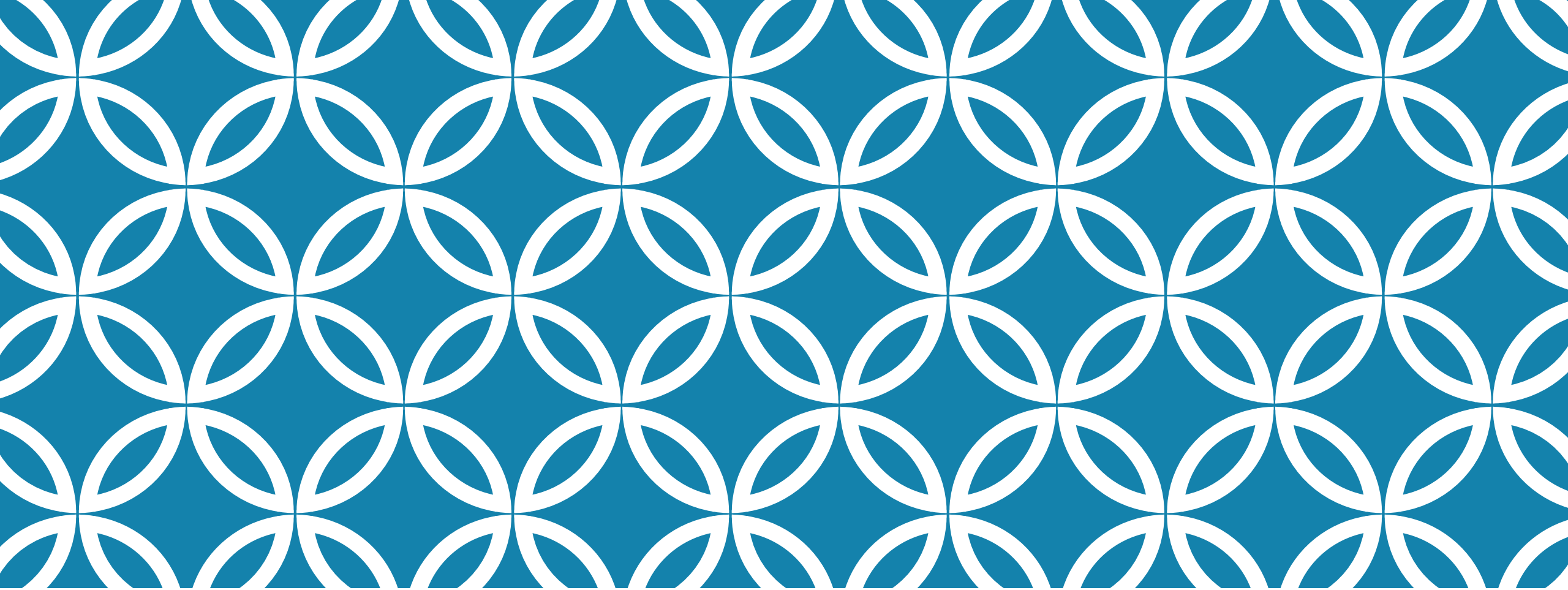


# DESIGN PRINCIPLES FOR WEB CONNECTIVITY

By Rahul Shrivastava

# WEB COMMUNICATION PROTOCOLS FOR CONNECTED DEVICES

- The protocols are used for communication between machines or between a machine and server.
- Due to constraints in processing capabilities and the low power requirements of IoT devices (which are generally meant to be deployed in environments with constant battery power) with limited bandwidth capabilities, a need was felt for dedicated standards and protocols especially designed for IoT.
- Since those who manufacture IoT devices and those who create the IoT platforms are different, this required industry standards and protocols that were not high on power consumption, bandwidth usage, or processing power and could be adopted easily by all IoT players— hardware manufacturers, software developers or cloud solutions/service providers.

# COAP (CONSTRAINED APPLICATION PROTOCOL)

- The Constrained Application Protocol (CoAP) is a specialized web transfer protocol designed for use in constrained environments and low-power, resource-constrained devices in the context of the Internet of Things (IoT). CoAP is similar in many ways to the HTTP protocol used for the World Wide Web but is optimized for IoT devices with limited processing power, memory, and network bandwidth.

# COAP (CONSTRAINED APPLICATION PROTOCOL)

**Lightweight Protocol**: CoAP is designed to be extremely lightweight. Its header is only a few bytes long, which makes it suitable for devices with limited memory and processing capabilities.

