International Journal of Computer Science and Engineering (IJCSE)
ISSN (P): 2278–9960; ISSN (E): 2278–9979
Vol. 13, Issue 2, Jul – Dec 2024; 97–136

International Academy of Science, Engineering and Technology
Connecting Researchers; Nurturing Innovations

OPEN-SOURCE INNOVATION IN CROSS-DOMAIN IOT: STRATEGIES FOR INTEGRATION, INTEROPERABILITY, AND PLATFORM MANAGEMENT

Abhishek Kumar Vishwakarma[1] & Dr. Rahul Kumar Ghosh[2]

[1]Research Scholar, Ram Krishna Dharmarth Foundation University, Ranchi, Jharkhand, India

[2]Assistant Professor, Ram Krishna Dharmarth Foundation University, Ranchi, Jharkhand, India

ABSTRACT

The Internet of Things (IoT) has driven major technological advancements, transforming how we live, work, and interact with devices. IoT connects various objects through the Internet, changing how we think about and use technology in daily life. This shift has led to new applications and opportunities in fields like smart cities, healthcare, industrial systems, and home automation, allowing for more advanced, responsive, and integrated services. However, a significant challenge remains in fully integrating IoT systems and ensuring they can work together across different sectors and fields, each with unique demands. Achieving seamless connectivity and functionality, known as interoperability, is essential for making IoT truly universal.One promising approach to improving IoT interoperability and integration is through open-source solutions. Open-source IoT platforms provide free access to frameworks, technologies, and tools, enabling developers and stakeholders to build upon established systems and adapt them to their needs. This paper explores the potential of open-source solutions to address the challenges of integrating IoT systems across multiple domains. By focusing on open-source IoT frameworks and technologies, we aim to understand how these solutions can help achieve better connectivity and usability across various application areas. To assess the potential and usability of open-source IoT platforms, we examine them from the perspectives of different stakeholders, including device manufacturers, application developers, and end-users. Our study involves a gap analysis that considers several key factors affecting IoT integration. These factors include managing diverse types of sensors and actuators, handling and sharing data securely, addressing privacy concerns, and examining the overall readiness of the IoT ecosystem for widespread deployment. By identifying the gaps and limitations in current IoT systems, we aim to highlight areas where open-source solutions can offer practical improvements. Additionally, this paper addresses the importance of edge computing as a complement to IoT platforms, especially for applications that require real-time data processing. Edge computing enables data to be processed closer to the source, rather than sending it to a central server. This approach is crucial for time-sensitive applications where delays in data processing could affect performance, such as in healthcare monitoring or industrial automation. By incorporating edge computing into IoT systems, we can enhance the responsiveness and efficiency of these platforms, making them better suited for time- critical tasks. In summary, this paper aims to provide a comprehensive overview of how open-source IoT solutions can address the current challenges of cross-domain integration and interoperability. By examining the strengths and limitations of existing platforms, we hope to offer insights that will guide future development and adoption of open-source IoT frameworks. Through a deeper understanding of these platforms, we can foster a more connected, efficient, and adaptable IoT ecosystem, paving the way for further advancements across a wide range of applications

KEYWORDS: *Internet of Things (IoT), Open Source, IoT Platforms, Heterogeneous IoT, Integration, Interoperability, Management.*

Article History

***Received:** 01 Nov 2024 | **Revised:** 06 Nov 2024 | **Accepted:** 14 Nov 2024*

INTRODUCTION

The emergence of Internet of Things (IoT) has altered the way business is done by connecting devices, systems and networks easily. There is an increasing dependence on IoT centers to take responsibility of vast networks of connected gadgets that produce and share huge amounts of information in real-time. Nevertheless, the boom of this IoT landscape has challenged a great deal in relation to integration, management and inter-operability, especially between various domains. Often devices and platforms produced by different vendors have no semblance of a common language and controlling these situations across health care, agriculture, smart cities, manufacturing etc. is quite complex.

One of the most interesting ways to solve these problems is to make even more use of such solutions based on open-source principles. Open-source IoT development tool kits adopt the appropriate level and breadth of integration and interoperability without serving the interests of any one developer. These systems encourage cooperation among the developers and thus, the systems become more flexible and scalable in terms of IoT systems management avoiding the disadvantages of vendor lock. In addition, the open source structures allow for the modification and even development of the system from the user's perspective, enhancing the management of the inter operability of cross domain IoT platform.

IOT FUNCTIONAL BLOCKS

An IoT System includes a number of functional blocks to provide various services to the system such as sensing, identification, actuation, communication and management [1] as shown in the figure 01

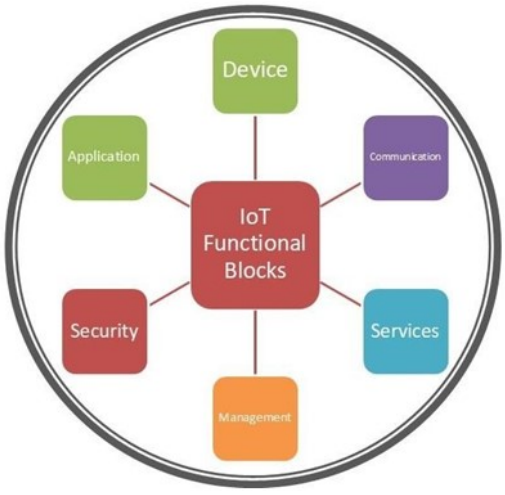

Figure 1: IoT Functional Blocks.

Device

An IoT system based on sensors, and actuators devices that can share and communicate with other connected applications and devices or collect data from different devices. These data are processed at local or cloud-based applications and after that, these data are used to perform further processes and may store data for future reference [2].

An IoT device can incorporate several ports, which may be for linking it with other devices. These include I/O interfaces for sensors, interfaces for the Internet, memory and storage interfaces, and audio/video interfaces. IoT devices can also be of varied types, for instance, wearable sensors, smart watches, LED lights, automobiles, and industrial machines. However, most IoT devices are capable of incorporating some kind of data collection which when processed by data analytical systems could lead to actionable information insights either for further local or international actions.

For example, data collected from a sensor to control a room temperature can be used to identify a required room temperature and sustain it once that temperature has been achieved [3].

Communication

The communication block performs the exchange of data between remote servers, devices, and applications. Communication protocols, basically work in the data link layer, network layer, transport layer, and application layer [4].

Services

An IoT system serves different types of functions like device identification, authentication, discovery, analysis, control, and management.

Management

This block manages the IoT system in search of the underlying governance of the IoT system. Management block can handle, device configuration, firmware updates etc [5]. It can also provide analytics and reporting to enable system administrators to understand how the system is operating and where to improve its usability.

Security

IoT systems protect through the security functional blocks providing authentication, authorization, privacy, message integrity, content integrity, and data security. It defends against illegal access in all data aspects, thereby preventing as well as protecting data against threats [6]. It also employs intrusion detection as well as response capabilities, thus assisting in the prevention of harm and minimizing their impact.

Application

In IoT systems applications play the most important role for users by providing a friendly interface to control and monitor different aspects of the system. An application may contain a display, interactive buttons, touchscreen, microphone, camera, etc. for user interaction [7].

PROTOCOLS IN THE IOT SYSTEM

Many IoT standards are proposed to streamline application programmers' and service providers' work. Several communities have been established to supply protocols supporting the IoT such as initiatives taken by W3C, IETF, EPCglobal, IEEE, and ETSI [8].

Communication protocols compete with each other to become the most popular and first choice for connected devices, but each application prefers its appropriate protocol. This point will create a difference between the functional protocol and the optimal solution.

When selecting a wireless communication technology, it's important to choose the one that best fits your specific needs. Existing wireless technologies offer different benefits and drawbacks, so the choice isn't always clear. That's why new protocols have been developed, tailored to the needs of connected devices. These protocols focus on features like low power usage, wide range, limited data speed, and easy setup. Most of these protocols are categorized based on the key layers of the IoT architecture.

Many different protocols play a role in IoT setups, but communication protocols are essential for making IoT networks function. Choosing the right protocol depends on factors like the distance you need the connection to cover, how much power is available, the speed of data transfer, and how long it takes for the data to travel—all while ensuring security. As mentioned earlier, IoT devices use specific network standards and protocols to enable communication between connected devices and the cloud. These network protocols are sets of rules that define how devices talk to each other.

Most devices connect to the internet using the Internet Protocol (IP), but they can also connect locally through technologies like Bluetooth, NFC (Near-Field Communication), and others. The key differences between these connection types are power usage, range, and processing power. IP-based connections are more complex, needing more power and memory but offering unlimited range. On the other hand, Bluetooth connections are simpler, using less power and memory but limited in range.

Devices like smartphones and computers use network communication protocols, but these protocols aren't always ideal for IoT solutions, especially when it comes to bandwidth, latency, and distance. IoT- specific communication protocols have been designed to work within the limitations of IoT devices, such as their lower processing power and range, while still ensuring reliable communication. Since traditional protocols like Wi-Fi don't fully meet IoT requirements, new IoT-specific protocols have been developed to address these needs.

One of the biggest challenges in designing IoT networks is power consumption. That's why low-power (LP) technologies are often preferred. These typically fall into two categories:

- Low-Power Wide Area Networks (LPWAN): These can cover long distances (up to several kilometers) but have lower data rates. Examples include 6LoWPAN, LoRaWAN, Sigfox, NB-IoT, and Wi-Fi HaLow.
- Wireless Personal Area Networks (WPAN): These cover shorter distances but support faster data rates. They are better for applications that require more speed over shorter ranges, like Bluetooth, ZigBee, and LiFi.

For instance, ZigBee can provide a range of up to 100 meters with data rates up to 250 kbps, while Bluetooth Low Energy can reach data rates of up to 3 Mbps.

- LPWAN (Low Power Wide Area Networks) is becoming more popular in both industry and research because of its ability to offer low-power, long-range, and low-cost communication. It can cover distances of 10–40 km in rural areas and 1–5 km in urban areas, depending on the environment, with data transfer speeds ranging from 0.3 kbit/s to 50 kbit/s per channel [9]. It's also very energy-efficient, with devices lasting over 10 years on a single battery, and it's inexpensive, with radio chipsets costing less than 2 euros and yearly operating costs around 1 euro per device [10-11]. Due to these benefits, LPWAN is ideal for IoT applications that only need to send small amounts of data over long distances. While traditional cellular networks can handle wide-area communication, their high power consumption and complex infrastructure (such as antennas and amplifiers) make them less suitable for IoT applications [12]. LPWAN, on the other hand, supports simpler, low-power devices that don't

require expensive gateways. This allows for the use of inexpensive batteries that can last for years, making it a more practical option than cellular networks.

LPWAN started gaining attention only after 2013, and now several technologies like Sigfox, LoRa, and NB-IoT are leading in this space. These technologies work in both licensed and unlicensed frequency bands and have key technical differences. An example of an LPWAN protocol is 6LoWPAN, which combines IPv6 and LoWPAN technologies, offering benefits like excellent connectivity, compatibility with older systems, low energy usage, and self-organizing networks [13-14].

- WPAN, or Wireless Personal Area Network, is a network where devices are connected wirelessly within a short range, typically around 10 meters, and are centered around a person's workspace. It's sometimes called "person-centered short-range wireless connectivity" because it's designed for close- proximity communication, where devices interact as if they were connected by cables. Unlike WLAN, WPAN doesn't need much infrastructure, making it a power-efficient and low-cost solution for connecting various devices.

WPAN connections often happen in small areas like a room or a building. For example, two or more devices can chat and share files over Wi-Fi within the same room. Other examples of WPAN include Bluetooth devices, wireless mice, wearable tech, USB drives, digital cameras, thermostats, security systems, lighting controls, motion sensors, and leak sensors [15].

According To IEEE Standards [17], WPAN Is Classified Into Three Types

- High-rate WPAN (HR-WPAN) – Based on IEEE 802.15.3, offering data speeds over 20 Mbps.
- Medium-rate WPAN (MR-WPAN) – Defined by IEEE 802.15.1, with speeds around 1 Mbps.
- Low-rate WPAN (LR-WPAN) – Defined by IEEE 802.15.4, with data speeds less than 0.25 Mbps.

WPANs usually work as local mesh networks, where devices are connected directly to each other without needing a gateway. This mesh setup makes the network simple, resilient, and cost-effective because it doesn't require extra equipment. ZigBee is one of the most widely used mesh protocols in IoT, offering short-range communication but with very low power consumption. While ZigBee can transmit data at higher speeds compared to LPWAN, it's best suited for smaller areas like smart home networks due to its limited range [16].

In IoT, communication can be wired or wireless, and the protocols used vary depending on the type of connection and the layer of the network involved.

Application Layer

The application layer has five main protocols: MQTT, CoAP, REST, XMPP, and AMQP. Each of these protocols has its features, benefits, and security challenges

MQTT

The Message Queuing Telemetry Transport (MQTT) is a messaging protocol as shown in figure 02, designed for publishing and subscribing, using a simple client/server model that runs over TCP/IP or other network protocols. It's ideal for IoT environments because it's open, lightweight, and easy to implement, making it perfect for devices with limited resources or networks with high latency and low bandwidth [18].

