is the purpose of the 6LoWPAN network layer protocol?**The purpose of the 6LoWPAN network layer protocol is to enable IPv6 to be used with wireless embedded devices. Since Zigbee does not natively support Internet protocols, 6LoWPAN allows these devices to communicate using the standard Internet Protocol, thereby facilitating the integration of wireless embedded devices into IP-based networks.   
     
   Here are some key features and function that this network layer protocol provides
   1. **Enables IPv6 for low devices:** as low power wireless networks like using IEEE 802.15.4 (Zigbee), this doesn’t typically support the full suite of Internet protocols, that’s why using 6LoWPAN network layer protocol provides a framework that allows IPv6 packets to be sent and received over IEEE 802.15.4 based networks.
   2. **Efficient Packet Transmission:** The challenge is that IPv6 headers are large (40 bytes), which can be too big for the small frame sizes (127 bytes) of IEE 802.15.4 networks, that’s why using 6L0WPAN includes header compression techniques that significantly reduces the size of IPv6 packets. For example, it can compress the IPv6 header and UDP header to just 7 bytes, making them suitable for transmission over low-power wireless networkit also supports the mesh networking
4. **How does the network protocol COAP work in principle?**COAP is a stateless request/response protocol. However, a separate protocol extension is used to allow observation of resources. Like in the case of MQTT, a client can subscribe to a resource. It uses UDP (User Datagram Protocol) instead of TCP, which helps in reducing overhead and maintaining efficiency. CoAP supports the standard HTTP methods (GET, POST, PUT, DELETE) for resource manipulation.
5. **How does the network protocol MQTT work in principle**MQTT works on the publish-subscribe communication model where a client connects to a broker, which acknowledges the connection and keeps it alive as long as possible. The client subscribes to specific topics on the broker, and the broker manages message distribution to all subscribed clients. The MQTT protocol allows clients to send regular ping requests to maintain the connection and uses "last will and testament" messages to notify others when a client disconnects. Implemented via TLS for security, MQTT brokers can request authentication details like username and password. This model negates the need for constant polling, enhancing efficiency.
6. **What role do "topics" have in the MQTT protocol?**A topic is a UTF-8 string, which is used by the broker to filter messages. Subscribers are often interested in a great number of topics. Individual subscribing to each topic would be cumbersome (Wild-cards)
7. **What role do the symbols "\*" and "#" have in relation to MQTT topics?**In MQTT topics, the symbols "+" and "#" serve as wildcards to simplify subscription to multiple topic levels. The "+" symbol is a single-level wildcard that matches one topic level. For example, "building/+/sensors" can match "building/floor-1/sensors" and "building/floor-2/sensors". The "#" symbol is a multi-level wildcard that matches multiple topic levels, such as "building/floor-1/#", which can match all sub-topics under "building/floor-1", like "building/floor-1/sensors" and "building/floor-1/blinds". These wildcards make it easier to subscribe to a broad range of topics without specifying each one individually.  
     
   A screenshot of a computer

   Description automatically generated
8. **What advantages and disadvantages does MQTT have over COAP?**  Advantages:
   1. MQTT has a highly decoupled publisher and subscriber model, whereas CoAP has an asynchronous communication model.
   2. MQTT allows 16 different types of messages, whereas, CoAP allows only 4 types.
   3. MQTT offers three levels of Quality of Service (QoS) for message delivery (at most once, at least once, exactly once), ensuring reliable communication as per application requirements.   
        
      Disadvantages:
   4. MQTT uses TCP connections, which require keeping the connection alive for longer periods. This increases power consumption, making it less suitable for battery-operated IoT devices compared to CoAP, which uses the more lightweight UDP.
   5. MQTT packets are generally larger than CoAP packets, which can lead to higher overhead and reduced efficiency
9. **Which architectural principles are implemented by REST-based APIs?**   
   REST-based APIs implement key architectural principles including **uniform interfaces** for consistent resource access, **stateless** communication where each request contains all necessary information, and **catchability** to enhance efficiency by allowing response caching. The **client-server model** separates the user interface from data storage and processing, enabling independent development and scalability. These principles collectively ensure simplicity, scalability, and performance in web services.
10. **What special features does the WebSocket communication protocol have?**
    1. **Allow bi-directional, full-duplex communication between clients and servers**
       1. Exclusive pair communication model
    2. **Does not require a new communication setup for each message to be sent**
       1. Unlike response/request APIs such as REST
    3. **Suitable for applications with low latency / high throughput requirements**
       1. No overhead for connection setup
       2. There is no HTTP header information included in each message  
          A white background with black text

          Description automatically generated
11. **Are there IoT use cases where a WebSocket-based API is better suited than a REST-based API? Describe an example!**  
    OLD  
    Yes, a Shipment Monitoring IoT system is a great example of a use case, where a  
    WebSocket-based API is more suitable, since in this case we need a bidirectional persistent connection between the client and the servers.  
      
    NEW  
    Yes, there are IoT use cases where a WebSocket-based API is better suited than a REST-based API. For example, a Shipment Monitoring IoT system benefits significantly from a WebSocket-based API because it requires a bidirectional, persistent connection between clients and servers. This system needs real-time updates on conditions like temperature, humidity, and location of the shipment, which can be efficiently handled by WebSockets. Unlike REST, which involves overhead for each request-response cycle, WebSockets maintain an open connection, allowing for low-latency, high-throughput communication without the need for constant reconnection, making it ideal for continuous monitoring and immediate alerts in shipment tracking.
12. **What functions does the ThingSpeak platform provide for IoT applications?**  
      
    Real-time data collection and analysis
    1. Visualizing the collected data in the form of charts
    2. Ability to create apps for collaborating with web services
13. **Django´s core structure can be broken into three parts: Models, Views and Templates. Describe the function of these three components!**   
     **NEW**  
    Models.  
    Models serve as the data layer in Django. They define the structure of the data in the database by representing what the objects in the database will look like. For  
    example:  
    **from django.db import models**  
    **class Student**(models.Model):  
     first\_name = models.CharField(max\_length=30)  
     last\_name = models.CharField(max\_length=30)  
     age = models.IntegerField()
    1. In this example,
       1. **First\_name, last\_name, and age** are fields in each Student object.
       2. **Django provides easy-to-use validation** methods such as CharField, IntegerField, and others, allowing model fields to accept only certain types of inputs
    2. Templates:   
       A template is an HTML page used to present data. It includes:
       1. The static parts of the desired HTML output.
       2. Special syntax for inserting dynamic content.
    3. Views:   
       Views act as the bridge between models and templates. They:
       1. Tie the model to the template.
       2. Contain the logic that generates the web pages.
       3. Determine what data to display.
       4. Retrieve data from the database.
       5. Pass the data to the template for rendering.

**OLD**  
A screenshot of a computer program

Description automatically generated

Templates.

A template is simply an HTML page. A template contains:

1. the static parts of the desired HTML output
2. some special syntax describing how dynamic content will be inserted

Views.

A white background with black text

Description automatically generated



Questions for the Prof.

1. For some question, with theory if we create a diagram, can we cover points from some missing things that were not written in our answer (Like missing keywords)

2. How much should we prepare from the Lab ex.