**RESTful**: CoAP follows a Representational State Transfer (REST) architectural style, just like HTTP. It uses similar methods (GET, POST, PUT, DELETE) to interact with resources identified by URIs (Uniform Resource Identifiers).

**UDP-Based**: CoAP is typically implemented over UDP (User Datagram Protocol), which is a connectionless and lightweight transport protocol. This choice reduces overhead and makes it suitable for constrained devices and lossy networks.

# COAP (CONSTRAINED APPLICATION PROTOCOL)

**Request-Response Model**: CoAP follows a simple request-response model, similar to HTTP. Clients send requests to servers, and servers respond with appropriate responses, which can include data or status codes.

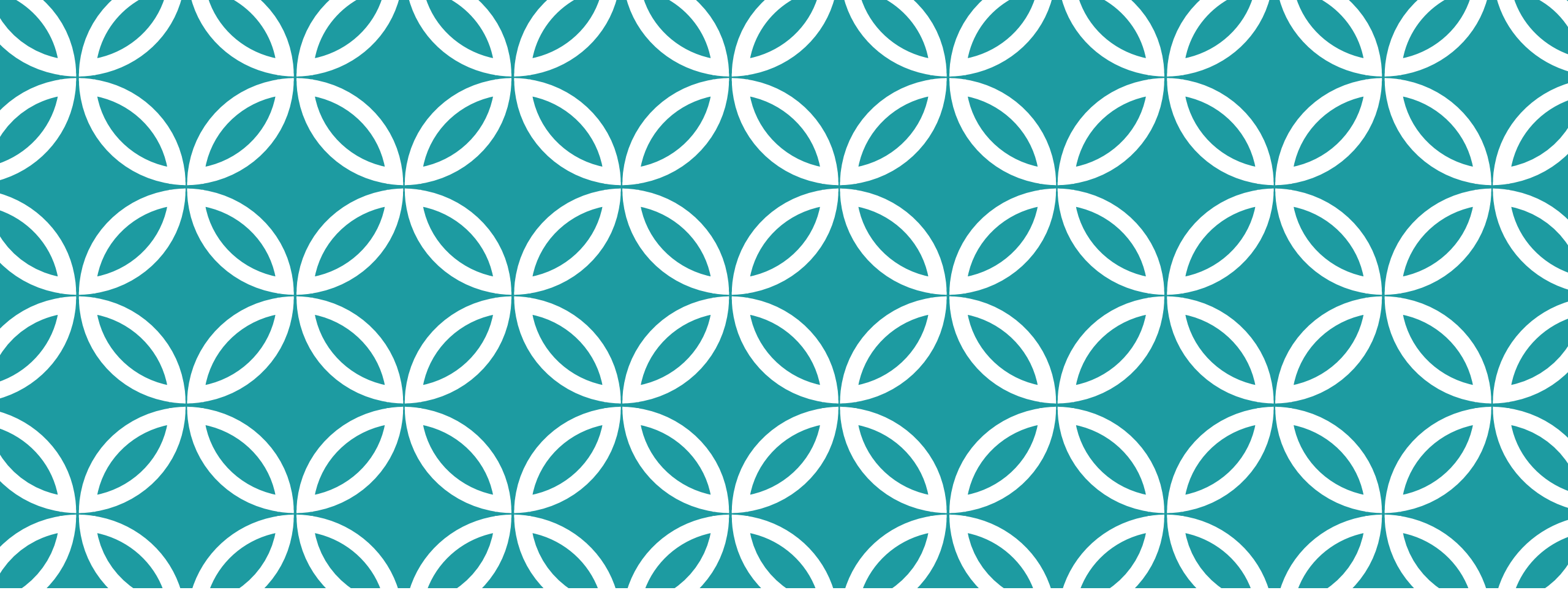
**Multicast Support**: CoAP supports multicast, allowing a single CoAP request to be sent to multiple devices simultaneously. This is useful for applications like device discovery in IoT networks.

**Observing Resources**: CoAP allows clients to "observe" resources. When a resource changes, the server can push updates to all clients observing that resource. This is helpful for real-time monitoring of sensor data or other dynamic information.

# COAP (CONSTRAINED APPLICATION PROTOCOL)

**Low Power and Energy-Efficient:** CoAP is designed to minimize power consumption and is well-suited for battery-operated IoT devices that need to conserve energy.