MQTT, introduced by Andy Stanford Clark at IBM in 1999 and standardized by OASIS in 2013, is widely used due to its simplicity and small message header, requiring less power than other messaging protocols [19]. It's commonly used in IoT because it operates on a publish-subscribe model rather than a request-response type, making it highly flexible for IoT applications like mobile-to-mobile (M2M) communication and remote telemetry [20]. Security in MQTT involves authentication, authorization, and secure communication, especially in critical applications handling sensitive data.

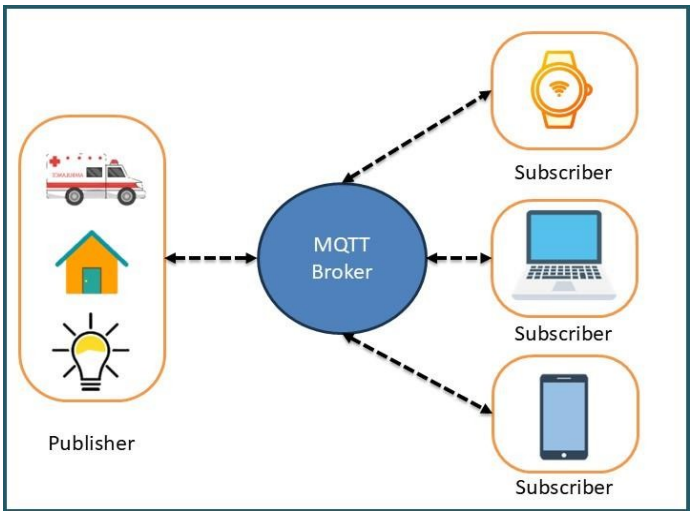

Figure 2: MQTT Model Of Message Exchange In IoT.

The MQTT protocol has four key components [21-22]:

- Broker: Acts as the server, managing data exchange between devices and sensors. It ensures devices connect automatically and supports three levels of Quality of Service (QoS).
- Topic: Used by sensors and devices to categorize and share information.
- Publisher: A device or sensor that sends data (messages) to the broker.
- Subscriber: A device that receives data from the broker, based on the topics it has subscribed to.

Clients can either publish or subscribe to topics, receiving messages filtered through the broker. Publishers, typically lightweight sensors, send data to the broker and then go into sleep mode, while subscribers connect to receive the data they are interested in. The broker handles the distribution of messages by filtering them by topic and sending them to the appropriate subscribers [23]. MQTT is designed to create an embedded connection between middleware, applications, communications, and networks, making it a key protocol for IoT solutions.

COAP

The Constrained Application Protocol (CoAP) is a lightweight web transfer protocol designed for use with constrained devices and networks, as described in RFC 7252. It's ideal for machine-to-machine (M2M) applications, such as managing supply chains and monitoring smart meters for energy consumption. CoAP integrates well with HTTP, making it easier to connect with the web. However, CoAP itself is not secure, which is a major drawback, although security can be added using Datagram Transport Layer Security (DTLS), which isn't widely adopted in IoT [24].

CoAP operates on a request/response model as shown in figure 03, and was developed by the Internet Engineering Task Force (IETF) and Constrained RESTful Environment (CORE) to provide a lightweight RESTful interface [25]. It's commonly used in applications like smart energy systems and environmental monitoring. Designed for low-power, limited-resource devices, CoAP allows these devices to interact through RESTful communication, which is similar to HTTP but more efficient for constrained environments. Instead of using TCP, as HTTP does, CoAP is built on the UDP platform, which reduces bandwidth needs but also decreases reliability. IETF later created a version that allows CoAP to run over TCP to improve reliability [26].

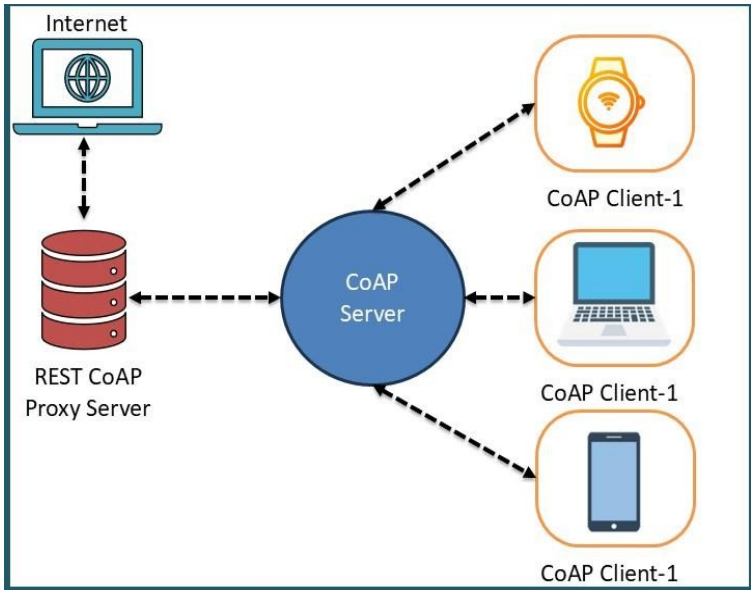

Figure 3: Coap Model of Message Exchange in IoT.

CoAP's architecture consists of two sub-layers: the messaging sub-layer, which handles reliable message delivery using stop-and-wait transmission, and the request/response sub-layer, which manages communication and can support both synchronous and asynchronous responses.

CoAP has four transmission modes:

- Confirmable: Ensures reliable message delivery.
- Non-confirmable: Allows non-reliable transmission.
- Piggyback: Direct communication between client and server with acknowledgment.
- Separate: For more flexible communication patterns.

CoAP uses familiar methods like GET, PUT, POST, and DELETE to retrieve, create, update, and delete messages in the network, much like HTTP, but with reduced overhead, making it suitable for resource-constrained IoT environments [27].

REST

REST, which stands for Representational State Transfer, is an architectural style used to build web services, not an IoT protocol itself. It provides a framework for creating.

RESTful web services or APIs, allowing different systems to interact over the web. When an API call is made, REST dictates how the request is formatted and what format the response will be in. It is flexible and can work over any communication protocol, though it commonly uses HTTP or CoAP in IoT settings to interact with components like files, objects, and media [28].

RESTful web services use standard HTTP methods such as POST, DELETE, PUT, and GET to handle requests. It was introduced by Fielding and defines a set of rules or constraints to design efficient systems [29]. Some key constraints include:

- Client-server: Separation between the client and server for better scalability.
- Statelessness: Each request from the client contains all the information needed, improving reliability and visibility.
- Cache: Enhances network efficiency by allowing data to be stored temporarily.
- Uniform interface: Provides a standard way for components to interact.
- Layered system: Organizes the system in layers for better scalability and security.
- Code-on-demand (optional): Allows servers to send executable code to the client.

REST is widely used due to its simplicity, scalability, and compatibility across various devices and platforms, making it an ideal choice for web services and IoT applications [30].

XMPP

XMPP (Extensible Messaging and Presence Protocol) is an open standard messaging protocol, initially developed by the IETF for instant messaging. It was designed to enable real-time message exchanges between applications, offering basic security features like authentication and end-to-end encryption [31]. XMPP is a text-based protocol built on XML, and it can support both client-server and publish-subscribe communication models.

XMPP is now being adapted for IoT applications because of its flexibility and extensibility through XML. It allows users to create topics, publish information, and notify subscribers when new data is available [32]. Communication between clients and servers is done through XML streams, with data being exchanged using XML tags or "stanzas," which come in three types:

- <presence/>: notifies about status updates,
- <message/>: carries message content,
- <iq/>: used for info/query purposes.

While XMPP supports security features like TLS (Transport Layer Security), which ensures data integrity and confidentiality, it has some limitations for IoT. Since it uses XML, the message size can be large, making it inefficient in networks with low bandwidth. Additionally, it lacks reliable Quality of Service (QoS) guarantees and is not practical for machine-to-machine (M2M) communication in low-power, lossy networks typically used in IoT. However, ongoing efforts are working to optimize XMPP for better performance in IoT environments [23].

Though initially intended for instant messaging, XMPP is being repurposed for IoT, making it more versatile. Despite its challenges, its extensibility and security features make it an option worth considering for certain IoT applications.

AMQP

AMQP (Advanced Message Queuing Protocol) is a lightweight, open-standard messaging protocol used for machine-to-machine (M2M) communication. Developed by OASIS and standardized by ISO, AMQP is widely used in business environments for its ability to support interoperability across different systems. It operates as a binary application layer protocol and works well with various messaging patterns like request- response and publish-subscribe models [33].

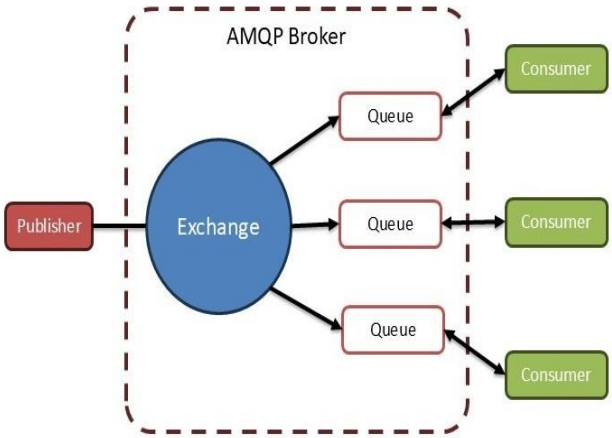

Figure 4: AMQP Model of Message Exchange In Iot.

AMQP was designed as shown in figure 04, to handle large volumes of messages efficiently, using techniques like point-to-point communication and store-and-forward. Like MQTT, it follows a publish-subscribe model and runs over TCP, providing reliable message delivery. Its main advantage is its flexibility, allowing different platforms, even those built with different programming languages, to exchange messages. This makes it particularly useful for heterogeneous systems [28].

In AMQP, messages are self-contained, and their contents are opaque, meaning they can support messages of any size. The protocol uses entities like "exchanges" and "queues" to route messages between publishers and subscribers [34]. The newer version of AMQP (1.0) adds even more flexibility by enabling direct messaging between clients without a broker, whereas older versions rely on brokers for message routing.

AMQP is also known for its high level of security, incorporating encryption via TLS and authentication through SASL. However, one drawback is its resource-heavy nature, as it requires significant power, processing, and memory, making it less suitable for IoT systems with bandwidth or latency constraints [35].

Despite these challenges, RabbitMQ, a popular implementation of AMQP, is commonly used as a message broker to store and forward messages in distributed systems [36]. It has proven to be reliable for applications where message exchanges are frequent and need robust handling. However, in mobile networks and IoT settings, AMQP's heavy nature means it's not always the best choice.

Transport Layer

The transport layer handles communication between devices over a network, ensuring data is sent and received reliably. It offers services like supporting data streams, ensuring reliability, managing multiple connections, and providing security for the transmission.

Protocols used at the transport layer are

TCP

Transmission Control Protocol (TCP) is a widely used connection-oriented protocol designed for reliable data transmission, as shown in figure 05. It operates in three phases: establishing a connection, exchanging data, and closing the connection. This makes it ideal for applications requiring consistent data delivery, such as web browsing and file transfers [37].

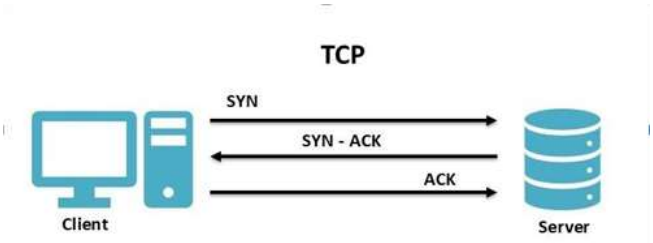

Figure 5: Transmission Conotrol Protocol (TCP)

TCP works by creating a secure virtual circuit between the sender and receiver before data transmission begins, similar to a phone call where both parties must be connected before they can communicate. Data sent through TCP is broken down into smaller packets, each with a unique sequence number, ensuring that it can be reassembled in the correct order at the destination. If a packet is lost or not acknowledged within a certain time, TCP automatically retransmits it [38].

Although TCP guarantees reliable data delivery and is perfect for applications involving large data transfers, it comes with certain drawbacks. The protocol adds overhead to each packet and requires more power due to the constant need for acknowledgment and retransmission, making it less suitable for some IoT systems with low bandwidth and power constraints. However, its error-checking features, flow control, and ability to handle large volumes of data ensure its continued dominance in Internet communications [39].

UDP

User Datagram Protocol (UDP) as shown in figure 06, is a connectionless transport protocol that is often used in IoT systems because it has low overhead and reduces bandwidth consumption, unlike TCP. However, UDP doesn't guarantee message reliability as it lacks flow control, error checking, and packet retransmission [40]. It is ideal for applications where speed and small data size are prioritized over security and reliability, such as video or voice streaming. While UDP can result in packet loss and unpredictable performance, it offers faster data delivery and supports features like checksum error control and process-to-process communication [41].

