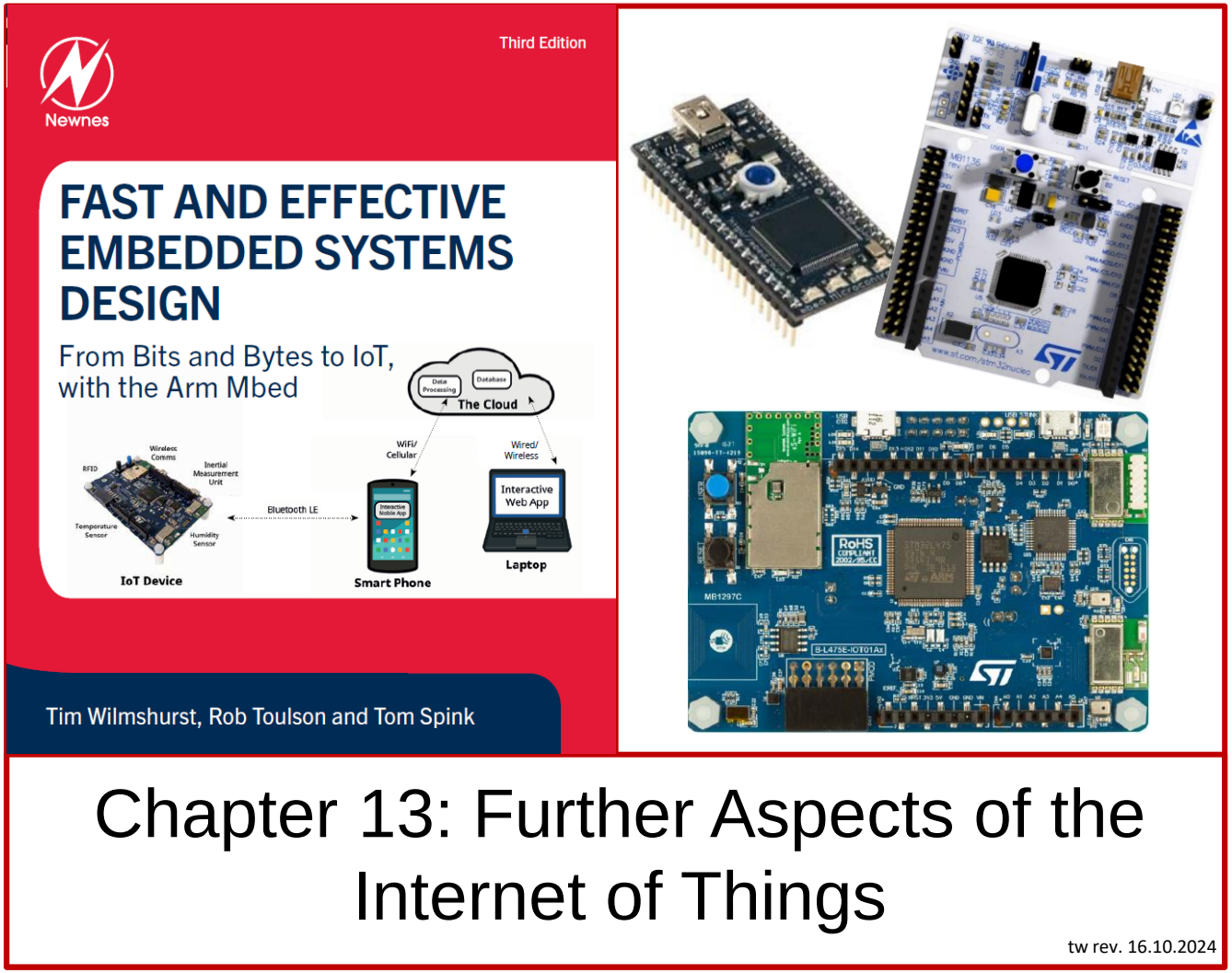


Chapter 13: Further Aspects of the Internet of Things

tw rev. 16.10.2024

If you use or reference these slides or the associated textbook, please cite the original authors' work as follows:

Wilmshurst, T., Spink, T. & Toulson, R. (2024). Fast and Effective Embedded Systems Design – From Bits and Bytes to IoT, with the ARM Mbed (3rd edition), Newnes, Oxford, ISBN: 978-0323951975.

www.embedded-knowhow.co.uk

Ethernet: Introduction and Frames

Ethernet is a serial protocol designed for network communications, particularly on Local and Wide Area Networks (LANs and WANs).

Ethernet communications are defined by the IEEE (Institute of Electrical and Electronics Engineers) 802.3 standard, and support fast data rates up to 400 Gbps (Gigabits per second).

On the physical layer, Ethernet uses differential send (TX) and receive (RX) signals, resulting in 4 wires labelled RX+, RX-, TX+ and TX-.

Ethernet messages are communicated as serial data packets called *frames*. Each frame includes a unique source and destination 6-byte MAC (Media Access Control) address. The frame is wrapped within a set of *preamble* and *start of frame* (SOF) bytes and a *frame check sequence* (FCS) which enables devices on the network to understand the function of each communicated data element. The standard 802.3 Ethernet frame is constructed as shown.

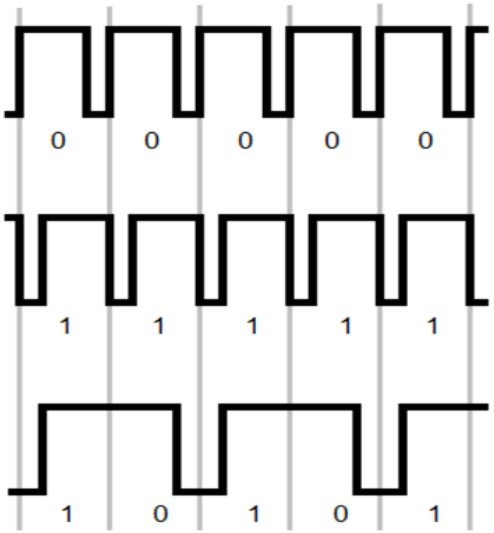
The minimum Ethernet frame is 72 bytes. However the preamble, SOF and FCS are often discarded once a frame has been successfully received. So a 72 byte message “on the wire” is often reported as having just 60 bytes.

Preamble	Start of frame delimiter	Destination MAC address	Source MAC Address	Length	Data	Frame Check Sequence	Interframe gap
7 bytes of 10101010	1 byte of 10101011	6 bytes	6 bytes	2 bytes	46 – 1500 bytes	4 bytes	

Ethernet: Manchester Coding

The Ethernet protocol uses the *Manchester encoding* method, which relies on the direction of the edge transition within the timing window, as shown. This shows first a string of zeros being transmitted, then a string of ones, then alternate ones and zeros. If the edge transition within the timing frame is high-to-low, the coded bit is a 0; if the transition is low-to-high then the bit is a 1.