**Block Transfer:** CoAP supports block-wise transfer of large payloads, allowing it to handle resource representations that are larger than the maximum packet size of the underlying transport.



# REPRESENTATIONAL STATE TRANSFER

# REPRESENTATIONAL STATE TRANSFER

REST, which stands for Representational State Transfer, is an architectural style and a set of constraints for designing networked applications. It was introduced by Roy Fielding in his doctoral dissertation in 2000 and has since become a popular choice for designing web services and APIs due to its simplicity, scalability, and statelessness. Here are key features and aspects of REST:

# REPRESENTATIONAL STATE TRANSFER

**Stateless**: In a RESTful architecture, each request from a client to a server must contain all the information needed to understand and process the request. Servers do not maintain any client state between requests. This statelessness simplifies scalability and allows for easy load balancing.

**Resources**: In REST, everything is considered a resource, which can be a physical object, a data entity, or even a service. Resources are identified by URIs (Uniform Resource Identifiers), such as URLs (Uniform Resource Locators).

# REPRESENTATIONAL STATE TRANSFER

**HTTP Methods:** RESTful interactions are typically performed using the standard HTTP methods:

**GET:** Retrieve data from the server.

**POST:** Create a new resource on the server.

**PUT:** Update an existing resource on the server.

**DELETE:** Remove a resource from the server.

**PATCH:** Partially update a resource.

# REPRESENTATIONAL STATE TRANSFER

**Uniform Interface:** REST emphasizes a uniform and consistent interface for interacting with resources. This makes it easy for clients to understand how to use the API without needing to learn custom methods or operations.

**Representation:** Resources can have multiple representations, such as JSON, XML, HTML, or other formats. Clients can request the representation that best suits their needs using the "Accept" header in the HTTP request.

**Stateless Communication:** Each request from a client to a server is independent, meaning that the server does not store any client-specific information between requests. This makes it easier to build highly scalable systems.

# REPRESENTATIONAL STATE TRANSFER

**Client-Server Architecture:** REST encourages a separation of concerns between the client and server, enabling them to evolve independently. This separation promotes modularity and allows for better scalability and flexibility.

**Layered System:** REST supports a layered architecture, where each component (e.g., a load balancer, cache, proxy) can be added or modified without affecting the overall system. This improves fault tolerance and scalability.

**Caching:** REST allows for caching of responses to improve performance and reduce server load. The use of HTTP caching headers helps control the caching behavior.

# REPRESENTATIONAL STATE TRANSFER

**Idempotent Operations:** Some HTTP methods in REST, such as GET, PUT, and DELETE, are idempotent, meaning that making the same request multiple times will have the same effect as making it once. This property is important for ensuring predictable behavior in distributed systems.