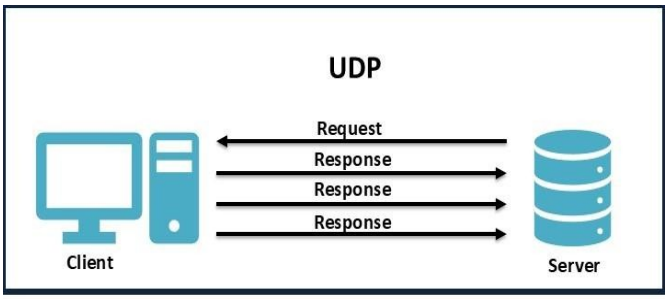

Figure 6: User Datagram Protocol (UDP).

DCCP

Datagram Congestion Control Protocol (DCCP) is a transport layer protocol designed for bidirectional unicast connections, commonly used in applications like streaming media and VoIP [42]. Unlike TCP, which focuses on reliable, in-order delivery but may cause delays, and UDP, which has no built-in congestion control, DCCP strikes a balance. It provides unreliable transport like UDP but includes adaptive congestion control, similar to TCP, without guaranteeing reliability. This makes DCCP ideal for delay-sensitive applications that prioritize speed over strict reliability, offering the benefits of congestion control to prevent network overload while maintaining a simpler, faster transmission process [43].

SCTP

The Stream Control Transmission Protocol (SCTP) is a reliable transport layer protocol designed for transmitting messages, especially in Public Switched Telephone Network (PSTN) signaling over IP [43]. Unlike TCP, SCTP supports multiple streams of data between two connected endpoints, ensuring that data is delivered in the correct order and without loss. It also includes security measures to prevent attacks like flooding and masquerading. Sometimes referred to as "next-generation TCP" (TCPng), SCTP is useful for internet-based telephone connections but may offer more features than needed for simpler applications like DCCP, including built-in congestion control mechanisms [44].

RSVP

The Resource Reservation Protocol (RSVP) sets up a unidirectional path for data flow in the network. It starts when an inbound router sends a "path" message to the outbound router, which includes details about the required resources. As the message passes through each router, these routers store information about the resource reservation [45].

Once the outbound router receives this path message, it starts the resource reservation process by sending a "reservation" message back to the inbound router. Each router along the path passes the reservation message upstream, following the same route as the initial path message. Once the inbound router receives the reservation message, the path for data flow is fully established.

RSVP sessions remain active as long as they continue, with maintenance messages sent every 30 seconds to keep the path open. If a router doesn't receive these updates for three minutes, it will end the session and reroute the data through a different router [46].

RSVP creates a separate session for each data flow, identified by a destination address, port, and protocol. Multiple senders can be part of the same session, each with its source address and port. The process starts when a sender initiates RSVP path messages to the destination, and receivers, such as in a multicast setup, may register to join using protocols like IGMP.

Once the receivers get the path messages, they send back "Resv" messages to make the necessary resource reservations. Data can begin to flow before all reservations are completed, but without a confirmed reservation, the traffic is treated as normal best- effort data without quality guarantees [47].

TLS

Transport Layer Security (TLS) is a widely used protocol that ensures secure communication between client and server applications over the internet. It prevents data from being intercepted, altered, or forged by using cryptographic algorithms. First introduced in 1999 and updated to version 1.3 in 2018, TLS is primarily used for securing web communications but can also encrypt emails, messaging, and voice-over IP (VoIP) [47].

TLS operates by providing three key features:

- Encryption: This keeps the data being exchanged private, ensuring third parties cannot access it.
- Authentication: It verifies that both the sender and receiver are who they claim to be.
- Integrity: It checks that the data hasn't been tampered with during transmission.

The process starts with a TLS handshake, which initiates a secure connection by agreeing on encryption methods and authenticating the client and server. Once the handshake is complete, a secure channel is established for data transfer [48].

TLS is the successor of Secure Sockets Layer (SSL) and uses two protocols [26]:

- TLS Handshake Protocol: This authenticates the parties and negotiates encryption methods.
- TLS Record Protocol: This ensures the integrity and confidentiality of data during transmission, ensuring it remains unaltered and inaccessible to unauthorized parties.

TLS can work with other protocols, such as UDP, to secure communications in various types of network applications.

DTLS

Datagram Transport Layer Security (DTLS) is a version of the TLS protocol designed to secure communications in datagram-based networks, where issues like packet loss and reordering can occur [49]. While TLS works well for stream-based communications, it doesn't suit environments using protocols like UDP, which DTLS is built to handle. By making key adjustments, DTLS maintains the security of TLS while being more reliable for datagram transmissions.

DTLS offers strong encryption and security similar to TLS but with lower overhead and reduced latency, making it a good fit for IoT devices and protocols like CoAP that use UDP for communication. With DTLS, encryption is applied without the need for a TCP/TLS stack, and resource consumption is optimized by using efficient cipher suites and pre-shared keys [50].

Key steps involved in using DTLS include packet input/output, tracking connection states, and processing packets by encrypting or decrypting them [51]. These features make DTLS ideal for securing real-time communications while keeping performance efficient.

RPL

RPL, or the Routing Protocol for Low-Power and Lossy Networks, is designed to handle networks with limited memory, processing power, and energy. It's commonly used in IoT applications and is based on the IPv6 standard. RPL builds a tree-like structure called a Destination Oriented Directed Acyclic Graph (DODAG) for routing data. In this graph, all paths lead toward a root node, ensuring efficient and loop-free communication [52].

RPL uses several control messages to manage its network. For instance, the root node sends a DODAG Information Object (DIO) message to other nodes, which helps them determine their rank and position in the graph. Nodes reply with a Destination Advertisement Object (DAO) to inform the root node they are available, and these messages help maintain the network's hierarchy and connectivity.

There are two modes in RPL [53]:

- Storing Mode, where each node keeps a full routing table, knowing how to reach every other node.
- Non-Storing Mode, where only the border router has the complete routing table, and other nodes just maintain a list of parent nodes to forward data to the root.

RPL is highly adaptable, using objective functions to decide the best paths based on factors like energy efficiency or security. This makes it ideal for various low-power wireless networks, ensuring reliable communication with minimal resource use [54].

CARP

The Channel-Aware Routing Protocol (CARP) is a distributed protocol designed for underwater wireless sensor networks, enabling multi-hop data delivery to a sink node. It is also suitable for IoT due to its lightweight data packets. CARP selects forwarding nodes based on link quality, determined by the success rate of previous data transmissions between neighboring sensors [55].

There are two main processes in CARP:

- Network Initialization, where the sink node broadcasts a HELLO packet to all other nodes.
- Data Forwarding, where packets are sent hop-by-hop from sensors to the sink, with each next hop decided independently.

One limitation of CARP is that it doesn’t reuse previously collected data, which can be inefficient for applications that only need updated sensor data. To address this, an enhanced version, E-CARP, allows the sink to store old sensor data and request updates only when needed, reducing communication overhead significantly [56].

CORPL

CORPL (Cognitive RPL) is an extension of the RPL protocol, designed for cognitive radio networks. It retains the Directed Acyclic Graph (DAG) structure of RPL but introduces key changes to make it suitable for cognitive environments. CORPL uses opportunistic forwarding, meaning each node selects multiple potential next-hop neighbors (a forwarder set) instead of just one, and then coordinates to ensure that the best receiver forwards the packet [57].

The DAG construction in CORPL follows the same process as RPL. When a gateway node detects an available channel, it sends a Destination Information Object (DIO) message. Nodes use this message to update their neighborhood information and build their forwarder set, prioritizing neighbors dynamically based on their updated information [58]. This approach allows more flexible and reliable packet forwarding in dynamic networks.

QUIC

QUIC (Quick UDP Internet Connections) is a connection-oriented protocol developed by Google to improve the speed and reliability of internet connections. It works over UDP, allowing faster and more secure communication between two endpoints. QUIC ensures low-latency connections by using encryption to protect data, similar to the security provided by TLS [59].

Unlike TCP, QUIC allows multiple streams of data to be transmitted over a single connection, reducing delays and improving performance. It also incorporates congestion and flow control mechanisms to handle network traffic efficiently [60]. The Internet Engineering Task Force (IETF) is working to standardize QUIC, and it's being widely adopted by major browsers and servers for faster and more stable web connections, particularly in high-latency environments like mobile networks [61].

UIP

The UIP TCP/IP stack is a compact implementation of the TCP/IP protocol designed for very small microcontrollers, even 8-bit ones. Written in the C programming language, it requires only a few kilobytes of memory and operates with very limited RAM [62]. Despite its small size, it includes essential protocols like IP, ICMP, UDP, and TCP, allowing communication with other lightweight stacks. Its minimal design makes it ideal for embedded systems using low-end 8 or 16-bit microcontrollers, offering a much smaller footprint compared to typical TCP/IP stacks [63].

Aeron

Aeron is an open-source messaging system designed for high throughput and low- latency data streaming, making it ideal for applications that require fast and efficient communication (as shown in figure 07, like sending large files or handling real-time data. It works over UDP unicast and multicast, and because it operates without a broker, it's widely used in industries like financial services for building trading systems.

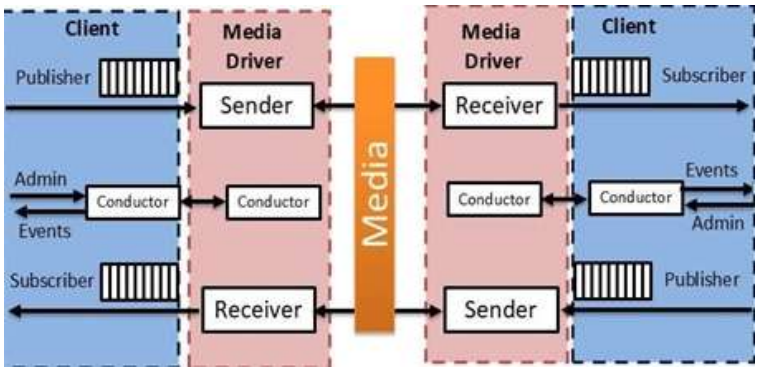

Figure 7: Aeron Architecture.

Its predictable, high performance makes Aeron perfect for applications where low-latency and high throughput are crucial [64]

CCN

Content-Centric Networking (CCN), also known as Information-Centric Networking (ICN), introduces a new approach to communication where requests are made for named content instead of sending packets to specific hosts. Two popular ICN architectures are Named Data Networking (NDN) and CCNx [65].

In CCN, a requester, called the "consumer," sends an "interest" request for content. Any node that has the requested content can respond by sending the "data" back to the consumer along the same path. This architecture shifts the focus from host-based communication to content-based communication.

The CCN project, developed by PARC (Palo Alto Research Center), aims to create a more flexible, efficient, and scalable communication model. It reduces the complexity of traditional systems and is designed to work seamlessly with existing communication technologies [66].

CCN uses two main messages: "interest" and "data." When a consumer node needs content, it sends an interest message, and the provider node responds with the requested data. Each node in the network maintains data structures like the Pending Interest Table (PIT), Forwarding Information Base (FIB), and Content Store (CS) to manage and forward these messages effectively [67].

NanoIP

NanoIP is a lightweight protocol suite designed for tiny devices like sensors and embedded systems, providing Internet-like networking without standard TCP/IP complexity. It offers two main communication methods: nanoIP for reliable connections and nanoUDP for connectionless communication, acting as functional alternatives to TCP and UDP [68].

NanoIP simplifies embedded networking by focusing on local subnets and shifting more complex tasks to Internet gateways. It includes familiar services such as unreliable datagrams (nanoUDP), connection-oriented streams (nanoTCP), and a socket-compatible interface, making it easier for developers to adopt. The protocol is ideal for small-scale, low-power environments like sensor networks, smart dust applications, peer-to-peer pervasive networks, and home networks [69].

The main goal of NanoIP is to optimize networking for small devices, offering minimal overhead while supporting wireless networking and local addressing.

TSMP

The Time Synchronized Mesh Protocol (TSMP) is a protocol stack designed for Wireless Sensor Networks (WSNs) that prioritizes reliability, security, timely data delivery, and low power consumption. It operates within a managed network, where a centralized controller schedules communication, preventing packet collisions and reducing unnecessary data overhearing [70].

TSMP is designed to ensure [71]:

- Reliability with low power consumption – Achieving over 99.9% packet delivery with long node battery life.
- Scalability – Supporting hundreds of nodes in a mesh and thousands in the same radio frequency (RF) environment.
- Flexibility – Accommodating various traffic patterns from different nodes.

- Security – Ensuring all transmitted packets' confidentiality, integrity, and authenticity.
- Environmental adaptability – Operating effectively in extreme temperatures (-40°C to 85°C) and fluctuating RF noise levels.

Despite concerns about scalability and network-wide synchronization, real-world testing has proven TSMP effective in maintaining these capabilities within managed networks [72].

Network Layer

Five network protocols are introduced for the application layer: Wi-Fi, Bluetooth, ZigBee, Z- Wave, and LoRaWAN.

WiFi

Wi-Fi is a popular choice for IoT connectivity because it provides fast, wireless communication and easy setup. It operates using the IEEE 802.11 standard, which is primarily for local area networking (LAN), offering in-building broadband coverage [73]. Wi-Fi allows devices to connect wirelessly within a range of 100 to 150 meters, although actual ranges can be shorter.