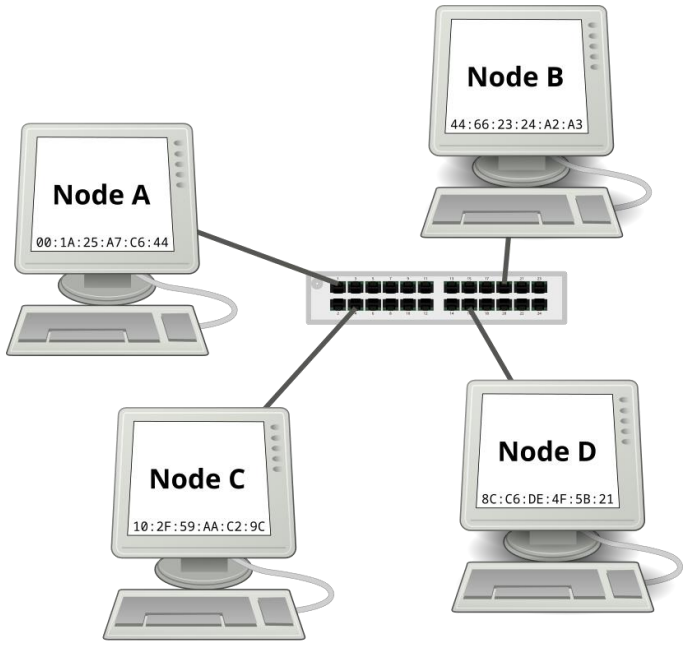
The Manchester protocol is very simple to implement in integrated circuit hardware and, as there is always a switch from 0 to 1 or 1 to 0 for every data value, the clock signal is effectively embedded within the data. As also shown in the Figure, even when a stream of zeros or ones is transmitted, the signal still shows transitions between high and low.



Ethernet: Linking Nodes

To build an Ethernet network, devices or “nodes” are connected with each other through either a network hub or switch. This creates a star topology, as shown. Every node in the network has its unique MAC address, which allows it to be individually addressed, and therefore communicated with.

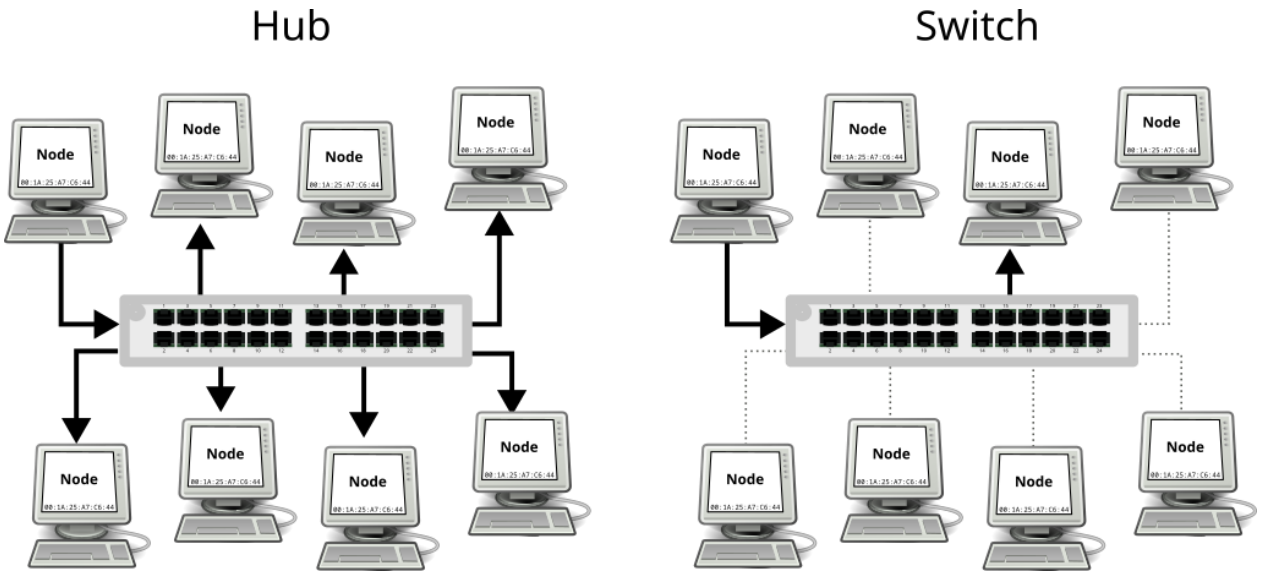
When a data packet is being sent between nodes, the sending node encodes its own MAC address in the “Source MAC Address” field of the frame, and the intended recipient of the packet in the “Destination MAC Address”. If the packet is intended for a group of nodes, a special broadcast MAC address is used.



Ethernet: Hubs and Switches

A switch is a more advanced networking device, which uses the node's MAC addresses to forward data packets to their intended destinations. The switch examines the destination MAC address in a packet header and forwards the packet only to the node with the matching address. This results in better use of the network bandwidth and therefore less network traffic.

The Figure shows how this arrangement looks. The hub broadcasts a packet to all connected devices. A switch forwards a packet only on the port for the intended device.



Wi-Fi: Introduction

Wi-Fi is a wireless networking technology that allows devices to connect to a network and with each other. It operates on the 2.4 GHz and 5 GHz radio frequency bands, and applies the IEEE 802.11 standard.

Wi-Fi networks can be set up and configured in different ways, depending on the requirements of the user. A basic setup can be created with a single access point that connects to a wired network and broadcasts a wireless signal. More complex setups, such as those in large buildings or public spaces, use multiple wireless access points to provide coverage.

As in Ethernet, Wi-Fi data is transmitted and received in frames. The two main types of frame are management frames, and data frames.

The two main types of Wi-fi device are *access points* and *stations*.



Wi-Fi: Management and Data Frames

Management frames manage the network, and include:

- **Beacon frames:** transmitted periodically by an access point to advertise its presence and provide information about the network.
- **Probe request frames:** sent by a device looking for available networks to connect to.
- **Probe response frames:** sent by an access point in response to a probe request to provide information about the network.
- **Association request frames:** sent by a device to request access to a network.
- **Association response frames:** sent by an access point to grant or deny access to a network.

Data frames are used to transmit data between devices. Some examples are:

- **Data frames:** contain the payload of the data being transmitted.
- **Control frames:** used for flow control and error correction.
- **Acknowledgment frames:** sent by a device to confirm receipt of a data frame.
- **Request-to-Send/Clear-to-Send frames:** manage access to the wireless medium to avoid collisions.