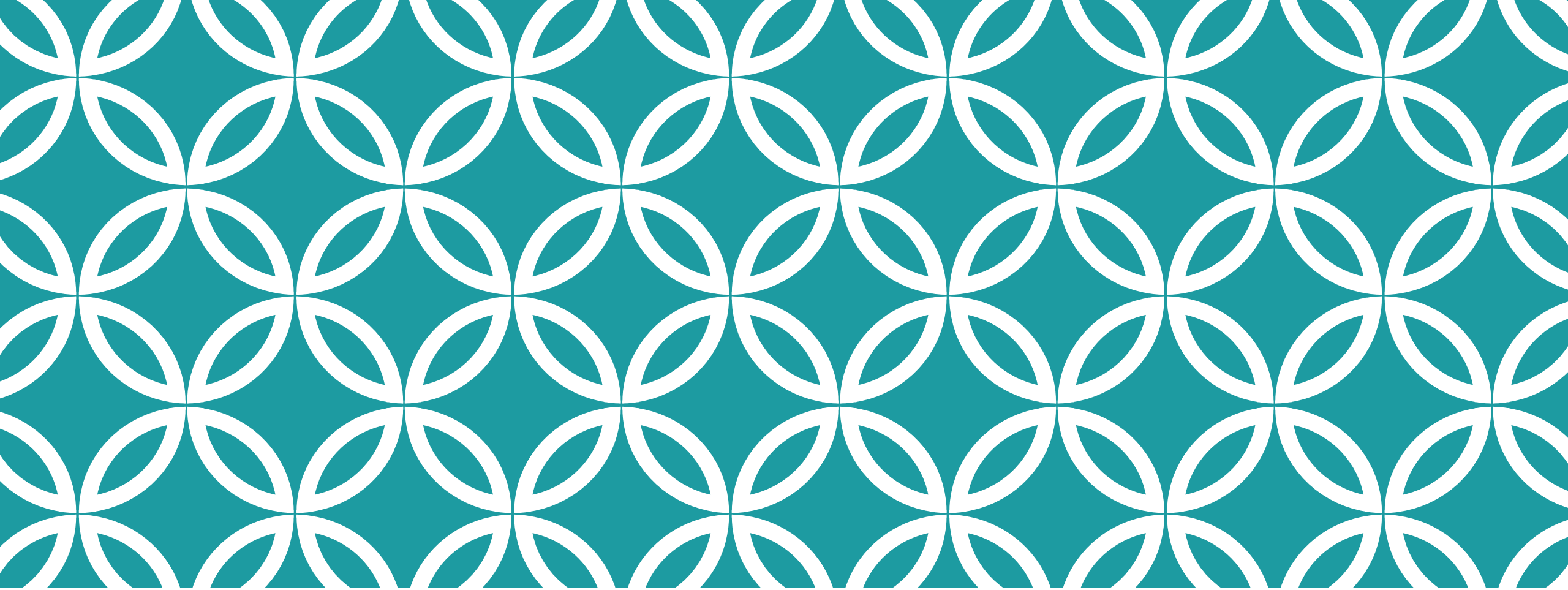
**Security:** REST can use standard HTTP security mechanisms such as HTTPS for data encryption and authentication. Additional security mechanisms can be implemented as needed.

**Scalability:** RESTful APIs are highly scalable due to their stateless nature and simple request-response model. Load balancers can distribute incoming requests across multiple servers.

**Simplicity:** REST is designed to be simple and easy to understand, making it accessible to a wide range of developers and clients.

# REPRESENTATIONAL STATE TRANSFER

REST is commonly used for building web services and APIs, and it has become the basis for many web applications and mobile app backends. It is particularly well-suited for scenarios where simplicity, scalability, and interoperability are essential.



# WEBSOCKET

# WEBSOCKET

WebSockets is a communication protocol that provides full-duplex, bidirectional communication channels over a single TCP (Transmission Control Protocol) connection. Unlike traditional HTTP communication, which follows a request-response model, WebSockets enable persistent, real-time, and interactive communication between clients (such as web browsers) and servers. Here are some key characteristics and features of WebSockets:

# WEBSOCKET

**Full-Duplex Communication**: WebSockets allow both the client and server to send and receive data simultaneously, making it ideal for real-time applications where immediate responses are required.

**Low Latency**: Unlike HTTP, which requires establishing a new connection for each request-response cycle, WebSockets maintain a single connection, resulting in lower latency and reduced overhead.

**Bidirectional**: WebSockets support bidirectional communication, meaning that either the client or server can initiate data transmission at any time. This enables interactive and dynamic applications.

# WEBSOCKET

**Stateful**: Unlike the stateless nature of HTTP, WebSockets maintain a persistent connection, preserving the state of the communication session. This is beneficial for applications that require ongoing interaction, such as online games or chat applications.

**Efficient**: WebSockets are efficient in terms of both bandwidth and processing power because they eliminate the need for repeated handshakes and headers associated with HTTP requests.

**Real-Time Updates**: WebSockets are commonly used for real-time updates in web applications, such as live chat, notifications, stock tickers, and collaborative tools.