Wi-Fi enables wireless networking between multiple devices, not just internet access, but the two are often used together. Devices like laptops, smartphones, cameras, and more come equipped with built-in Wi-Fi interfaces, making it widely compatible with existing infrastructure. This makes the deployment and scaling of IoT networks easier and more cost-effective.

One of the key advantages of Wi-Fi for IoT is its high data transfer rate, which is essential for applications like video streaming from security cameras. Wi-Fi also offers widespread availability, making it simple to integrate new devices into homes, offices, or industrial settings. Its extensive range and continuous improvements, like enhanced security features, make it suitable for various use cases, from smart homes to industrial monitoring systems [74].

Bluetooth

Bluetooth is a wireless Personal Area Network (WPAN) technology designed for short- range communication, initially developed by Ericsson in 1994. It allows devices within 10 meters to share up to 721 Kbps of data using the 2.4 GHz ISM band, making it an alternative to cables for connecting portable and fixed devices [75].

Bluetooth Low Energy (BLE), also known as "Bluetooth Smart," was introduced in 2010 as an energy-efficient version of Bluetooth. BLE is ideal for applications requiring low power and short-range communication, such as healthcare, fitness, and security devices. BLE operates with a shorter range but is highly energy-efficient, making it suitable for devices like fitness trackers and smartwatches. It uses 40 channels with a 2 MHz bandwidth and supports quick transfer of small data packets at 1 Mbps [76].

In BLE, devices operate as masters or slaves, with the master (e.g., a phone or PC) managing connections to multiple slave devices (e.g., wearables or sensors). Slaves typically remain in sleep mode to conserve power, only waking up periodically to communicate with the master. Unlike classic Bluetooth, which stays connected continuously, BLE is designed for low-duty cycles, transmitting small packets quickly and efficiently. Its energy efficiency is significantly higher than other protocols like Zigbee, making it 2.5 times more efficient in some cases [77].

ZigBee

ZigBee is a wireless technology designed for controlling sensors, based on the IEEE 802.15.4 standard for low-power, low-cost wireless personal area networks (WPAN). It is widely used for applications like controlling and monitoring over distances ranging from 10 to 150 meters, making it simpler and cheaper than other short-range networks like Bluetooth and Wi-Fi. ZigBee consumes very little power, about 1mW, and operates on different frequencies depending on the region: 868 MHz in Europe, 915 MHz in North America and Australia, and 2.4 GHz worldwide. Its data rates vary from 20 Kbps to 250 Kbps [78].

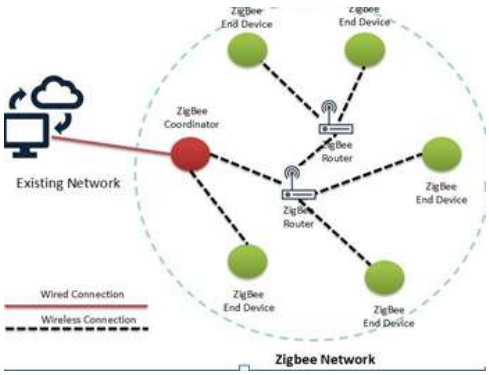

Figure 8: Zigbee (A Working Model).

ZigBee networks are made up of three types of devices: coordinators, routers, and end devices, as shown in Figure 08. The coordinator sets up the network and selects the radio channel, the router manages data transmission, and the end devices remain off most of the time to save power, only waking periodically to check for data from the router [79].

ZigBee's mesh network topology allows any node to communicate with others within range, making it highly reliable for wireless sensor networks. Enhanced features include data encryption, authentication, routing, and forwarding, providing security for applications like smart energy monitoring, building automation, smart healthcare, and smart lighting.

ZigBee is supported by the ZigBee Alliance and continues to evolve, with some versions even allowing energy harvesting in the absence of batteries. While it's globally reliable, researchers suggest improvements in power consumption. The integration of 5G technology into smart devices can enhance ZigBee's performance, particularly in data transmission [80].

Z-Wave

Z-Wave is a low-energy radio frequency (RF) technology designed for sub-GHz communications, primarily used in home automation, security systems, and lighting controls. It operates on the 900 MHz frequency band and supports data rates of 9.6, 40, and 100 kbps [81]. Z-Wave is a simple and efficient mesh networking protocol, allowing fast and easy development. It is highly scalable and doesn't require a coordinator node, making it suitable for managing multiple connected devices in a network.

Z-Wave is particularly well-suited for smart homes and small commercial IoT applications, enabling point-to-point communication over distances of around 30 meters. It's often used for small data transmissions, such as controlling lights, managing energy usage, or monitoring wearable health devices. It relies on a master/slave architecture where the master manages communication and scheduling for the connected slave devices [82].

Originally developed by Zensys, Z-Wave allows seamless wireless communication between devices, with a focus on home automation. It enables users to monitor and control lighting, temperature, and security systems via tablets, smartphones, or computers. Z-Wave's mesh network topology ensures that devices can send and receive messages from any other connected device, making it a robust solution for smart home environments [83].

LoRaWAN

LoRaWAN is a Media Access Control (MAC) protocol designed for large-scale public networks managed by a single operator. It uses LoRa as its physical layer, which spreads data across multiple radio channels and transmission rates using coded messages instead of narrowband transmission [84]. LoRaWAN defines three classes of end devices, each supporting duplex communication but differing in energy requirements and downlink latency based on their application needs.

LoRaWAN is ideal for large networks with hundreds, thousands, or even millions of nodes, making it suitable for smart city applications like street lighting, waste management, and other IoT services. This low-cost, secure protocol supports long- range, bi-directional transmissions, designed for energy efficiency and extended battery life.

As a Low Power Wide Area Network (LPWAN) technology, LoRaWAN is optimized for long-range communication with lower energy consumption, in contrast to short- range cellular networks. It uses a star topology where end devices communicate with gateways, and these gateways then relay messages to a central server. LoRaWAN is developed and maintained by the LoRa Alliance, a non-profit organization that promotes the technology [85].

Physical Layer

The physical layer is responsible for receiving raw data between devices and transmission media, converting digital information into electrical, radio, or optical signals. It includes sensors that gather information from the environment by detecting physical parameters or identifying other smart objects.

- The IEEE 802.15.4 standard was created for Low-Rate Wireless Personal Area Networks (LR-WPANs), focusing on low data rates and energy-efficient communication, making it ideal for applications like wireless sensor networks (WSNs) [86]. Released in 2003, it defines the physical layer (PHY) and media access control layer (MAC), providing a foundation for many widely used communication protocols in industry and research [87].

Over time, the standard has evolved to support a wide range of applications such as smart metering and vehicular communication. Its low-cost, short-range communication allows devices to cooperate for multi-hop routing, extending the communication range. This standard remains key for power-constrained devices, enabling reliable communication in various sensor-based services [88].

THE DESIGN OF A FRAMEWORK IOT FRAMEWORK DESIGN GOALS

An IoT framework provides a solid base for building applications. By using a framework, developers gain access to a variety of features and tools that simplify the creation of IoT solutions and smart products. Instead of starting from scratch, these frameworks allow you to develop solutions faster, helping you bring products to market quickly and maintain a competitive edge [89].

Both open-source and proprietary frameworks are available. While open-source doesn’t mean everything is free, it means the source code is accessible for improvement and use by anyone. Proprietary platforms like those from Amazon Web Services and Google Cloud are paid, offering additional services.

Over the next decade, IoT frameworks will evolve with four main design goals [90]: (1) speeding up development and time to market; (2) reducing the complexity of deploying and managing IoT networks; (3) improving application portability and interoperability; and (4) enhancing reliability, serviceability, and long-term support. Given the vast range of communication technologies available today, it doesn't make sense for applications to manage every connection option. Frameworks simplify this by abstracting connectivity using methods like REST and publish-subscribe.

To meet these challenges, IoT frameworks rely on standardization. Standards organizations define the design principles, architecture, and connection methods for IoT systems. They also provide reference implementations, including source code, which speeds up the development process by offering pre- tested solutions that meet compliance requirements. This reference code is easier to maintain and improves over time through contributions from the open-source community.

IoT frameworks simplify network management by hiding the technical details, allowing developers to focus on node interactions rather than complex connectivity. Applications can be built using high-level languages like Node.js, making them portable across different platforms. Frameworks expose a rich data model that supports various IoT use cases, from home automation to industrial control. Once developed, applications can run on any platform that supports the framework, ensuring compatibility even across different operating systems and hardware [91].

These frameworks also ensure that devices from different vendors, running various operating systems, can communicate seamlessly. By hiding the complexity of network management, frameworks enable faster deployments. They also support easy system updates, such as firmware updates, across the network [92].

Furthermore, IoT frameworks can handle both streaming data and big data. Streaming data needs to be processed immediately for quick decision-making, while big data involves large datasets analyzed over time. The fusion of data from various sources is crucial for time-sensitive IoT applications, as it provides accurate insights for real-time decisions.

FRAMEWORK DESIGN REQUIREMENTS

The design of the framework must meet the following key requirements [93]:

- Generality: The framework should be able to connect to various sensors without focusing on specific drivers. While individual sensor drivers are outside the framework's scope, we define necessary specifications for drivers. Generality is achieved by implementing drivers using Arduino.
- Connectivity: Smart devices, like smartphones, must connect to smart appliances regardless of network environment. Typically, a private LAN is used at home, but smartphones need to connect both inside and outside of it. To address this, the framework enables smartphones and smart appliances to connect via a cloud-based hub, ensuring seamless connectivity.
- Interoperability: The framework should not restrict devices to only connect to services from the same manufacturer. It should work across devices and services from different manufacturers. Interoperability is achieved by adhering to device specifications.

- Security: The framework must protect smart appliances from malicious attacks by incorporating encryption. With the growing trend of IoT, security is a significant concern, as network-connected devices can be exploited for malicious purposes like spamming. Given the expected increase in connected devices, security in this framework is a top priority.
- Real-Time Response: To use smartphones as remote controllers for smart appliances, real-time responsiveness is essential. Although smart appliances tend to have high latency, the framework aims for a fast enough response time to avoid noticeable delays, targeting around 400ms similar to browser- based games.
- Efficiency: The framework must ensure efficiency, minimizing waste of performance and power, especially on low-end devices. A low-power ARM Linux machine, like Raspberry Pi, is used as the hardware platform to achieve this.

Additional system requirements include [94]:

- Proactive Self-Adaptation: The platform should be capable of analyzing data and adapting the system in problematic situations. In resource-constrained environments, any planned adaptation must be validated to ensure necessity.
- Uncertainty Management: Since not all events that trigger system reconfiguration can be predicted, the platform must compare current system performance with user expectations and adjust accordingly.
- Variability Management: The platform must manage variability at different levels within a fleet of connected devices throughout the system's lifecycle.
- Physical Abstraction: Though not the focus here, the platform should also support communication with various devices and technologies for monitoring and control.
- These requirements lay the foundation for a robust, secure, and adaptable IoT framework capable of handling real-time interactions across multiple devices and networks.

SIMPLIFIED IOT FRAMEWORK DESIGN

For an IoT framework to be reliable, it must meet certain basic criteria that enable integration and interoperability. These frameworks play a vital role in both academic and organizational research focused on connecting devices in the IoT space [95] As the IoT landscape continues to evolve, we suggest several essential measures that these frameworks should address [96]:

- Contract Decoupling: An IoT system consists of various devices using different communication protocols. The framework should allow service producers and consumers to evolve independently without breaking their contract. For example, if a service sends data in JSON format but the consumer needs it in XML, the framework should convert the data appropriately.
- Scalability: With the rapid growth of IoT, the integration framework must be scalable to support the billions of devices that will soon connect to the internet.
- Ease of Testing: The framework should facilitate easy testing and debugging. This includes support for different types of testing, such as integration, component, system, compatibility, installation, functional, non-functional, performance, and security testing.

- Ease of Development: The framework should simplify the development process, making it accessible for developers and non-developers alike. Comprehensive documentation should help users understand how to work with the framework without being overwhelmed by complexity.
- Fault Tolerance: IoT systems must be reliable and resilient. The integration framework should manage faults effectively, especially since IoT devices frequently switch between online and offline states. It should include self-healing capabilities for various types of faults, such as network issues, unauthorized access, and server crashes.
- Lightweight Implementation: The integration framework should be easy to install, update, and adapt, with minimal overhead during development and deployment.
- Service Coordination: The framework should support service coordination, which involves orchestrating and chaining services to perform specific tasks. This can be done through either orchestration, where a mediator coordinates services, or choreography, where services interact directly.
- Inter-Domain Operability: The framework should also support communication across different domains. For instance, a smart car system should be able to interact with traffic signals or road conditions in a smart city.

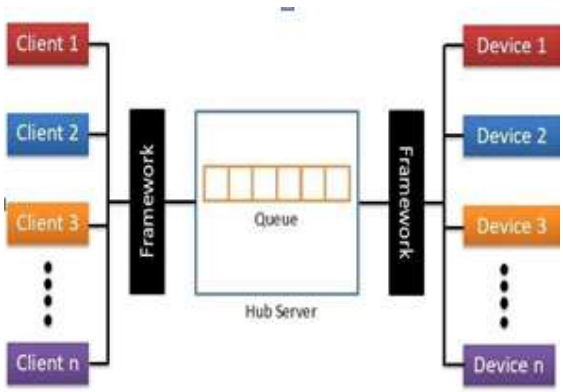

Figure 9: Overview of The System.