Wi-Fi: Device Types, Making Connections

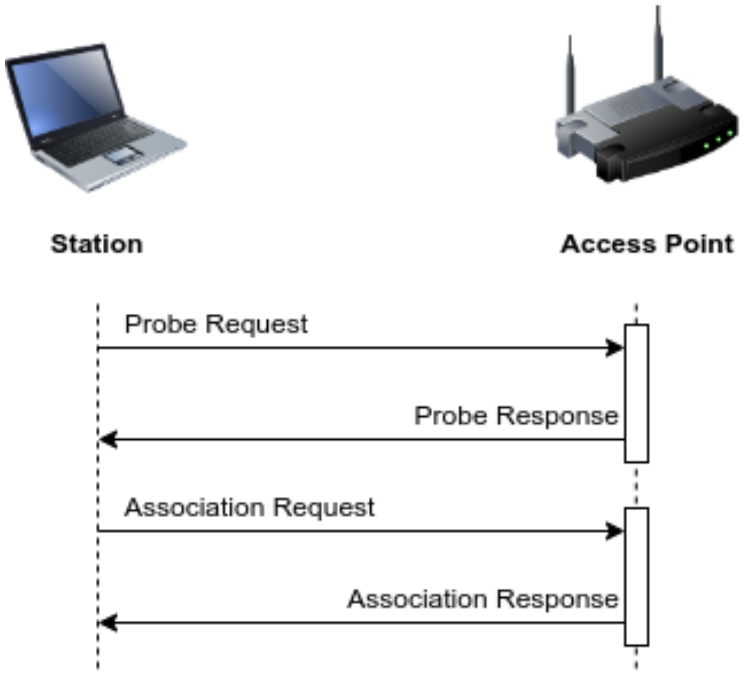
The two main types of Wi-fi device are access points and stations.

An access point (AP) connects wireless devices to a wired network. It acts as a central hub for wireless devices and allows them to communicate with each other and with wired devices on the network – like a network switch/hub. They typically have a wired connection to a switch, which provides connectivity to the rest of the wired network and beyond.

A station (STA) connects to a wireless network through an access point. A station could be a laptop, smartphone, tablet, or other device that has wireless capability. Stations use the wireless network to communicate with other devices on the network, as well as with devices on the Internet or other networks.

When a station wants to connect to a network, it sends a probe request to the access point. This responds with a probe response, providing information such as the network name (also known as the service set identifier, or SSID), the type of security used, and supported data rates. The station then decides whether to connect to the network.

If the station decides to connect, it sends an association request. The access point responds with an association response, either accepting or rejecting the connection based on the network's security settings, the number of connected devices, and other factors.



Wi-Fi: Frames

All 802.11 frames contain several fields in their headers. Some fields are unused for some frame types. The following fields are present in the header:

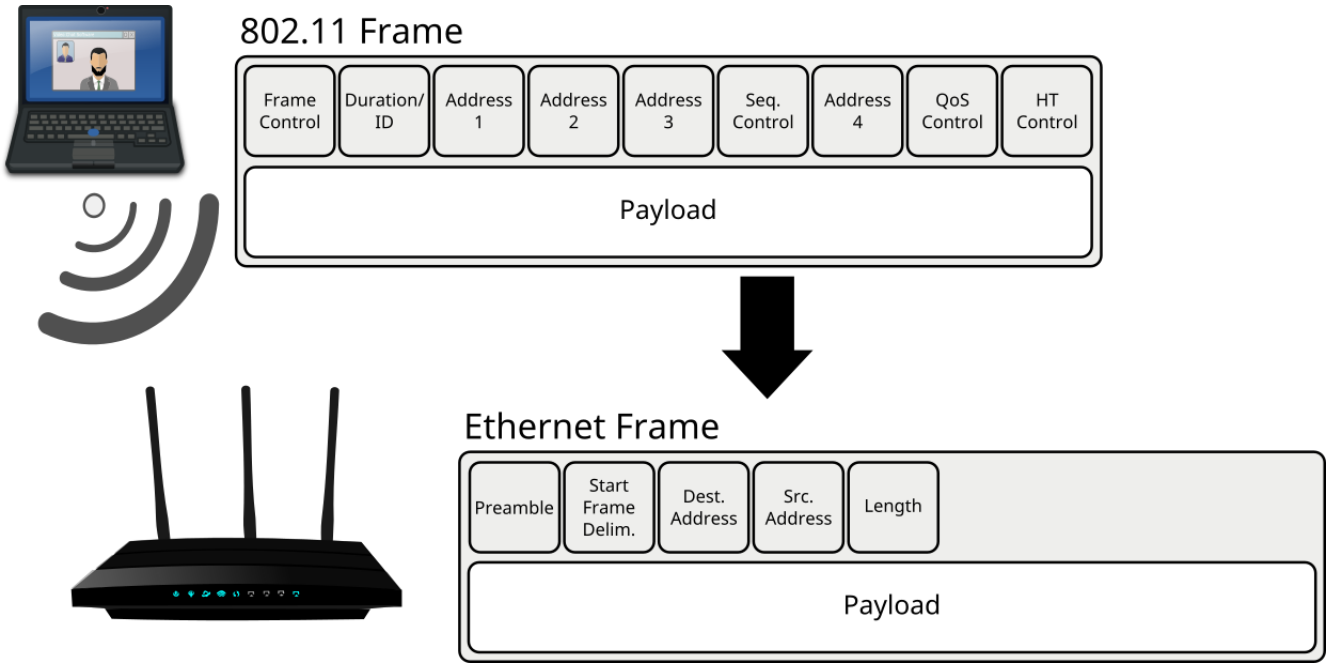
- **Frame Control:** contains information about the type of frame e.g. various control flags, such as whether the frame is a data frame, the type of data, and whether the frame is protected by encryption.
- **Duration/ID:** This field contains the duration of the transmission in microseconds or an identification number for the transmission.
- **Address fields:** These fields contain the source and destination MAC addresses, as well as the address of the device that the frame will be sent to next.
- **Sequence Control:** This field contains a sequence number for the frame and a fragment number, used for fragmentation and reassembly of large data packets.
- **Quality of Service (QoS):** This field contains information about the priority and other characteristics of the data being transmitted.
- **High-Throughput (HT) Control:** This field contains control data for facilitating fast data rates, and is specified by the 802.11n variant of the protocol.

Following the header is the frame body, which contains the payload of the data being transmitted.

A data frame may also contain a “Trailer” that contains the integrity check value or the cyclic redundancy check.

Wi-Fi: Frame Translation

It is the task of the access point to translate an 802.11 data frame into an Ethernet frame, when forwarding traffic onto the wired network. This translation is shown.



Getting Data from A to B: Routing and IP Addresses (1)

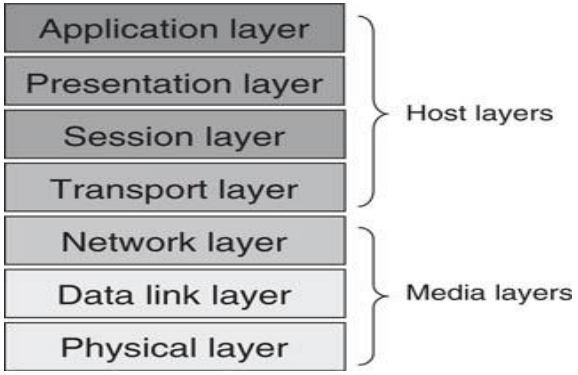
In order for a device in one network to communicate with a device in another, there needs to be a means of routing data between the two networks.