# WEBSOCKET

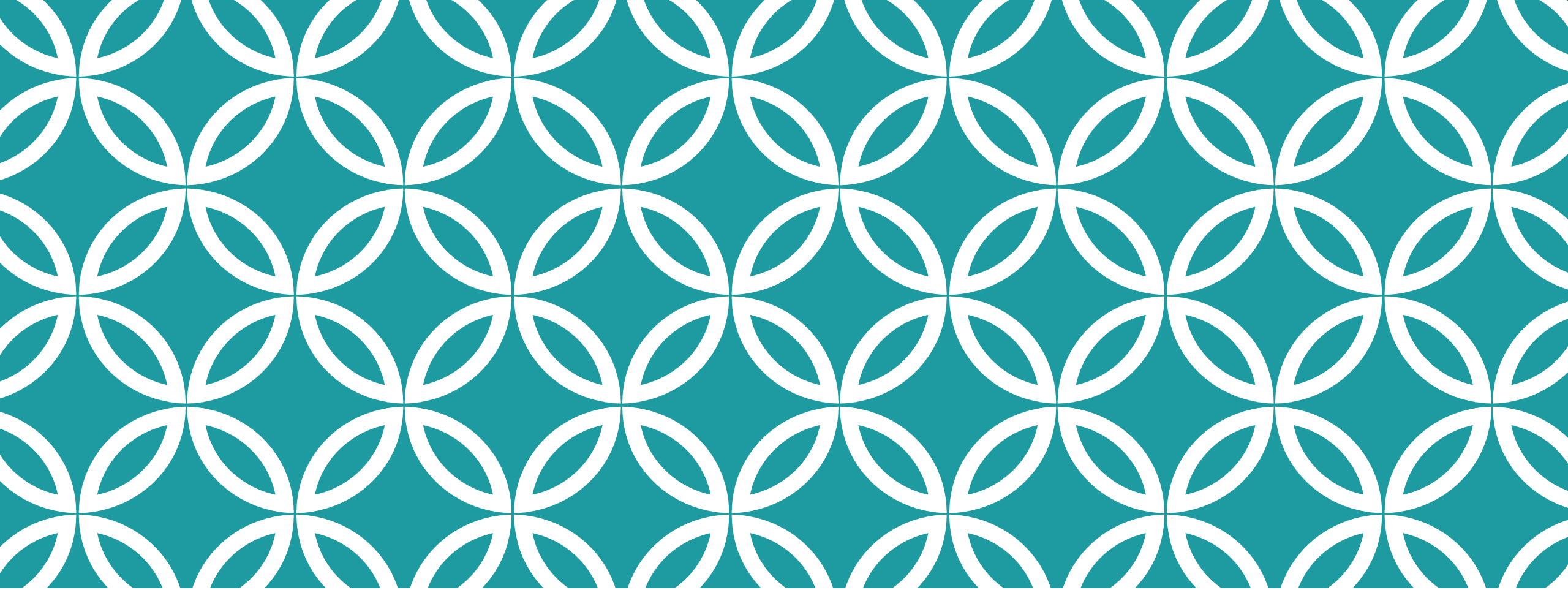
**Web-Based Applications:** WebSockets are well-suited for web-based applications and are often used in conjunction with HTML5, JavaScript, and other web technologies to create interactive and dynamic web applications.

**Protocols:** WebSocket communication begins with a handshake using HTTP, but once the connection is established, it switches to the WebSocket protocol. The WebSocket protocol uses a simple frame-based format for data exchange.

**Security:** WebSocket connections can be secured using the WebSocket Secure (WSS) protocol, which is essentially WebSocket over HTTPS. This provides encryption and ensures data confidentiality and integrity.

# WEBSOCKET

WebSockets have become a crucial technology for building modern web applications that require real-time, interactive, and dynamic features. They have found applications in chat applications, online gaming, financial trading platforms, collaborative tools, live sports scores, and numerous other use cases where immediate and bidirectional communication is essential.



# IP ADDRESSING IN IOT

# IP ADDRESSING IN IOT

IP (Internet Protocol) addressing in IoT (Internet of Things) is a critical aspect of networking, as it provides a means to identify and communicate with IoT devices over the internet or local networks. Here are some key considerations regarding IP addressing in IoT

# IP ADDRESSING IN IOT

**IPv4 and IPv6**: IoT devices can use either IPv4 or IPv6 addresses. IPv4 addresses are the older, more common format (e.g., 192.168.1.1), but they are running out due to the limited address space. IPv6 addresses are longer and offer a vastly larger address space (e.g., 2001:0db8:85a3:0000:0000:8a2e:0370:7334), making them well-suited for the large number of IoT devices.

# IP ADDRESSING IN IOT

**Static IP Address:** Manually assigned to the device, often used for fixed infrastructure or critical devices.

**Dynamic Host Configuration Protocol (DHCP):** Addresses are assigned dynamically by a DHCP server, which can simplify management but requires network connectivity for address assignment.

**IPv6 Stateless Address Autoconfiguration (SLAAC):** IoT devices using IPv6 can generate their addresses based on the network prefix, reducing the need for a centralized DHCP server.

**IoT-Specific Addressing Solutions:** In some IoT deployments, proprietary or specific addressing schemes are used for efficiency, security, or compatibility with low-power devices.