System Overview

In designing a system that meets these requirements, the proposed architecture (as shown in Figure 09) includes the following components [97]:

- Client (Smart Appliance): Users control smart appliances through smartphone apps or web applications. The web service implements the smart appliance's business logic. However, smartphones cannot connect directly to home appliances.
- Hub Service: Since devices are on a private local area network (LAN), they cannot be accessed directly from the client or cloud. The hub server facilitates communication between the devices and the web services, acting as a messaging queue. It remains passive until clients and devices connect to it, allowing them to communicate even when separated by a firewall. Unlike other systems, we use TCP for reliable communication between devices, and since the hub is implemented as a web service, it can easily communicate through HTTP/HTTPS, making it firewall-friendly [98].

- Device: Each smart appliance connects to the home LAN and is assigned an address through DHCP. Once connected, the device can communicate with the hub server and become operational.

This framework simplifies the interaction between diverse devices, allowing for better integration and management within the IoT ecosystem.

OPEN SOURCE IOT PLATFORMS

1. VSCP (Very Simple Control Protocol) [99] Origin Date: 2006

Founders/Organization: Created by Brad Fauth with the support of the VSCP project community.

Development Details

VSCP is an open-source communication protocol made to simplify how devices in IoT and automation systems communicate. The focus of this protocol is on ease of use and flexibility. It allows devices to share control data easily. It can work with different types of devices and applications, making sure they can function together smoothly, regardless of hardware or software differences.

Key Features for Cross-Domain Use:

- Standardized Communication: Establishes a common way for devices to share information.
- Protocol Agnostic: Works well with a variety of communication protocols.
- Flexible: Can be easily modified to support new devices and functionalities.
- Interoperability: Ensures smooth connections between different IoT systems.

2. Open PLC [100] Origin Date: 2008

Founders/Organization: Created by Thiago Alves with the support of the OpenPLC Project team.

Development Details

OpenPLC is an open-source programmable logic controller (PLC) designed as a cost-effective alternative to traditional PLCs used in industrial automation. It supports standard PLC programming languages and can be used in a variety of industrial and IoT environments, helping automate and control different systems across multiple sectors. Key Features for Cross-Domain Use:

- Standard Protocol Support: Works with widely-used communication protocols like Modbus, MQTT, and OPC UA.
- Interoperable Systems: Connects easily to many industrial and IoT platforms.
- Modular Architecture: Custom modules can be added to increase its capabilities.
- Community-Driven Development: Ongoing support from an active community ensures it's regularly updated and compatible with a broad range of systems.

3. Thing Speak [101] Origin Date: 2010

Founders/Organization: Developed by MathWorks. Development Details

ThingSpeak is an open-source platform built on MATLAB, made for analyzing and managing data from IoT devices. It allows users to collect sensor data in real-time, analyze it using MATLAB, and display the results through customizable dashboards. ThingSpeak integrates with various hardware devices and provides APIs for data access and control. Key Features for Cross-Domain Use:

- Real-Time Data Collection: Gathers sensor data from multiple sources instantly.
- MATLAB Integration: Offers powerful tools for analyzing and visualizing data.
- Customizable Dashboards: Lets users create flexible interfaces for different applications.
- API Support: Provides RESTful APIs for easy integration with other platforms and services.

4. Open HAB [102] Origin Date: 2010

Founders/Organization: Initially created by Kai Uwe Broulik, Michael Jackson, and the openHAB community.

Development Details

OpenHAB (Open Home Automation Bus) is an open-source platform designed for smart home automation. It brings together various smart devices and technologies under one system. Its flexible setup allows for controlling devices from many different manufacturers by supporting a wide range of connections.

Key Features for Cross-Domain Use:

- Extensive Binding Library: Works with over 200 device types and services.
- Protocol Flexibility: Compatible with Zigbee, Z-Wave, MQTT, HTTP, and other protocols.
- Rule Engine: Lets users create custom automation rules for multiple devices.
- Community Support: A large, active community constantly adding new integrations and improvements.

5. Open Remote [103] Origin Date: 2012

Founders/Organization: Built by the OpenRemote.org community, co-founded by Jerome van de Geer.

Development Details

OpenRemote is an open-source platform made for smart homes and buildings, allowing easy automation and integration of various IoT devices. It supports many communication protocols and provides tools for creating automation rules, designing custom dashboards, and controlling devices through mobile apps.

Key Features for Cross-Domain Use:

- Protocol Compatibility: Works with Zigbee, Z-Wave, MQTT, HTTP, and more.
- Rule-Based Automation: Enables complex automation across different devices and systems.
- Custom Dashboards: Create tailored control interfaces for different setups.

- API Integrations: Can connect with external services and platforms for added features.

6. Kaa IoT Platform [104] Origin Date: 2012

Founders/Organization: Created by KaaIoT, founded by Roman Shaburov and Anton Makarov. Development Details

Kaa is a flexible open-source IoT platform designed for managing devices, gathering data, performing analytics, and integrating systems. It works with a variety of devices and offers software development kits (SDKs) in different programming languages to make device connections and data sharing easier. The platform is built to scale, so it can handle everything from small projects to large deployments, and it allows developers to customize it to fit their unique needs.

Key Features for Cross-Domain Use:

- Multi-Protocol Support: Works with MQTT, HTTP, CoAP, and other custom protocols.
- SDK Availability: Offers SDKs for different programming languages and platforms.
- Scalable Architecture: Can be used for both small and large projects in different fields.
- Flexible Data Models: Can handle various types of data from different sources.

7. Node-Red [105] Origin Date: 2013

Founders/Organization: Created by Nick O'Leary and Dave Conway-Jones as part of IBM's Emerging Technology group.

Development Details

Node-RED is a tool for visual programming that helps connect hardware, APIs, and web services. Built on Node.js, it uses a browser-based interface where users can simply drag and drop elements to build workflows. With a large library of pre-made nodes, it's easy to extend and is commonly used for quickly setting up IoT integrations and prototypes. Key Features for Cross-Domain Use:

- Large Node Library: Offers thousands of pre-built nodes for a wide range of tasks.
- Protocol Flexibility: Works with MQTT, HTTP, WebSockets, and other protocols.
- Platform Flexibility: Can run on many platforms, including Raspberry Pi, cloud services, and desktop systems.
- Custom Integration: Allows users to create custom nodes for connecting different systems.

8. Thinger.io [106] Origin Date: 2013

Founders/Organization: Created by Thinger.io Inc., founded by Ivo Marušić. Development Details

Thinger.io is an open-source IoT platform that helps developers connect, manage, and monitor their devices. It supports RESTful APIs, works with various programming languages, and is compatible with hardware like Arduino and Raspberry Pi. The platform is designed to be user-friendly, offering real-time data visualization, device management, and easy integration with other services.

Key Features for Cross-Domain Use:

- Protocol Support: Works with MQTT, HTTP, and WebSockets.
- Multi-Language Compatibility: Supports several programming languages.

- Device Management: Centralized control for different types of devices.
- Real-Time Data Visualization: Customizable dashboards for various applications.

9. Zetta [107] Origin Date: 2013

Founders/Organization: Developed by Eran Hammer and colleagues at the University of Washington.

Development Details

Zetta is a framework built on Node.js for creating real-time web applications and IoT systems. It uses a RESTful architecture, allowing developers to build APIs and data streams that easily work with web technologies. Zetta is ideal for applications that need real-time data updates and interactive user interfaces, using WebSockets to maintain constant connections. Key Features for Cross-Domain Use:

- RESTful Architecture: Makes it easy to integrate with different web services and platforms.
- Real-Time Communication: Uses WebSockets to keep connections open for live data exchange.
- Extensible Drivers: Allows custom drivers to connect various devices and services.
- Cloud Integration: Easily links with cloud platforms to enhance functionality.

10. AllJoyn Framework [108] Origin Date: 2013

Founders/Organization: Initially developed by Qualcomm, now managed by the Open Connectivity Foundation (OCF).

Development Details

AllJoyn is an open-source framework that allows devices and applications to find and communicate with each other, no matter the manufacturer or operating system. It focuses on making different platforms work together and supports a variety of devices and communication protocols. AllJoyn is useful for applications like smart homes, media sharing, and device control.

Key Features for Cross-Domain Use:

- Cross-Platform Compatibility: Operates on different operating systems and device types.
- Rich API Set: Makes integration across various domains easier.
- Interoperability: Bridges different ecosystems and technologies for smooth interaction.
- Device Discovery and Communication: Helps diverse devices connect and communicate effortlessly.

11. Home Assistant [109] Origin Date: 2013

Founders/Organization: Created by Pavle Krstic and the Home Assistant community. Development Details

Home Assistant is an open-source platform designed for home automation, emphasizing user privacy and local control. It connects with a wide variety of smart devices and services, offering a simple interface for managing them. The platform also allows users to set up advanced automations that link different devices and services across various systems.

Key Features for Cross-Domain Use:

- Broad Integration: Supports thousands of devices and services from different platforms.
- Advanced Automations: Set up automations that work with multiple devices and services at once.
- Custom Components: Extend the platform's capabilities by adding integrations for new devices.
- Simple Interface: Provides a unified dashboard for controlling and monitoring devices.

12. Site Where [110] Origin Date: 2014

Founders/Organization: Developed by SiteWhere LLC, founded by Derek Collison. Development Details

Site Where is an open-source platform built to create flexible and scalable IoT solutions. It offers features like device management, data collection, event processing, and rule-based workflows. The platform supports multiple users or organizations (multi-tenancy), allowing each to have a separate environment. Even after being acquired by Helena IoT in 2020, Site Where's open- source version continues to receive support from the community. Key Features for Cross-Domain Use:

- Multi-Tenancy: Allows different users or organizations to have separate spaces on the platform.
- Comprehensive Device Management: Works with a variety of devices and protocols.
- Event Processing: Handles complex events across different systems.
- Integration: Easily connects to other data sources and third-party services.

13. IoTivity [111] Origin Date: 2014

Founders/Organization: Developed by the Open Connectivity Foundation (OCF). Development Details

IoTivity is an open-source framework created to connect IoT devices smoothly. It is designed to ensure devices can communicate with each other easily, regardless of platform or protocol, while focusing on security and scalability. IoTivity offers APIs for discovering devices, exchanging data, and managing devices remotely, making it ideal for building interconnected IoT systems.

Key Features for Cross-Domain Use:

- Interoperability: Uses OCF standards to ensure compatibility across devices.
- Multi-Platform: Works with Android, Linux, Windows, and others.
- Security: Prioritizes secure communication between devices.
- Extensible APIs: Can be integrated with various platforms and services.

14. DeviceHive [112] Origin Date: 2014

Founders/Organization: Developed by the DeviceHive team, co-founded by Alexander Shpak. Development Details

DeviceHive is a versatile open-source IoT platform that helps manage devices, gather data, and enable real-time communication. It supports multiple communication protocols such as MQTT and WebSockets and provides REST APIs for easy connectivity. DeviceHive is built to be scalable and adaptable, allowing seamless integration with different databases and cloud services. Its modular design makes it customizable for specific IoT needs. Key Features for Cross-Domain Use:

- Multi-Protocol Support: Compatible with MQTT, WebSockets, and HTTP.
- RESTful APIs: Easy to integrate with other platforms and services.
- Scalable: Works well for projects of any size across various domains.
- Extensible: Modular design allows adding new features to connect different systems.

15. PlatformIO [113] Origin Date: 2014

Founders/Organization: Developed by the PlatformIO team, founded by Ivan Kravets. Development Details

PlatformIO is an open-source ecosystem designed for IoT development. It offers a cross- platform IDE (Integrated Development Environment), a library manager, and a build system. It supports many embedded boards and frameworks, helping developers with firmware development and project management. PlatformIO works with popular code editors like Visual Studio Code and Atom, and it includes features like smart code completion, debugging, and continuous integration for smoother development.

Key Features for Cross-Domain Use:

- Cross-Platform IDE: Works on Windows, macOS, and Linux.
- Board Compatibility: Supports a large variety of embedded boards and microcontrollers.
- Library Manager: Makes it easy to manage and add different libraries and frameworks.
- Continuous Integration: Allows automated testing and deployment across various platforms.

16. WSO2 IoT [114] Origin Date: 2014 (initial release as WSO2 IoT Server)

Founders/Organization: Developed by WSO2, an open-source tech company founded by Sanjiva Weerawarana.

Development Details

WSO2 IoT Server is an open-source platform designed for managing IoT devices, analyzing data, and building applications. It integrates well with WSO2's other middleware solutions and offers features like security, scalability, and flexibility. The platform supports many types of devices and operating systems, helping businesses manage IoT devices and use the data to make smart decisions.

Key Features for Cross-Domain Use:

- Device Management: Handles different devices from various domains easily.
- Data Analytics: Offers real-time data analysis and visualization for valuable insights.
- Security Management: Strong security measures to keep devices and data safe.
- Extensibility: Modular design makes it easy to integrate with different platforms and services.

17. Eclipse Kura [115] Origin Date: 2015

Founders/Organization: Developed by the Eclipse Foundation, initially created by CNR-IEIIT (Italian National Research Council - Institute for Embedded Systems and Information Technology).