MAC addresses are not suitable at the internet scale, as they are effectively random, and have no structure, which makes it very hard to find a path from A to B, e.g. from one side of the world to the other. They are also tied to the physical hardware device, and cannot (easily) be changed, making them an inflexible addressing method.

Instead, IP (Internet Protocol) technology is layered on top of Ethernet/Wi-Fi to enable “inter-network” (Internet!) communications. This works by assigning unique IP addresses to nodes, and having those addresses structured in a hierarchy, such that network routers can decide which path to send a particular packet down. Routing is solved by being able to split the IP address up, and deduce where to send it next.

IP addresses are assigned at the network layer (Layer 3) of the OSI model (below). One address contains two key pieces of information: the network and the host. The network mask identifies which portion of the address represents the network, and which represents the host. Routers use the network portion of the address to determine which individual network a packet is destined for.

When a packet is sent to an IP address that does not exist on the local network, a router picks up the packet and sends it on. This happens multiple times (each transmission between a router is called a “hop”) until the packet arrives at the destination network.



Getting Data from A to B: Routing and IP Addresses (2)

Internally, routers use routing tables, i.e. lists of networks and the IP address of either the router responsible for that network, or another router that knows about the ultimate destination.

The IP addressing system in use today has two main types: IPv4 and IPv6. IPv4 addresses are 32-bit, often represented in dotted decimal notation (e.g. 192.168.1.1). IPv6 addresses are 128-bit addresses represented in hexadecimal notation (e.g. 2001:ae2:7765::290:1337).

IPv4 addresses are the most common form of IP address in use, but are suffering from a severe shortage. (A 32-bit IP address can only address 2^32 (4.2 billion) different nodes, and there are more than 4.2 billion devices connected to the internet!)

Although IPv6 solves this problem by providing over 3.4×10^{38} IP addresses, its adoption has been slow. Like MAC addresses, so-called “public” IP addresses are globally unique. They are assigned by the Internet Assigned Numbers Authority (IANA) and then delegated to regional Internet registries.

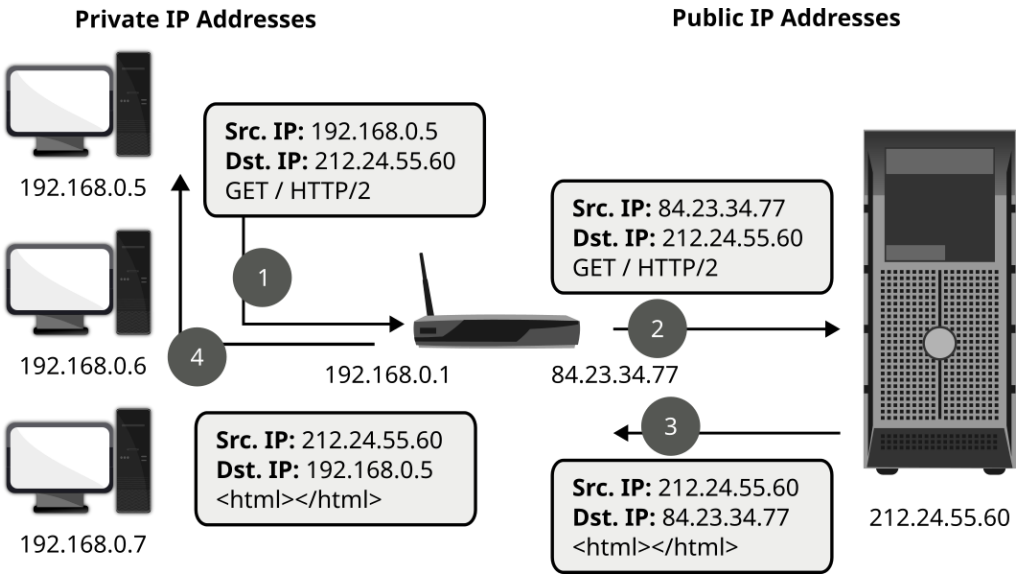
Due to the exhaustion of IPv4 addresses, techniques such as Network Address Translation (NAT) are used to “virtually” inflate the IPv4 address space. This works by keeping a range of IP addresses private to a network, and exposing a single public IP address on the router.

Any packets coming from any host on the private network will look like they have come from the single public IP address, and any packets coming back into the network will be translated back into the corresponding private IP address. This technique allows multiple networks to use the same IP address space, but still allow traffic to be routed between them.

Getting Data from A to B: Network Address Translation

The Figure demonstrates how NAT works.

1. a packet is sent from a device with the private IP address 192.168.0.5, to the server with public IP address 212.24.55.60.
2. the packet has been modified by the router, to replace the Source IP address with the router's public IP address (84.23.34.77). This is because the private IP address has no meaning outside of the network.
3. the server responds to the original packet with a new packet, addressed to the public IP address of the router (84.23.34.77). Finally, at
4. the router has re-written the packet, and replaced the destination IP address with the private IP address of the original device (192.168.0.5) – the one for which the response was intended. NAT relies on *connection tracking* to remember which internal IP address corresponds to which active connection.



The Role of the Gateway (1)

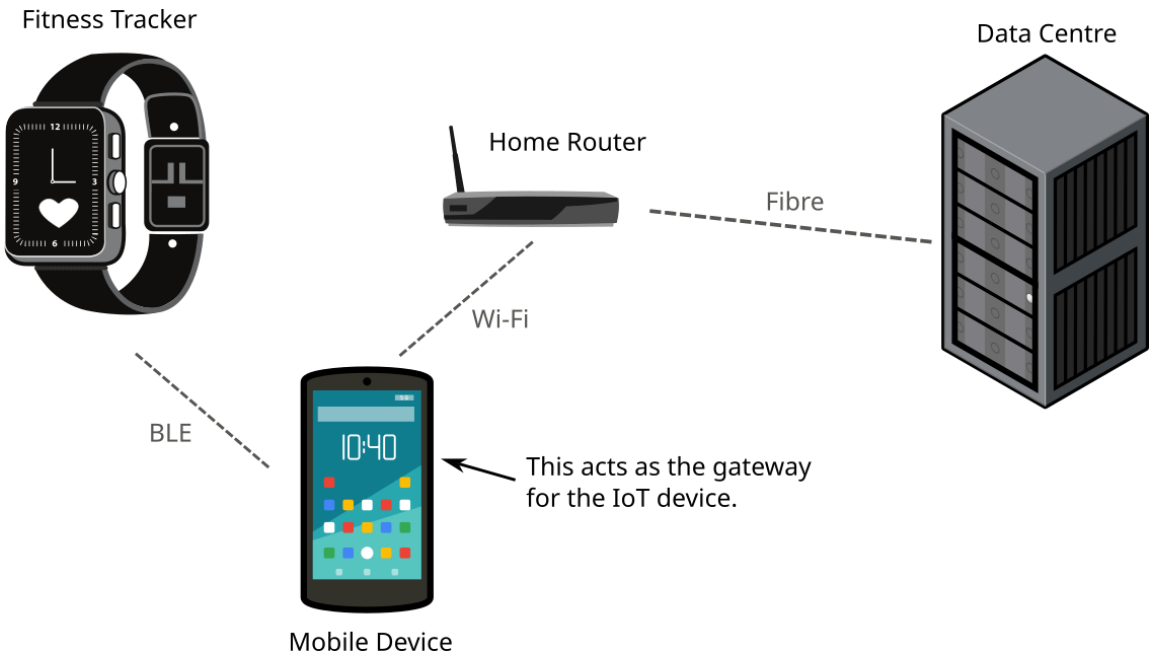
Many IoT devices do not have direct connections to the internet, and do not have an IP address. There needs to be a way for the device to communicate with a server, and this is the job of a network gateway.