# IP ADDRESSING IN IOT

**Subnetting:** In larger IoT deployments, subnetting can be employed to segment the network into smaller, more manageable subnetworks. Each subnet can have its IP address range and routing rules, which can improve network performance and security.

**Network Address Translation (NAT):** NAT is often used in IoT networks to conserve public IPv4 addresses. IoT devices within a private network share a single public IP address, and NAT translates internal private addresses to the public address when communicating over the internet.

**IP Address Management (IPAM):** Effective IP address management is crucial in IoT deployments. It involves tracking IP addresses, allocating them efficiently, and ensuring that there are no conflicts or address exhaustion issues.

# IP ADDRESSING IN IOT

**Security**: IoT devices are vulnerable to attacks if not properly secured. Proper IP addressing and network segmentation can help enhance security. Implementing firewalls, access control lists (ACLs), and intrusion detection systems can further protect IoT networks.

**Scalability**: IoT networks can scale rapidly as more devices are added. Choosing the appropriate IP addressing scheme and subnetting strategy can ensure that the network remains manageable as it grows.

**Mobility**: Some IoT devices may be mobile, moving between different networks. Mobile IPv6 and other mobility protocols can be used to maintain connectivity as devices change locations.

# IP ADDRESSING IN IOT

**Quality of Service (QoS)**: IoT applications with varying requirements may need different QoS levels. IP addressing should be aligned with QoS policies to ensure that critical data is prioritized over less important traffic.

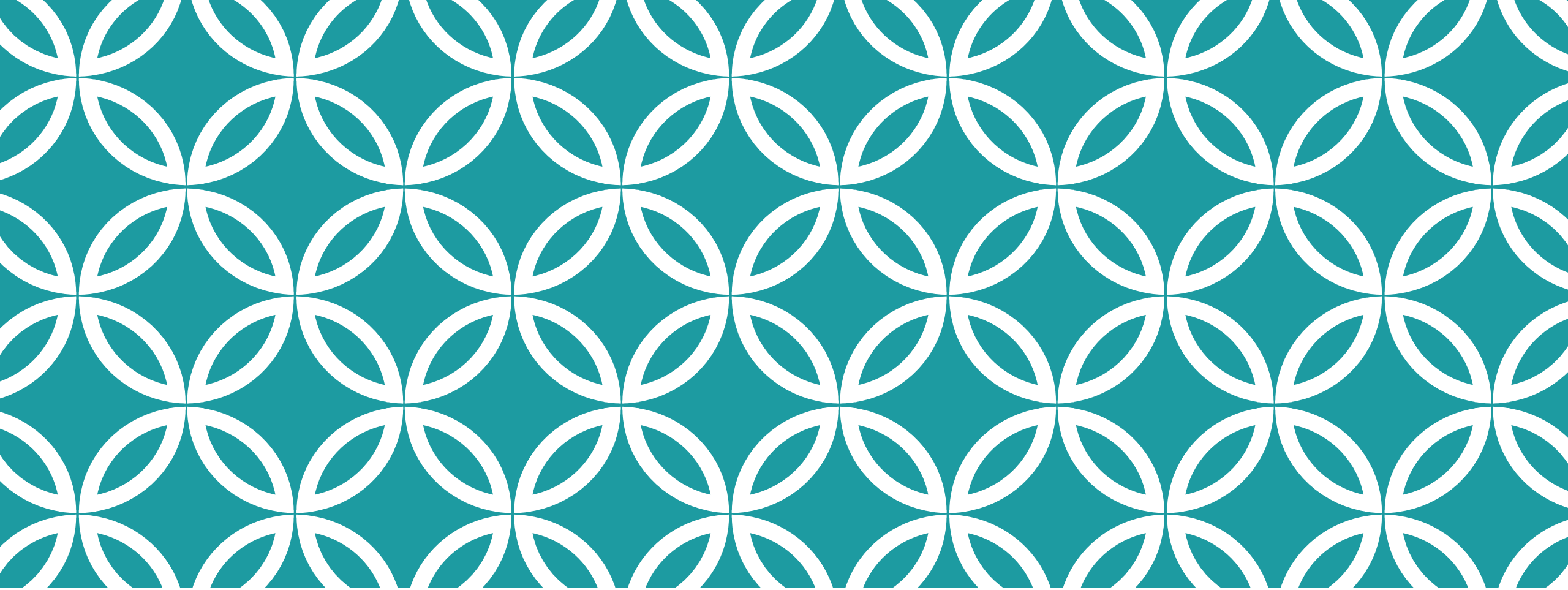
**IPv6 Transition**: As IPv6 adoption grows, IoT networks need to consider the transition from IPv4 to IPv6. Dual-stack networks that support both IPv4 and IPv6 can ease this transition.