Development Details:

Eclipse Kura is an open-source framework designed for IoT gateways and edge computing. It offers essential services like device management, network setup, and data communication using protocols like MQTT and HTTP. Built using Java and OSGi, Kura is highly flexible and scalable, allowing developers to add features through its modular design. Key Features for Cross-Domain Use:

- Modular Architecture: Makes it easy to add and integrate different services and protocols.
- Data Connectivity: Supports multiple communication methods like MQTT and HTTP.
- Device Management: Simplifies managing connected devices across various domains.
- Web-Based Administration: Offers user-friendly web interfaces for easy monitoring and control.

18. EdgeX Foundry [116] Origin Date: 2015

Founders/Organization: Developed by a community of industry partners and hosted by the Linux Foundation.

Development Details:

EdgeX Foundry is an open-source platform created to support edge computing in IoT. It is vendor-neutral and has a flexible design, making it easy to connect different hardware and software components. The platform is focused on being interoperable, allowing it to work well across various domains and applications. Its architecture is scalable, making it suitable for many types of IoT projects.

Key Features for Cross-Domain Use:

- Microservices Architecture: Makes it easy to integrate different technologies and components.
- Protocol Flexibility: Works with many communication protocols and standards.
- Interoperability: Ensures compatibility with a wide range of devices and systems.
- Extensible Framework: Easily customizable to add new features or services.

19. OpenHIM (Open Health Information Mediator) [117] Origin Date: 2015

Founders/Organization: Created by Jembi Health Systems and the OpenHIM Community. Development Details:

OpenHIM is an open-source middleware designed to help different systems share and manage data, especially in health information systems. However, it can also be used for IoT applications that need to connect data across multiple domains. It helps integrate data from different sources securely and ensures smooth communication between systems.

Key Features for Cross-Domain Use:

- Data Mediation: Helps combine and process data from various sources.
- Interoperability: Works with many data formats and communication methods.
- Security and Governance: Ensures data exchange is secure and well-managed.
- Scalable Middleware: Provides a strong foundation for connecting diverse systems across domains.

20. ThingsBoard [118] Origin Date: 2016

Founders/Organization: Created by ThingsBoard Inc., founded by Ales Ponkratov and Sergey Yatsenko.

Development Details:

ThingsBoard is a flexible and scalable IoT platform that handles data collection, visualization, and device management. It can be deployed in the cloud or on-premises and supports various protocols. The platform has a rule engine for event processing, customizable dashboards, and strong integration features, making it suitable for many applications.

Key Features for Cross-Domain Use:

- Protocol Support: MQTT, CoAP, HTTP.
- Integration: REST APIs, Rule Engine for advanced events.
- Multi-Tenancy: Allows multiple users/organizations.
- Dashboard Customization: Adaptable for different domains.

21. Mainflux [119] Origin Date: 2016

Founders/Organization: Developed by the Mainflux team, led by Antti Rautiainen. Development Details:

Mainflux is an open-source IoT platform that focuses on secure and scalable device connectivity, data management, and app development. It supports multiple communication protocols and uses a microservices architecture, making it highly scalable and resilient. Mainflux offers features like real-time data streaming, easy device management, and cloud service integration, making it versatile for different IoT projects.

Key Features for Cross-Domain Use:

- Protocol Support: Works with MQTT, HTTP, WebSockets, and more.
- Microservices Design: Makes it easy to integrate across different services and domains.
- Device Management: Centralized control of various devices.
- Interoperability: Seamless integration with other platforms and cloud services.

22. Zigbee2MQTT [120] Origin Date: 2016

Founders/Organization: Created by Koenkk and the Zigbee2MQTT community. Development Details:

Zigbee2MQTT is an open-source project that lets you control Zigbee devices using MQTT, making it easy to connect Zigbee devices to MQTT-based IoT systems. This project bridges the gap between Zigbee and MQTT, allowing different devices to work together in one platform.

Key Features for Cross-Domain Use:

- Protocol Bridging: Links Zigbee devices with MQTT platforms.
- Broad Device Support: Works with many Zigbee devices from various brands.
- Flexible Integration: Can be used with popular platforms like Home Assistant and Node- RED.

- Community-Driven: A strong, active community ensures compatibility and ongoing updates.

23. Homie Convention [121] Origin Date: 2016

Founders/Organization: Created by the Homie Convention Community. Development Details:

The Homie Convention is a lightweight communication standard for IoT devices using MQTT. It helps different devices follow the same communication rules, making it easier to connect various devices to MQTT-based platforms.

Key Features for Cross-Domain Use:

- Standardized Messaging: Ensures devices communicate in a consistent way.
- Easy Integration: Simplifies connecting different devices to MQTT systems.
- Cross-Platform Support: Compatible with many hardware and software platforms.
- Extensible Schema: Allows adding custom features to suit different needs across platforms.

24. Zetta [122] Origin Date: 2013

Founders/Organization: Created by Eran Hammer and team at the University of Washington. Development Details:

Zetta is a framework built on Node.js, designed to help developers create scalable, real-time web applications and IoT systems. It uses a RESTful design, making it easy to create APIs and data streams that integrate well with web technologies. Zetta is ideal for apps that need real- time data updates and interactive interfaces, using WebSockets to keep connections open for continuous data flow.

Key Features for Cross-Domain Use:

- RESTful Design: Makes it easy to connect with different web services and platforms.
- Real-Time Communication: Uses WebSockets to keep data flowing in real time.
- Extensible Drivers: Allows developers to add drivers for different devices and services.
- Cloud Integration: Can easily connect with cloud platforms for more features.

25. Macchina.io [123] Origin Date: 2013

Founders/Organization: Created by the Macchina.io team, led by Luigi Ronga. Development Details:

Macchina.io is an open-source platform for the Internet of Things (IoT) that helps developers build, manage, and expand their IoT applications. It provides options for storing data, RESTful APIs for communication, and the ability to connect with many different devices and services. The platform is designed to be flexible and user-friendly, making it easy to quickly develop and launch IoT solutions for various needs.

Key Features for Cross-Domain Use:

- Flexible Data Storage: Offers different storage options that can fit many different applications.
- RESTful APIs: Makes it simple to connect with various platforms and services.
- Scalable Architecture: Built to support growth across various applications.

- Extensive Integration Capabilities: Can connect with a broad range of devices and external services.

26. Distributed Services Architecture for IoT (DSA) [124] Origin Date: 2015

Founders/Organization: Created under the Eclipse Foundation as part of the Eclipse DSA project.

Development Details:

DSA is an open-source framework built for creating distributed IoT systems using a microservices structure. It emphasizes scalability, flexibility, and the ability to work with different devices and services. It provides APIs and tools to help developers build complex IoT applications, allowing for efficient data flow and communication between devices.

Key Features for Cross-Domain Use:

- Microservices-Based: Enables easier scaling and flexibility for various applications.
- API-Driven: Simplifies integration with different platforms and services.
- Protocol Support: Works with multiple communication protocols, making it versatile.
- Interoperability: Designed to easily connect different devices and systems.

27. OpenIoT [125] Origin Date: 2012

Founders/Organization: Created as part of the European Union's OpenIoT project. Development Details:

OpenIoT is an open-source platform designed to manage and process data from IoT devices. It allows users to gather, store, and analyze data, making it easier to build IoT applications. The platform focuses on scalability and compatibility, supporting various data sources and working with other IoT services. It also includes tools for real-time data analysis, event detection, and data visualization, helping to create complete IoT solutions.

Key Features for Cross-Domain Use:

- Data Integration: Works with different data formats from multiple sources.
- Scalable Data Processing: Capable of handling large amounts of data across various domains.
- Interoperable Services: Ensures communication between different services across domains.
- API Support: Offers APIs for connecting with external platforms and services.

28. OpenPLC [126] Origin Date: 2008

Founders/Organization: Created by Thiago Alves and the OpenPLC Project. Development Details:

OpenPLC is an open-source Programmable Logic Controller (PLC) designed as a more accessible option compared to expensive, proprietary PLCs used in industrial automation. It supports standard PLC programming languages and can be easily connected to IoT and industrial systems, allowing for automation and control in a variety of fields.

Key Features for Cross-Domain Use:

- Standard Protocol Support: Works with widely used protocols like Modbus, MQTT, and OPC UA for flexible communication.

- Interoperable Systems: Can connect to many industrial and IoT platforms.
- Modular Architecture: Allows adding custom modules to expand its functionality.
- Community-Driven Development: Actively supported by a community that ensures compatibility and ongoing updates.

The above list highlights active open-source IoT applications and platforms that excel in cross-domain interoperability, enabling seamless integration and communication across diverse devices, protocols, and platforms. These solutions are well-suited for complex IoT ecosystems where integration across multiple domains is essential.

CONCLUSION

The use of open-source IoT solutions is becoming essential for integrating and managing devices across different industries. These platforms provide the flexibility needed to ensure that different systems, devices, and data sources can work together smoothly. By adopting these solutions, businesses and developers can overcome challenges related to data handling, scalability, and security, while also avoiding vendor lock-in. As the IoT ecosystem continues to grow, these open-source platforms will play a vital role in enabling efficient and cost-effective solutions for future IoT applications.

REFERENCES

1. *Sebastian S, Ray PP. Development Of Iot Invasive Architecture For Complying With Health Of Home. In: Proc: I3CS, Shillong; 2015. P. 79–83.*
2. *Gaona-Garcia P, Montenegro-Marin CE, Prieto JD, Nieto YV. Analysis Of Security Mechanisms Based On Clusters Iot Environments. Int J Interact Multimed Artif Intell. 2017;4(3):55–60.*
3. *Zhou J, Cap Z, Dong X, Vasilakos AV. Security And Privacy For Cloud-Based Iot: Challenges. IEEE Commun Mag. 2017;55(1):26–33. Https://Doi.Org/10.1109/MCOM.2017.1600363CM.*
4. *Pereira C, Aguiar A. Towards Efficient Mobile M2M Communications: Survey And Open Challenges. Sensors. 2014;14(10):19582–608.*
5. *Olivier F, Carlos G, Florent N. New Security Architecture For Iot Network. In: International Workshop On Big Data And Data Mining Challenges On Iot And Pervasive Systems (Bigd2m 2015), Procedia Computer Science, Vol. 52; 2015. P. 1028–33.*
6. *Minoli D, Sohraby K, Kouns J. Iot Security (Iotsec) Considerations, Requirements, And Architectures. In: Proc. 14th IEEE Annual Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA, 8–11 January 2017. Https://Doi.Org/10.1109/Ccnc.2017.7983271.*
7. *Gungor, V. C., Sahin, D., Kocak, T., Ergut, S., Buccella, C., Cecati, C., & Hancke, G. P. (2011). Smart Grid Technologies: Communication Technologies And Standards. IEEE Transactions On Industrial Informatics, 7(4), 529-539.*

8. *Ala Al-Fuqaha, Senior Member, IEEE, Mohsen Guizani, Fellow, IEEE, Mehdi Mohammadi, Student Member, IEEE, Mohammed Aledhari, Student Member, IEEE, And Moussa Ayyash, Senior Member, IEEE, Internet Of Things: A Survey On Enabling Technologies, Protocols, And Applications, IEEE COMMUNICATION SURVEYS & TUTORIALS, VOL. 17, NO. 4, FOURTH QUARTER 2015 2347.*
9. *M. Centenaro, L. Vangelista, A. Zanella, M. Zorzi, Long-Range Communications In Unlicensed Bands: The Rising Stars In The Iot And Smart City Scenarios, IEEE J. Wirel. Comm. 23 (5) (2016) 60–67.*
10. *D. Patel, M. Won, Experimental Study On Low Power Wide Area Networks For Mobile Internet Of Things, In: Proc. Of VTC, Sydney, Australia, 2017, Pp. 1–5.*
11. *U. Raza, P. Kulkarni, M. Sooriyabandara, Low Power Wide Area Networks: An Overview, IEEE J. Commun. Surv. Tuto. 19 (2) (2017) 855–873.*
12. *A.M. Baharudin, W. Yan, Long-Range Wireless Sensor Networks For Geolocation Tracking: Design And Evaluation, In: Proc. Of IES, Denpasar, Indonesia, 2016, Pp. 76–80.*
13. *W. Guibene, J. Nowack, N. Chalikias, M. Kelly, Evaluation Of LPWAN Technologies For Smart Cities: River Monitoring Use-Case, In: Proc. Of WCNCW, San Francisco, CA, USA, 2017, Pp. 17–22.*
14. *O. Vondrous, Z. Kocur, T. Hegr, O. Slavicek, Performance Evaluation Of Iot Mesh Networking Technology In ISM Frequency Band, In: Proc. Of ME, Prague, Czech Republic, 2016, Pp. 1–8.*
15. *Jiasong Mu And Liang Han, "Performance Analysis Of The Zigbee Networks In 5G Environment And The Nearest Access Routing For Improvement", Ad Hoc Networks, Vol. 56, March 2017, Pp. 1-12.*
16. *I.B.F. De Almeida, L.L. Mendes, J.J. Rodrigues, M.A. Da Cruz, 5g Waveforms For Iot Applications, IEEE Commun. Surv. Tutor. 21 (2019) 2554–2567*
17. *IEEE Std. 802.15.4, Wireless MAC And PHY Specifications For Low-Rate Wireless Personal Area Networks, 2003.*
18. *Jasenka Dizdarevic, Francisco Carpio, Admela Jukan And Xavi Masip-Bruin, "A Survey Of Communication Protocols For Internet-Of-Things And Related Challenges Of Fog And Cloud Computing Integration", ACM Computing Surveys, Vol.1, No.1, April 2018, Pp. 1-27.*
19. *Tara Salman And Raj Jain, "Networking Protocols And Standards For Internet Of Things", Chapter 13, John Wiley & Sons, Inc, 2017. Https://Doi.Org/10.1002/9781119173601.Ch13.*
20. *Tanya Mohan Tukade And R M Banakar, "Data Transfer Protocols In Iot-An Overview", International Journal Of Pure And Applied Mathematics, Vol. 118, No. 16, January 2018, Pp. 121-138. Url: Http://Www.Ijpam.Eu.*
21. *Nishant M. Sonawala, Bharat Tank And Hardik Patel, "Iot Protocol Based Environmental Data Monitoring", IEEE Proceedings International Conference On Computing Methodologies And Communication, 2017, Pp.1041-1045.*
22. *MQTT Essentials Part 1 To 9: Detail Study On MQTT[Online] Available At: Http://Www.Hivemq.Com/Blog/Mqttessentials-Part-6-Mqttquality-Of-Service-Levels, June 18, 2016.*