A network gateway can simply be a forwarding device, which takes data from a device, and sends it on to a server, relaying any responses back to the device. It could also perform some local data processing, before sending the information on.

The Figure shows how a fitness tracker that only has a Bluetooth LE connection might upload its sensor data to a remote server.

The tracker communicates with a mobile device, and sends its sensor data to an app. The app then connects to the data centre, via a Wi-Fi connection to transmit the collected sensor data from the tracker.

Here, the mobile device is the gateway for the fitness tracker.

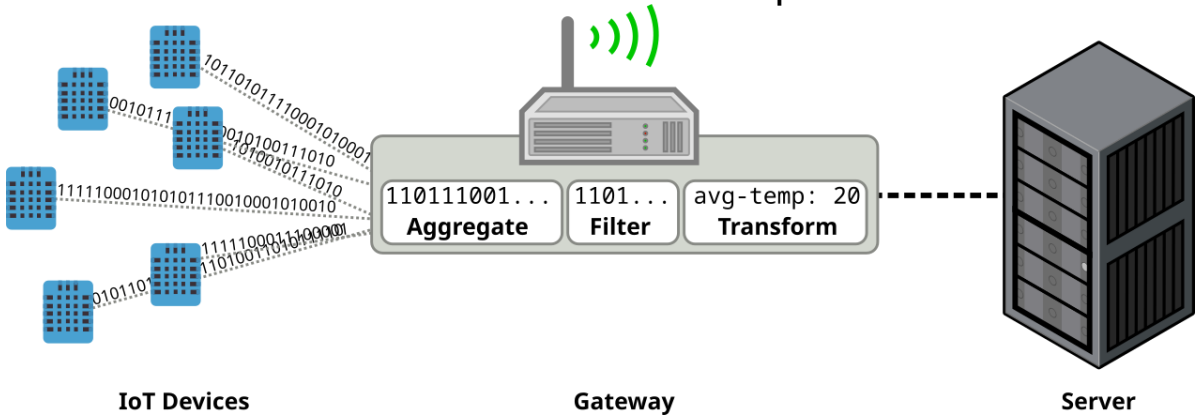


The Role of the Gateway (2)

Network gateways often include advanced functionality that operates on the data that they are responsible for, such as aggregation, filtering, and transformation:

- **Data Aggregation:** Network gateways collect data from multiple devices and aggregate it into a single, centralized data stream. This allows for easier analysis and management of the data.
- **Data Filtering:** Network gateways can be used to filter data to reduce the volume transmitted over the network, and to prevent sensitive or unnecessary data from being transmitted.
- **Data Transformation:** Network gateways can perform data transformations, such as converting data from one format to another, to make it compatible with the backend infrastructure.

In the Figure, a network gateway aggregates data from multiple IoT devices, and turns it into a single data stream. It also strips out unnecessary information that the devices send across, and then transforms it into a format that the server can process.



Data Exchange: Transmission Control Protocol (TCP)

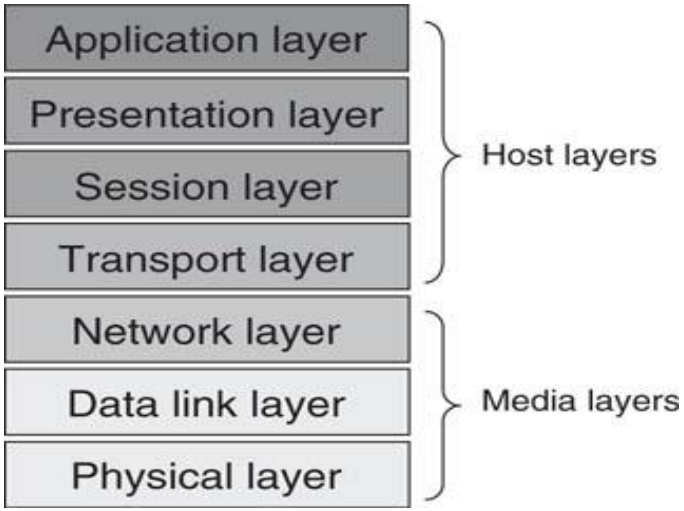
Underpinning most of the application-level network protocols (Layer 7 in the OSI model, below) in use today are TCP and User Datagram Protocol (UDP) layer 4 transport protocols. Both are used for transferring data between applications over an IP network, but they differ in their approach and characteristics.

TCP is a connection-oriented, stream-based protocol, establishing a reliable, two-way communication stream between two devices before any data transfer begins. TCP ensures that data is received in the correct sequence, and that any lost packets are retransmitted, thus providing the illusion of a continuous flow of bytes between devices.

TCP uses a range of techniques to split data up into packets, and to detect when packets go missing, or arrive at the destination out-of-order. TCP will insert missing packets and reconstruct the original ordering, ensuring the application does not see anything out of order..

TCP reliability comes at a cost, as it introduces overhead in the processing pipeline. If an earlier packet goes missing, the entire connection is blocked until it is retransmitted, and re-inserted in the correct position.

TCP is widely used in applications that require a high level of reliability, such as file transfers, email, and web browsing



Data Exchange: User Datagram Protocol (UDP)

UDP is a connectionless, datagram-based protocol, which means that it does not establish a connection before data transfer, and individual messages are transferred one at a time, rather than being a continuous stream.

Conceptually, UDP is “faster” than TCP because it does not have the overhead of connection establishment, sequence checking, and retransmission. However, UDP does not provide reliable data transmission or congestion control, which means that lost packets may not be retransmitted, and data may go missing in the case of network congestion.

UDP is commonly used in applications that require real-time data transfer, but are tolerant to dropped packets, such as online gaming, video streaming, and voice-over-IP (VoIP).