**Edge Computing**: In IoT deployments, edge computing can be used to process data closer to the source. This may affect IP addressing decisions, as some data processing may occur locally before data is sent to central servers.

# IP ADDRESSING IN IOT

**DNS (Domain Name System)**: DNS is essential for translating human-readable domain names into IP addresses. Proper DNS setup ensures that IoT devices can be accessed using meaningful names rather than just numerical IP addresses.

IP addressing in IoT should be carefully planned to accommodate the specific requirements of the IoT deployment, including scalability, security, mobility, and the transition to IPv6. Proper addressing strategies and management contribute to the efficient and secure operation of IoT networks.



# MEDIA ACCESS CONTROL

# MEDIA ACCESS CONTROL

Media Access Control (MAC) is a sublayer of the data link layer (Layer 2) in the OSI (Open Systems Interconnection) model of network communication. MAC is responsible for managing access to the physical network medium, such as Ethernet, Wi-Fi, or token ring, and it plays a crucial role in governing how devices share and access the communication channel. Here are key aspects and functions of MAC:

# MEDIA ACCESS CONTROL

**Addressing:** MAC addresses, also known as hardware addresses or physical addresses, are unique identifiers assigned to network interface cards (NICs) and are used at the data link layer to identify devices on a local network segment. These addresses are typically represented in hexadecimal notation and are six bytes long in Ethernet.

**Data Link Layer:** MAC operates within the data link layer and interacts directly with the physical layer, which is responsible for transmitting raw bits over the network medium.

# MEDIA ACCESS CONTROL

**Access Control Methods**: MAC defines various methods for controlling access to the shared network medium, depending on the underlying technology. Common MAC access control methods include:

- **CSMA/CD (Carrier Sense Multiple Access with Collision Detection)**: Used in Ethernet networks, this method involves devices listening for a clear channel before transmitting data. If a collision is detected, devices follow a backoff algorithm to retry transmission.
- **CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance)**: Used in Wi-Fi networks, this method involves devices requesting permission (using a protocol called Request-to-Send/Clear-to-Send, or RTS/CTS) before transmitting data to avoid collisions.
- **Token Passing**: Used in token ring networks, this method involves devices passing a token, which grants them permission to transmit data. Only one device can transmit at a time, ensuring collision-free communication.

# MEDIA ACCESS CONTROL

**Frame Formatting**: MAC is responsible for framing data into packets suitable for transmission over the network medium. It adds information such as source and destination MAC addresses, frame type, and error-checking information to the data.

**Address Resolution Protocol (ARP)**: ARP is a protocol at the data link layer that helps resolve IP addresses to MAC addresses. It is used when devices need to communicate on a local network.

**Switching**: In modern Ethernet networks, switches operate at the data link layer and use MAC addresses to make forwarding decisions. They create and maintain MAC address tables to efficiently forward frames to the appropriate devices.

# MEDIA ACCESS CONTROL

**Duplex Modes**: MAC also manages the duplex mode of network communication. In half-duplex mode, devices can either transmit or receive but not both simultaneously. In full-duplex mode, devices can transmit and receive simultaneously, improving network efficiency.

**Flow Control**: MAC provides flow control mechanisms to manage the rate at which data is sent and received to prevent congestion and ensure efficient data transfer.

**Error Detection and Correction**: MAC may include error-detection mechanisms, such as CRC (Cyclic Redundancy Check), to identify and, in some cases, correct errors in transmitted frames.

# MEDIA ACCESS CONTROL

**Security:** MAC addresses can be used in access control lists (ACLs) to control which devices are allowed or denied access to a network.

In summary, Media Access Control (MAC) is a crucial component of network communication, particularly at the data link layer. It governs how devices share the physical network medium, assigns unique hardware addresses (MAC addresses), manages frame formatting, controls access to the network channel, and ensures efficient and reliable data transmission within local network segments. Different technologies and protocols may use variations of MAC for access control and network management.