23. *Jasenka Dizdarevic, Francisco Carpio, Admela Jukan And Xavi Masip-Bruin, "A Survey Of Communication Protocols For Internet-Of-Things And Related Challenges Of Fog And Cloud Computing Integration", ACM Computing Surveys, Vol.1, No.1, April 2018, Pp. 1-27.*
24. *E. Rescorla, N. Modadugu, Rfc 6347: Datagram Transport Layer Security Version 1.2, Internet Engineering Task Force (IETF), 2012, P. 2070, 1721.*
25. *Tara Salman And Raj Jain, "Networking Protocols And Standards For Internet Of Things", Chapter 13, John Wiley & Sons, Inc, 2017. Https://Doi.Org/10.1002/9781119173601.Ch13.*
26. *C.Bormann, S.Lemay, H.Tschofenig, K. Hartke, B.Silverajan, And B.Raymor, "Coap(Constrained Application Protocol Over TCP, TLS And Websockets. RFC8323." RFC Editor, 2018. Https://Doi.Org/10.17487/RFC8323.*
27. *Edited By Andrew Banks And Rahul Gupta, "MQTT Version 3.1.1. OASIS Standard. Oct 29, 2014.*
28. *J. L. Fernandes, I. C. Lopes, J. J. P. C. Rodrigues, And S. Ullah, ''Performance Evaluation Of Restful Web Services And AMQP Protocol,'' In Proc. 5th Int. Conf. Ubiquitous Future Netw. (ICUFN), Jul. 2013, Pp. 810–815.*
29. *R.T. Fielding, Rest: Architectural Styles And The Design Of Network-Based Software Architectures, Doctoral Dissertation, University Of California, 2000.*
30. *Robomq Over REST. Accessed: Apr. 18, 2020. [Online]. Available: Https://Robomq.Readthe Docs.Io/En/Latest/REST.*
31. *P.Saint Andre, "Extensible Messaging And Presence Protocol (XMPP): Core RFC 3920." RFC Editor, 2004.*
32. *Extensible Messaging And Presence Protocol (XMPP). Apr. 18, 2020. [Online]. Available: Https://Xmpp.Org.*
33. *Yassein, M. B., Shatnawi, M. Q., And Al-Zoubi, D. (2016). "Application Layer Protocols For The Internet Of Things: A Survey," In International Conference On Engineering & MIS (ICEMIS).*
34. *AMQP Advanced Message Queuing Protocol, Protocol Specification. Accessed: Apr. 18, 2020. [Online]. Available: Https://Www.Rabbitmq.Com/Res Ources/Specs/Amqp0-9-1.Pdf.*
35. *E. C. M. Van Der Linden, Jonas Wallgren, And Peter Jonsson. A Latency Comparison Of Iot Protocols In Mes. 2017.*
36. *Rabitmq Multi-Protocol Support. Accessed: Apr. 18, 2020. [Online]. Available: Https:// Cloudamqp.Com/Docs/Protocols.Html.*
37. *W. Shang, Y. Yu, R. E. Droms, And L. Zhang. Challenges In Iot Networking Via Tcp/Ip Architecture. Number 2, Page 7, 2016.*
38. *The Real-Time Publish-Subscribe Protocol (RTPS). Apr. 18, 2020. [Online]. Available: Https://Www.Omg.Org/Spec/DDSI-RTPS/2.3.*
39. *Chaudhari, Ketulkumar. (2018). Analysis On Transport Layer Protocols And Its Developments. International Journal Of Advanced Research In Electrical Electronics And Instrumentation Engineering. 6. 7707-7712. 10.2139/Ssrn.3729064.*

40. *S. L. Keoh, S. S. Kumar, And H. Tschofenig, ‘‘Securing The Internet Of Things: A Standardization Perspective,’’ IEEE Internet Things J., Vol. 1, No. 3, Pp. 265–275, Jun. 2014.*
41. *Z. B. Babovic, J. Protic, And V. Milutinovic. Web Performance Evaluation For Internet Of Things Applications. IEEE Access, 4:6974–6992, 2016.*
42. *I. S. Chowdhury, J. Lahiry And S. F. Hasan, "Performance Analysis Of Datagram Congestion Control Protocol (DCCP)," 2009 12th International Conference On Computers And Information Technology, Dhaka, 2009, Pp. 454-459, Doi: 10.1109/ICCIT.2009.5407282. Keywords: {Performance Analysis; Protocols;CCID2;CCID3; Congestion Control; DCCP; IETF; Streaming Multimedia},*
43. *S. Floyd And E. Kohler. Profile For DCCP Congestion Control ID 2: TCP-Like Congestion Control. Internet-Draft Draft-Ietf-Dccp-Ccid2-02, Internet Engineering Task Force, May 2003.*
44. *R. Stewart Et Al. Stream Control Transmission Protocol. RFC 2960, Internet Engineering Task Force, Oct. 2000.*
45. *E. Summary, Deploying RSVP In Multiple Security Domains Networks : Securing Application Quality Of Service, Network, 2008.*
46. *A. Mankin, Resource Reservation Protocol RSVP Version 1 Applicability Statement Some Guidelines On Deployment, Best Curr. Pract., 2008.*
47. *Https://Www.Juniper.Net/Documentation/Us/En/Software/Junos/Mpls/Topics/Topic-Map/Rsvp- Overview.Html.*
48. *Porter, Thomas, And Michael Gough. “Architectures.” How To Cheat At Voip Security, 2007, Pp. 45–110, Https://Doi.Org/10.1016/B978-159749169-3/50004-9. Accessed 25 Oct. 2021.*
49. *A. Capossele, V. Cervo, G. De Cicco And C. Petrioli, “Security As A Coap Resource: An Optimized DTLS Implementation For The Iot”, IEEE International Conference On Communications (ICC), 2015.*
50. *E. Rescorla And N. Modadugu. Datagram Transport Layer Security Version 1.2. RFC 6347 (Proposed Standard), January 2012. Updated By RFC 7507.*
51. *Kothmayr, Thomas & Schmitt, Corinna & Hu, Wen & Brünig, Michael & Carle, Georg. (2013). DTLS Based Security And Two-Way Authentication For The Internet Of Things. Ad Hoc Networks. 11. 10.1016/J.Adhoc.2013.05.003.*
52. *Vasseur J., Agarwal N., Hui J., Shelby Z., Bertrand P., And Chauvenet C., Rpl: The Ip Routing Protocol Designed For Low Power And Lossy Networks, Internet Protocol For Smart Objects (IPSO) Alliance, 2011, No. 36.*
53. *Darabkh, Khalid A., Et Al. “RPL Routing Protocol Over Iot: A Comprehensive Survey, Recent Advances, Insights, Bibliometric Analysis, Recommendations, And Future Directions.” Journal Of Network And Computer Applications, Vol. 207, Nov. 2022, P. 103476, Https://Doi.Org/10.1016/J.Jnca.2022.103476.*
54. *K. A. Darabkh And M. Al-Akhras, "RPL Over Internet Of Things: Challenges, Solutions, And Recommendations," 2021 IEEE International Conference On Mobile Networks And Wireless Communications (ICMNWC), Tumkur, Karnataka, India, 2021, Pp. 1-7, Doi: 10.1109/ICMNWC52512.2021.9688375. Keywords: {Wireless Communication; Conferences; Routing Protocols; Internet Of Things; History; Task Analysis; Iot; Routing Protocols; RPL; RPL Challenges; Solutions; And Recommendations; RPL Future Directions},*

55. *Alamery, A., Shehab, M. A., Mahmood, O. A., Hammadi, Y. I., Aziz, A., & Muthanna, M. S.(2023, December). Internet Of Things Protocols (Iotp) Literature Review. In Proceedings Of The 7th International Conference On Future Networks And Distributed Systems (Pp. 730-734).*

56. *Alahari, H. P., & Yalavarthi, S. B. (2017). A Survey On Network Routing Protocols In Internet Of Things (IOT). International Journal Of Computer Applications, 160(2), 18-22.*

57. *Aijaz, Adnan, Hongjia Su, And Abdol-Hamid Aghvami. "CORPL: A Routing Protocol For Cognitive Radio Enabled AMI Networks." IEEE Transactions On Smart Grid 6.1 (2015): 477- 485.*

58. *Darabkh, Khalid A., Et Al. "RPL Routing Protocol Over Iot: A Comprehensive Survey, Recent Advances, Insights, Bibliometric Analysis, Recommendations, And Future Directions." Journal Of Network And Computer Applications, Vol. 207, Nov. 2022, P. 103476, Https://Doi.Org/10.1016/J.Jnca.2022.103476.*

59. *Gaetano Carlucci, Luca De Cicco, and Saverio Mascolo. Http Over Udp: An Experimental Investigation Of Quic. In Proceedings Of The 30th Annual ACM Symposium On Applied Computing, SAC '15, Pages 609–614, New York, NY, USA, 2015. ACM.*

60. *Vasileios Karagiannis, Periklis Chatzimisios, Francisco Vazquez-Gallego, And Jesus Alonso- Zarate. A Survey On Application Layer Protocols For The Internet Of Things. Transaction On Iot And Cloud Computing, 3(1):11–17, 2015.*

61. *Darabkh, Khalid A., Et Al. "RPL Routing Protocol Over Iot: A Comprehensive Survey, Recent Advances, Insights, Bibliometric Analysis, Recommendations, And Future Directions." Journal Of Network And Computer Applications, Vol. 207, Nov. 2022, P. 103476, Https://Doi.Org/10.1016/J.Jnca.2022.103476.*

62. *A. Dunkels. UIP-A Free Small TCP / IP Stack. 2017, Www.Semanticscholar.Org/Paper/Uip-A- Free-Small-TCP-IP-Stack-Dunkels/34137ccad5ff56c720edd144da542c392279df96. Accessed 10 Oct. 2024.*

63. *Wei Li And Ke-Bin Jia, "Implementation Of Embedded Web Server Based On Uip TCP/IP Stack," 2007 IET Conference On Wireless, Mobile And Sensor Networks (CCWMSN07), Shanghai, 2007, Pp. 668-671, Doi: 10.1049/Cp:20070236. Keywords: {Uip;Embedded System;Web Server},*

64. *F. Cirillo, G. Solmaz, E. L. Berz, M. Bauer, B. Cheng And E. Kovacs, "A Standard-Based Open Source Iot Platform: FIWARE," In IEEE Internet Of Things Magazine, Vol. 2, No. 3, Pp. 12- 18, September 2019, Doi: 10.1109/IOTM.0001.1800022.*

65. *Keywords: {Smart Cities;Autonomous Vehicles;Semantics;Privacy;Market Research;Internet Of Things;Augmented Reality},Ccnx—Content Centric Networking. Url Https://Www.Ccnx.Org/*

66. *Mosko M, Solis I, Uzun E, Wood C (2015) Ccnx 1.0 Protocol Architecture. Technical Report, Aug 2015. Url Http://Www.Ccnx.Org/Pubs/Ccnxprotocolarchitecture.Pdf.*

67. *Miorandi D, Sicari S, De Pellegrini F, Chlamtac I (2012) Internet Of Things: Vision, Applications And Research Challenges. Ad Hoc Networks 10(7):1497–1516, Sept 2012. ISSN 1570-8705, Http://Dx.Doi.Org/10.1016/J.Adhoc.2012.02.016.*