Data Exchange: HTTP (Hypertext Transfer Protocol) 1

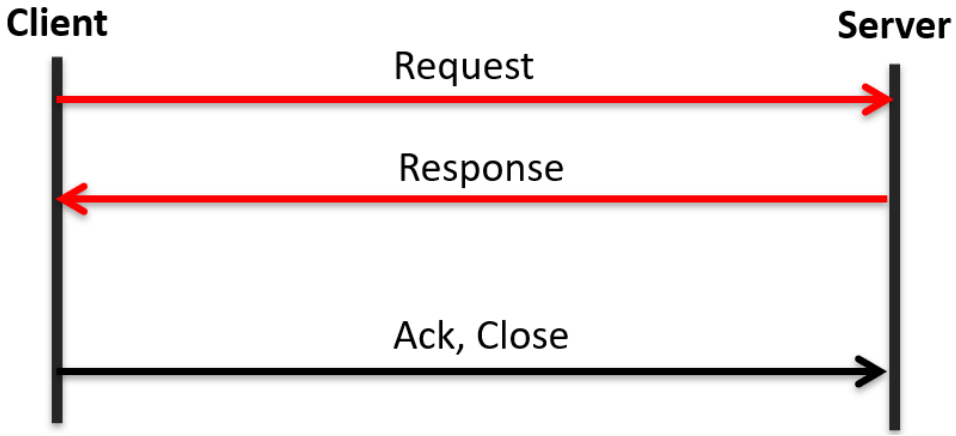
HTTP (Hypertext Transfer Protocol) is a request-response protocol that allows clients to communicate with servers over the internet.

HTTP is based on the client-server model, where the client initiates a request, and the server responds to the request. An HTTP connection is usually established over a TCP/IP link, although some recent developments (QUIC) have started using UDP communication to improve performance.

When a client makes an HTTP request, it sends an HTTP message to the server. The message contains several parts, including a request line, headers, and an optional message body.

The request line includes the method (e.g. GET, POST, PUT, DELETE), the Uniform Resource Identifier (URI) of the requested resource, and the protocol version (e.g. HTTP/1.1).

The headers provide additional information about the request, such as the type of content being requested and the client's preferred language. The message body may contain data, such as form data or a JSON payload.

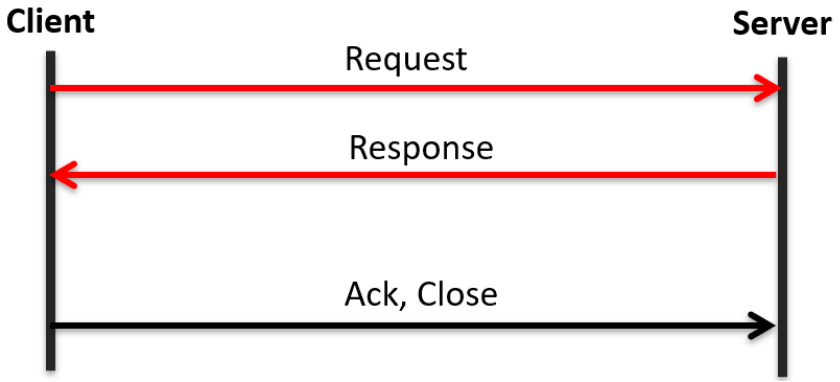


Data Exchange: HTTP (2)

The server then processes the request and sends an HTTP response back to the client. The response contains several parts, including a status line, headers, and a message body.

The status line includes the protocol version, a numeric status code, and a brief reason phrase (e.g. “200 OK”, or “404 Not Found”).

The headers provide additional information about the response, such as the type of content being returned and the server's software version. The message body contains the requested resource or an error message.



Data Exchange: HTTP (3)

HTTP is a stateless protocol, meaning that the server does not retain any information about the client between requests. However, cookies and sessions can be used to maintain state across multiple requests, should the application require this.

There are a few different versions of HTTP, the most common is HTTP/1.1, although HTTP/2 and HTTP/3 have emerged, designed to improve the performance, security and functionality.

HTTPS (HTTP Secure) is a variant of HTTP that uses SSL (Secure Sockets Layer) or TLS (Transport Layer Security) to encrypt data being transmitted over the connection. This added security helps to protect sensitive information, such as login credentials and financial data, from being intercepted.

HTTP is well suited to IoT applications for several reasons:

- **Widespread adoption:** HTTP is the foundation of the World Wide Web and is widely supported by web browsers, servers, and other devices.
- **Simple and well-documented:** HTTP is simple and easy to implement and understand. It is well-documented, with many resources and tutorials available.
- **Efficient:** HTTP is designed to be efficient, with minimal overhead and a small packet size. This makes it well-suited for use in low-power, resource-constrained IoT devices.
- **Proven:** HTTP has been in use for decades and has been proven to be a robust and reliable protocol for communication over the internet.
- **RESTful Web Services:** HTTP is the foundation of RESTful (Representational state transfer) web services, which is an architectural style for building web services.
- **Security:** HTTPS can be used to encrypt data being transmitted over the internet, which helps to protect sensitive information from being intercepted by malicious actors.

Data Interchange Formats: JSON and SOAP

The IoT requires transmission of data between devices, and that data needs to be encoded. There are a number of popular formats that can be used to transmit data, which maximise compatibility with other systems. These formats are usually used in conjunction with an HTTP connection.

JSON (JavaScript Object Notation) is a lightweight data-interchange format that is easy for humans to read and write and easy for machines to parse and generate. It is based on a subset of the JavaScript programming language, and is often used as the format for data exchange in web APIs. The program example below gives an example of a JSON data structure, which describes the values of some IoT sensors.

SOAP (Simple Object Access Protocol) is an alternative (and older) protocol for exchanging structured information. SOAP uses XML (eXtensible Markup Language) as its message format, and can be carried over a variety of lower-level protocols.

SOAP is more verbose than JSON and requires more bandwidth to transmit, which can make it less efficient. However, it provides more built-in functionality for things like data validation, error handling, and support for complex data types.

Data Interchange Formats: Protobuf

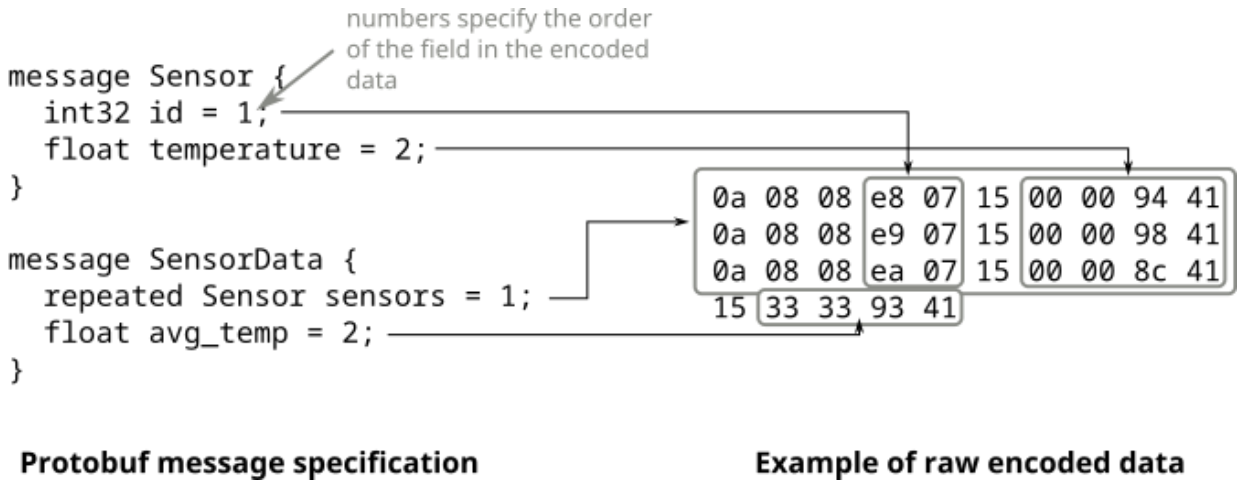
Protocol Buffers (often called Protobuf) is a binary data serialization format developed by Google. It is similar to JSON and XML, although it is not “self describing” in the sense that the data being transmitted does not define its own structure.