68. *Ed Callaway Et Al., "Home Netorking With IEEE 802.15.4: A Developing Standard For Low- Rate Wireless Personal Area Networks," IEEE Communications Magazine, Vol. Aug., Pp. 70– 77, 2002.*
69. *J. Kahn, R. H. Katz, And K. S. J. Pister, "Next Century Challenges: Mobile Networking For "Smart Dust"," ACM Mobicomm, 1999.*
70. *S. C. Ergen, P. Varaiya, PEDAMACS: Power Efficient And Delay Aware Medium Access Protocol For Sensor Networks, IEEE Trans. On Mob. Comp., 5(7), July 2006, 920-930.*
71. *Pister, Kristofer & Doherty, Lance. (2008). TSMP: Time Synchronized Mesh Protocol. IASTED International Symposium On Distributed Sensor Networks, DSN 2008.*
72. *N. Burri, P. Von Rickenback, R. Wattenhofer, Dozer: Ultra-Low Power Data Gathering In Sensor Networks, Proc. IPSN 07, Cambridge, MA, 2007.*
73. *Https://Www.Indmall.In/Faq/Is-Wifi-An-Iot-Protocol/#:~:Text=Key%20Takeaway,Choice%20for%20many%20iot%20solutions.*
74. *Triantafyllou, A., Sarigiannidis, P., & Lagkas, T. D. (2018). Network Protocols, Schemes, And Mechanisms For Internet Of Things (Iot): Features, Open Challenges, And Trends. Wireless Communications And Mobile Computing, 2018(1), 5349894.*
75. *Chang K.-H., Bluetooth: A Viable Solution For Iot? [Industry Perspectives], IEEE Wireless Communications. (2014) 21, No. 6, 6–7, Https://Doi.Org/10.1109/Mwc.2014.7000963, 2-S2.0- 84920838479.*
76. *A.Kurawar, A.Koul And Prof.V.T.Patil "Survey Of Bluetooth And Its Applications", International Journal Of Advanced Research In Computer Engineering And Technology (IJARCET), Vol. 3, Issue 8, (2014).*
77. *Julio León, "A Proposal For A Bluetooth Low Energy (BLE) Autoconfigurable Mesh Network Routing Protocol Based On Proactive Source Routing", Thesis For: Ph.D., Advisor: Yuzo Iano, October 2016.. Https://Doi.Org/10.13140/RG.2.2.16660.60803.*
78. *Jiasong Mu And Liang Han, "Performance Analysis Of The Zigbee Networks In 5G Environment And The Nearest Access Routing For Improvement", Ad Hoc Networks, Vol. 56, March 2017, Pp. 1-12.*
79. *Milu Hanna Mathew And Dr. T. Parithimar Kalaignan, "A Literature Survey On Zigbee", International Journal Of Current Engineering And Scientific Research (IJCESR), Vol. 5, Issue 2, 2018, Pp. 57-60. Https://Doi.Org/10.21276/Ijcesr.*
80. *Pankaj Jadhav ,Amit Chaudhary, Swapnil Vavale " Home Automation Using Zigbee Protocol", Maturitas, International Journal Of Computer Science And Information Technologies, Vol.5(2), (2014).*
81. *Stefan Marksteiner, V. J. Exposito Jimenez, H. Vallant, And H. Zeiner, "An Overview Of Wireless Iot Protocol Security In The Smart Home Domain", Joint 13th CTTE And 10th CMI Conference On Internet Of Things Business Models, Users, And Networks, Copenhagen. 18 Jan, 2018, Pp. 1-8, IEEE. Https://Doi.Org/10.1109/CTTE.2017.8260940.*
82. *Tara Salman And Raj Jain, "Networking Protocols And Standards For Internet Of Things", Chapter 13, John Wiley & Sons, Inc, 2017. Https://Doi.Org/10.1002/9781119173601.Ch13.*

83. *Barker P. And Hammoudeh M., A Survey On Low Power Network Protocols For The Internet Of Things And Wireless Sensor Networks, Proceedings Of The International Conference On Future Networks And Distributed Systems (ICFNDS '17), July 2017, New York, NY, USA, 44:1– 44:8, Https://Doi.Org/10.1145/3102304.3102348.*

84. *A Technical Overview Of Lora And Lorawan Alliance Technical Marketing Workgroup, November 2015.*

85. *Raza U., Kulkarni P., And Sooriyabandara M., Low Power Wide Area Networks: An Overview, IEEE Communications Surveys & Tutorials. (2017) 19, No. 2, 855– 873, Https://Doi.Org/10.1109/COMST.2017.2652320, 2-S2.0-85020458974.*

86. *Ieee Standard For Local And Metropolitan Area Networks–Part 15.4: Low-Rate Wireless Personal Area Networks (Lr-Wpans) Amendment 1: Mac Sublayer, IEEE Std 802.15.4e-2012 (Amendment To IEEE Std 802.15.4-2011), Pp. 1–225, April 2012.*

87. *2011. IEEE Standard For Local And Metropolitan Area Networks–Part 15.4: Lowrate Wireless Personal Area Networks (LR-Wpans). IEEE Std 802.15.4-2011 (Revision Of IEEE Std 802.15.4-2006) (Sept 2011), 1–314. Https://Doi.Org/10.1109/ IEEESTD.2011.6012487.*

88. *Alberto Gallegos Ramonet And Taku Noguchi. 2021. LR-WPAN: Beacon Enabled Direct Transmissions On Ns-3. In Proceedings Of The 6th International Conference On Communication And Information Processing (ICCIP '20). Association For Computing Machinery, New York, NY, USA, 115–122. Https://Doi.Org/10.1145/3442555.3442574.*

89. *F. Alam, R. Mehmood, I. Katib, N. N. Albogami, And A. Albeshri, "Data Fusion And Iot For Smart Ubiquitous Environments: A Survey," IEEE Access, Vol. 5, Pp. 9533–9554, 2017.*

90. *Di Lorenzo, P., Merluzzi, M., Binucci, F., Battiloro, C., Banelli, P., Strinati, E. C., & Barbarossa, S. (2023). Goal-Oriented Communications For The Iot: System Design And Adaptive Resource Optimization. IEEE Internet Of Things Magazine, 6(4), 26-32.*

91. *Tekinerdogan, B., Köksal, Ö., & Çelik, T. (2023). System Architecture Design Of Iot-Based Smart Cities. Applied Sciences, 13(7), 4173.*

92. *Ystgaard, K. F., Atzori, L., Palma, D., Heegaard, P. E., Bertheussen, L. E., Jensen, M. R., & De Moor, K. (2023). Review Of The Theory, Principles, And Design Requirements Of Human- Centric Internet Of Things (Iot). Journal Of Ambient Intelligence And Humanized Computing, 14(3), 2827-2859.*

93. *Gardašević, G., Veletić, M., Maletić, N., Vasiljević, D., Radusinović, I., Tomović, S., & Radonjić, M. (2017). The Iot Architectural Framework, Design Issues And Application Domains. Wireless Personal Communications, 92, 127-148.*

94. *Babar, S., Stango, A., Prasad, N., Sen, J., & Prasad, R. (2011, February). Proposed Embedded Security Framework For Internet Of Things (Iot). In 2011 2nd International Conference On Wireless Communication, Vehicular Technology, Information Theory And Aerospace & Electronic Systems Technology (Wireless VITAE) (Pp. 1-5). IEEE.*

95. *Ghaffari, K., Lagzian, M., Kazemi, M., & Malekzadeh, G. (2020). A Comprehensive Framework For Internet Of Things Development: A Grounded Theory Study Of Requirements. Journal Of Enterprise Information Management, 33(1), 23-50.*
96. *Alam, T. (2023). A Reliable Communication Framework And Its Use In Internet Of Things (Iot). Authorea Preprints.*
97. *Abdulaziz, Q. A., Mad Kaidi, H., Masrom, M., Hamzah, H. S., Sarip, S., Dziyauddin, R. A., & Muhammad-Sukki, F. (2023). Developing An Iot Framework For Industry 4.0 In Malaysian Smes: An Analysis Of Current Status, Practices, And Challenges. Applied Sciences, 13(6), 3658.*
98. *Kumar, A., Akhtar, M. A. K., & Pandey, A. (2023). Design Of Internet Of Things System Based Smart City Model On Raspberry Pi. IETE Journal Of Research, 69(12), 8781-8788.*
99. *VSCP (Very Simple Control Protocol). Available Online: Http://Www.Vscp.Org/ (Accessed On 15 June 2018)*
100. *Alves, T. R., Buratto, M., De Souza, F. M., & Rodrigues, T. V. (2014, October). Openplc: An Open Source Alternative To Automation. In IEEE Global Humanitarian Technology Conference (GHTC 2014) (Pp. 585-589). IEEE.*
101. *Thingspeak. Available Online: Https://Thingspeak.Com/ (Accessed On 20 October 2024).*
102. *Parocha, R. C., & Macabebe, E. Q. B. (2019, November). Implementation Of Home Automation System Using Openhab Framework For Heterogeneous Iot Devices. In 2019 IEEE International Conference On Internet Of Things And Intelligence System (Iotais) (Pp. 67-73). IEEE.*
103. *Novita Sari, L. Openremote: Software Integration Platform For Residential And Commercial Building Automation. Openremote: Software Integration Platform For Residential And Commercial Building Automation.*
104. *Kaa Available Online: Https://Www.Kaaproject.Org/ (Accessed On 20 October. 24)*
105. *Rajalakshmi, A., & Shahnasser, H. (2017, September). Internet Of Things Using Node- Red And Alexa. In 2017 17th International Symposium On Communications And Information Technologies (ISCIT) (Pp. 1-4). IEEE.*
106. *Luis Bustamante, A., Patricio, M. A., & Molina, J. M. (2019). Thinger. Io: An Open Source Platform For Deploying Data Fusion Applications In Iot Environments. Sensors, 19(5), 1044.*
107. *Zeta Available Online: Https://Www.Zeta.Com/En/ (Accessed On 21 October. 24)*
108. *Costa, D.; Mingozzi, E.; Tanganelli, G.; Vallati, C. An Alljoyn To Coap Bridge. In Proceedings Of The IEEE 3rd World Forum On Internet Of Things, Reston, VA, USA, 12–14 December 2016; Pp. 395–400*
109. *Obaid, A. J. (2021). Assessment Of Smart Home Assistants As An Iot. International Journal Of Computations, Information And Manufacturing (IJCIM), 1(1).*
110. *Sitewhere LLC., Sitewhere System Architecture. Available Online: Http://Documentation.Sitewhere.Org/ Architecture.Html (Accessed On 30 May 2018).*

111. Lee, J.C.; Jeon, J.H.; Kim, S.H. Design And Implementation Of Healthcare Resource Model On Iotivity Platform. In Proceedings Of The International Conference On Information And Communication Technology Convergence, Jeju, Korea, 19–21 October 2016; Pp. 887–891.

112. Kim, M., Lee, J., & Jeong, J. (2019). Open Source Based Industrial Iot Platforms For Smart Factory: Concept, Comparison And Challenges. In Computational Science And Its Applications–ICCSA 2019: 19th International Conference, Saint Petersburg, Russia, July 1–4, 2019, Proceedings, Part VI 19 (Pp. 105-120). Springer International Publishing.

113. Platformio. Available Online: Https://Platformio.Org/ (Accessed On 21 October 2024)

114. WSO2 Iot. Available Online: Https://Wso2.Com/Iot/ (Accessed On 21 October 2024)

115. Eclipse Kura. Available Online: Https://Projects.Eclipse.Org/Projects/Iot.Kura (Accessed On 21 October 2024)

116. Villali, V., Bijivemula, S., Narayanan, S. L., Prathusha, T. M. V., Sri, M. S. K., & Khan, A. (2021, October). Open-Source Solutions For Edge Computing. In 2021 2nd International Conference On Smart Electronics And Communication (ICOSEC) (Pp. 1185- 1193). IEEE.

117. Crichton, R. (2015). The Open Health Information Mediator: An Architecture For Enabling Interoperability In Low To Middle Income Countries (Doctoral Dissertation).

118. Thingsboard. Available Online: Https://Thingsboard.Io/ (Accessed On 22 October 2024).

119. Mainflux. Available Online: Https://Www.Mainflux.Com/ (Accessed On 17 October 2024)

120. Zigbee2MQTT. Available Online: Https://Www.Zigbee2mqtt.Io/ (Accessed On 17 October 2024)

121. Vishnu, S. S. H., & Nagappan, G. (2024, March). The Homie–An Application For Tenants And Owners. In 2024 IEEE International Conference On Interdisciplinary Approaches In Technology And Management For Social Innovation (IATMSI) (Vol. 2, Pp. 1-5). IEEE.

122. Zetta Available Online: Https://Www.Opensourceforu.Com/2017/07/Zetta-Api-First-Iot- Platform/ (Accessed On 22 October 2024)

123. Macchina.Io. Available Online: Https://Www.Macchina.Io/ (Accessed On 19 October 2024).

124. Distributed Services Architecture For Iot (DSA). Available Online: Http://Iot-Dsa.Org/ (Accessed On 16 June 2018).

125. Kim, J.; Lee, J.W. An Open Service Framework For The Internet Of Things. In Proceedings Of The IEEE World Forum On Internet Of Things (WF-Iot), Seoul, Korea, 6–8 March 2014; Pp. 89–93.

126. Alves, T. R., Buratto, M., De Souza, F. M., & Rodrigues, T. V. (2014, October). Openplc: An Open Source Alternative To Automation. In IEEE Global Humanitarian Technology Conference (GHTC 2014) (Pp. 585-589). IEEE.