Both sides of a Protobuf link must agree on the structure beforehand, which in turn makes it more efficient and smaller in size, especially when it comes to larger data sets.

It uses a specific interface definition language to define the data structure and then generates code in several programming languages that can be used to read and write data in the format.

It is particularly useful in situations where the data is being transmitted over a network and bandwidth is at a premium – so well suited for the IoT!

Protobuf requires additional setup and configuration compared to JSON and SOAP, and it is not as widely supported. The Figure shows how a Protobuf specification translates to raw data bytes. Unlike JSON and SOAP, this encoding is not human readable.



Challenges and Risks in Building IoT Systems (1)

Building IoT systems is challenging for a variety of reasons. Key challenges include:

Complexity: IoT systems are inevitably complex, involving many different types of devices and technologies that need to work together seamlessly. This can be difficult to achieve, especially when dealing with many devices or a wide range of environments.

Scalability: IoT systems often need to be able to scale to millions or even billions of devices. Each device needs to be able to communicate with others, or with the cloud. The system needs to be designed to handle the data generated by all these devices.

Security: IoT systems can be vulnerable to security threats, such as hacking or malware. This is because many IoT devices have limited processing power and are connected to the internet, making them easier to attack.

Interoperability: Different IoT devices and systems often need to be able to work together, regardless of their manufacturer or technology. There many differing ways for IoT devices to communicate, and different devices may use different protocols or technologies.

Privacy: IoT systems can generate a large amount of data about people's habits and behaviours. As IoT systems become more widespread the data they generate becomes more valuable.

Cost: Building and maintaining an IoT system can be costly, especially when dealing with many devices or complex environments. This can be a challenge for businesses and organisations that want to adopt the IoT but may not have the resources or budget to do so.

Challenges and Risks in Building IoT Systems (2)

Some of the key risks include:

Legal risks: There are several legal risks associated with the IoT, such as liability for data breaches or privacy violations. There are also questions about who is responsible for the actions of IoT devices, such as self-driving cars or industrial robots.

Operational risks: IoT systems can be complex and can involve many different devices and technologies that need to work together seamlessly. This can be a risk to organisations that adopt IoT systems, as problems with the system could result in disruptions or downtime, leading to loss of revenue.

Strategic risks: Adopting IoT systems can involve significant investments of time and resources, and there is always the risk that the technology will not deliver the expected benefits. This could be a risk to organisations that adopt IoT systems, as they could end up with a system that does not meet their needs or that becomes obsolete quickly.



Security and Privacy (1)

Security and privacy issues in IoT systems include a wide range of potential vulnerabilities and threats, such as unauthorised access to device data, data breaches, denial of service attacks, and malware infections.

The Open Web Application Security Project (OWASP) is a community-led organisation which aims to improve software security. It maintains an interesting list of the top 10 web security risks, which can be seen on its web site.

Numerous examples have appeared in the press and on the internet reporting IoT security failings.

A well-known one relates to a reported vulnerability of the Philips Hue lightbulb. In one scenario, the attacker (in range of the bulb) plays around with the color or brightness settings to convince the controller that the bulb is faulty. The controller tries to reset it, deletes the bulb and attempts to reconnect. The bulb (now hacker-controlled, with updated firmware) re-joins the IoT network. The bulb bombards the controller with data requests, triggering a buffer overflow. The hacker installs malware on the controller, which is connected to the target IP network. The attacker can now infiltrate the target network to spread ransomware or spyware.

Security and Privacy (2)

These issues can be mitigated through a variety of technologies and best practices, including:

Device authentication and access control: Ensure that only authorised devices and users can access the network and control devices, using authentication methods like password protection, biometric authentication (fingerprints), and public key infrastructure (PKI).

Network security: Protecting the communication channels and infrastructure used by IoT devices. This can be done through encryption and the use of secure protocols such as HTTPS.

Firewalls: Firewalls are the first line of defence into a network, and should be configured to be as restrictive as possible.

Intrusion detection and prevention systems (IDPS): These systems monitor network traffic for suspicious activity, and can take action to block or alert on potential threats.

Software updates and patch management: Keeping devices up to date with the latest security patches and software updates can help prevent known vulnerabilities from being exploited.

Data privacy: Ensuring that personal information is protected, and only shared with authorised parties, through techniques such as anonymisation or using pseudonyms to protect personal identity.

Compliance and regulations: Adhering to industry-specific regulations, such as the General Data Protection Regulation (GDPR), can help ensure that IoT systems are secure and compliant with relevant laws and regulations.

Security and Privacy (3)

Implementing security and privacy on IoT devices can be challenging due to their limited resources such as processing power, storage, and memory. However, there are several techniques and technologies that can be used to effectively secure these devices, such as:

Lightweight security protocols: Constrained devices may not have the resources to support complex security protocols, so it's important to use lightweight and efficient protocols such as Datagram Transport Layer Security (DTLS) or Secure MQTT (a lightweight machine-to-machine network protocol).

Secure boot: Secure boot is a process that ensures that only authenticated and trusted firmware can be executed on the device. This can help prevent unauthorised access to the device and protect against malware.

Secure storage: Constrained devices may not have the resources for full disk encryption, so it's important to use secure storage methods such as trusted platform modules (TPMs) to protect sensitive data.

Over-the-air (OTA) updates: Constrained devices may not have the ability to physically connect to a computer for software updates, so it's important to implement OTA update mechanisms to ensure that devices are kept up to date with the latest security patches.

Minimising the attack surface: The “attack surface” defines the range of possible entry points that an attacker can use to attempt to gain access to a device. Therefore, keeping this minimal reduces the number of opportunities for attackers to infiltrate a system.

Code signing: Ensuring that the firmware running on devices is authentic and has not been tampered with, by signing the code digitally before it is flashed on the device.

Dealing with IoT Scale and Complexity

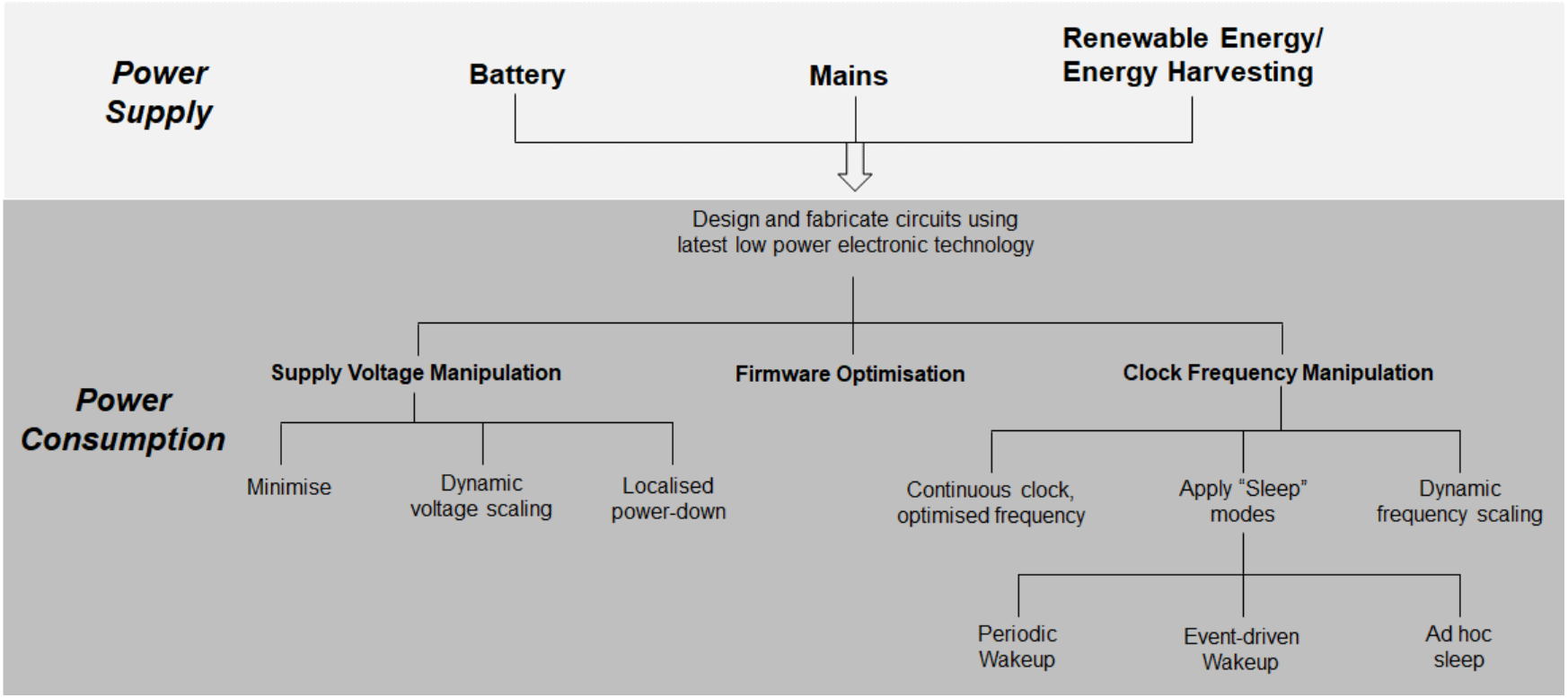
There are several strategies that can be employed to manage the scale and complexity of IoT systems effectively:

- Start with a well-designed architecture that separates out the different components of the IoT system. An architecture that employs so-called “microservices” can help to break down the system into smaller, more manageable components that can be developed, deployed, and maintained independently.
- Implement an effective device management solution to manage the system scale. In an IoT system with millions of devices it is practically impossible to manually configure, and update each of those devices individually. A solution that allows management and maintenance tasks to be largely automated is critical to supporting an IoT system of this scale.
- Manage the flow and storage of data. This can include implementing data storage solutions that are designed for the high volume, high data rate, and variety of IoT data, and using data analytics tools to gain insights into the data and make informed decisions.
- Choose scalable infrastructure solutions, such as these cloud-based platforms. This helps to ensure that the system remains performant and reliable as the number of devices and amount of data grows. Public cloud providers offer a wide range of services spanning from data storage through to complex artificial intelligence frameworks. These enable scalability, as the providers have the power of large data centre resources available on demand. Developers gain access to advanced technologies, which would require specialist expertise to implement internally.

Powering the IoT “Things” (1)

Many IoT “things” are battery powered, and need to optimise their power consumption. The Figure shows a low power “street map”, summarising the options available to the designer.

For *any* electronic product, we need to consider its power source(s), choosing from the three options shown. An IoT device may draw on any of mains, battery or renewables.



Powering the IoT “Things” (2)

Assuming a more or less given circuit, and noting the inextricable link between power consumption, clock frequency and supply voltage, three broad headings appear in power consumption management:

Firmware. Inefficient code requires more clock cycles to get through it, implying a higher clock frequency and hence higher power consumption. Code for a low power device should be as simple and efficient as possible. Simple choices, like choosing integer arithmetic over floating point, may allow you to run at a reduced clock frequency, and so save power. Other options may centre on reduction of sampling rates, or reducing the amount of data transmitted, or its frequency of transmission.

Clock frequency manipulation. One possibility is simply to choose the lowest continuous clock frequency that will allow the program to run satisfactorily. An alternative is to vary clock frequency according to need, a technique sometimes known as *dynamic frequency scaling*. Where intensive calculations are required for example, the CPU clock frequency is increased. Where only slow-speed background tasks are needed, the clock frequency, and hence power consumption can be reduced.

The clock can also be turned off entirely to unneeded parts of the microcontroller circuit, including possibly the CPU, resulting in a range of *idle, sleep* or *power down* modes.

Supply voltage manipulation: A circuit should operate at the lowest possible supply voltage, as long as circuit performance is not compromised. There are also sometimes opportunities to power down temporarily unused parts of a circuit, or sub-systems within a microcontroller. *Dynamic voltage scaling* allows the operating voltage to be adjusted according to need, noting that a higher supply voltage tends to allow faster operation.

Chapter Review

- There are several different networking technologies involved in making a communication systems work.
- The Ethernet protocol underpins many of the wired communication networks today, and allows data to be exchanged within a local network.
- The Wi-Fi protocol performs a similar function to the Ethernet protocol, but extends this to work wirelessly.
- Wi-Fi is the most widely used wireless networking protocol in the world, and is designed to integrate seamlessly with ethernet networks.
- The Internet Protocol allows data to be exchanged between networks, by associating IP addresses with devices, and specifying how routes to other networks can be discovered and traversed.
- Web services are a popular way of implementing internet-based services, and typically use the HTTP protocol, or its secure HTTPS variant, to operate.
- Data transfer between services requires that data to be encoded in ways that the communicating parties can understand. JSON, SOAP, and Protobuf are example formats.
- There are a wide range of challenges and risks, such as complexity, scalability, interoperability, and cost that need to be considered when designing and implementing an IoT system.
- Security and privacy are challenges that exist in every part of an IoT system.
- There is a range of very effective techniques available to minimise power consumption in an IoT “thing”.