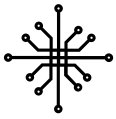

**Aregtech**

**bring your product to life service enabled**

# IoT Edge-Centric Application Development

## Edge development frameworks, platforms and architecture

Artak Avetyan
Aregtech founder,
AREG IoT SDK author
Aregtech UG
Munich, Germany
artak[at]aregtech.com

***Abstract*—With the rapid growth of the Internet of Things (IoT) technology market, many corporates are now faced with pulling together several technological solutions. However, potentially solutions that are not well chosen or thought through can create additional engineering, application development, and quality assurance difficulties in order to achieve their IoT goals. Technology change during any development phase is proving to be more difficult, more expensive, and sometimes impossible. Thus, there is a critical need to choosing the right technology, architecture, and application frameworks in line with business needs. This paper addresses software technology decision makers and encourages the introduction of edge-centric and cloud complementary IoT development solutions at the edge of the network in order to ease facilitate application development and reduce development costs. It also aims to facilitate the decision-making process around what technology to implement. The paper also addresses the issues that business leaders and decision-makers need to address when confronted with different IoT computing paradigms, networking infrastructures, communication and computing technologies and the relationships between them, as well as any technology disadvantages. In addition, we provide an introduction to connectivity frameworks that help to distribute computing tasks at the edge of the IoT network and use computing power of connected things more efficiently. In order to optimize communication, the introduction considers techniques to provide / enable services in embedded applications of IoT devices. This makes them more intelligent, so that instead of Big Data, by request, to provide meaningful and qualified information that can create numerous novel opportunities in IoT.**

***Keywords—Internet of Things (IoT); continuous nebula; distributed computing; cloud computing; fog computing; mist computing; edge-centric computing, embedded computing; service enabled applications; machine-to-machine communication.***

## I. INTRODUCTION

There is no need to point out how technology has changed in the last decades. Both the internet and connectivity have been growing at an unprecedented rate, affecting all areas of the economy and society and attracting huge public attention.

Billions of commercial smart devices with embedded operating systems are connected in both small and large networks. According to IoT Analytics estimates, at the end of 2019, there were roughly 9.5 billion connected IoT devices. That number is significantly larger than the forecast of 8.3 billion devices [40]. Cisco forecasts [1] that by the year 2022 the number of internet connected devices will reach 3.6 times the global population (28.8 billion devices), and the share of machine-to-machine (M2M) connections will be more than half, reaching 14.6 billion. The annual global IP traffic will reach 4.8 zettabytes by the year 2022, growing at a Compound Annual Growth Rate (CAGR) of 26% over the next five years. All this will require robust solutions in the areas of network communications, device management, interoperability, data streaming, and communication protocols. In traditional cloud-based application development, data is required to be sent and stored in a centralized cloud data center where all data is analyzed and decisions are made. The current mobile network will face significant obstacles handling such a huge amount of data. Due to bandwidth constraints, moving big data from IoT devices to the cloud might be ineffective in some cases. Moreover, sending data to the cloud is not a feasible solution in some applications due to privacy concerns. In addition, the cloud is not able to provide ultra-low latency or location-aware services to develop time-sensitive and location-aware

 

applications such as autonomous vehicles, real-time manufacturing, flocks of drones, patient monitoring, smart homes and office environments. In summary, the cloud is simply not enough [2] to provide self-sufficient solutions.

To solve the issues of network communication latency and high bandwidth, data privacy and dispersion, there is a need to move computing power closer to the edge of network, where data originate. The industry proposes edge-centric computing paradigms to address these issues, and to bridge the gap between cloud and connected devices by enabling and distributing computing, storage, device and data management applications on the network nodes closer to IoT devices. Where previously devices mainly collected and sent data for further computation, today smart devices are able to perform complex computations on-side. This extends cloud computing power by bringing and distributing services closer to the edge of the network.

Fog, edge and mist computing are edge-centric paradigms to address mentioned issues. Despite many similarities between edge and fog computing or edge and mist computing, both academia and industry differentiate edge computing from fog and mist. In this whitepaper, all of these terms are defined as "edge-centric". You can find descriptions and definitions of edge, fog and mist computing in separate chapters, where the similarities and differences are pointed out, and the way in which computing at the edge of network extends the cloud is outlined. For a deeper understanding, I've provided descriptions and tutorials of different network modeling, connectivity solutions, frameworks, and use case examples.

## II. IoT SOFTWARE TODAY

### A. History of IoT

The term "Internet of Things" was coined by Peter T. Lewis in a 1985 speech given at a U.S. Federal Communications Commission (FCC) during session at the Congressional Black Caucus 15th Legislative Weekend Conference. In particular, in his speech he stated *"By connecting devices such as traffic signal control boxes, underground gas station tanks and home refrigerators to supervisory control systems, modems, auto-dialers, and cellular phones, we can transmit status of these devices to cell sites, then pipe that data through the Internet and address it to people near and far that need that information. I predict that not only humans, but machines and other things will interactively communicate via the Internet. The Internet of Things, or IoT, is the integration of people, processes and technology with connectable devices and sensors to enable remote monitoring, status, manipulation and evaluation of trends of such devices. When all these technologies and voluminous amounts of Things are interfaced together – namely, devices/machines, supervisory controllers, cellular and the Internet, there is nothing we cannot connect to and communicate with. What I am calling the Internet of Things will be far reaching."* [3].

In 1985 the term "*Internet*" was in its infancy and "*Things*" generally meant any object that could not easily given a name.

### B. What Are Connected Things?

In his speech, Peter T. Lewis distinguished people, processes, technologies, devices/machines, and sensors as the objects to integrate in the digital world. There have been many attempts to define the "*Internet of Things*", but it has resulted in a dilution of understanding what it meant back in 1985 and what it should actually mean today. "*Things*" no longer necessarily mean devices. But on the other hand, devices are things. When we turn on the light of a smart lamp, the following connected *things* are involved:

- Actor - user or data as a trigger;
- Software - mobile and controlling applications;
- Devices - mobile, lamp / LED, switch, sensor, actuator (to control the power), network device.

To produce smart products, developers need to take an abstract view of things and devices and understand the nature of things as a data model. To achieve this, developers define data structure, separate telemetry and controlling data, define Application Programming Interface (API), choose and use communication protocol, and finally implement applications. If the same data model is valid for other devices, in principle the same application can be used to control other devices. For instance, if lighting and heating systems could have the same data structure, business logic, and communication protocol, the same application could be used to control both. In this case, the IoT application could be used to control at least two completely different smart products - smart lamp and smart heating. And if these devices can be controlled by one mobile application, they would probably use the same mobile application, have different instances of data-tables and applications in cloud, share sensors (motion sensors) and connectivity devices (WiFi / Bluetooth), share data structure and API, and would exchange the data and states. In this case, beside smart physical devices, there is an interaction of virtual objects such as object properties, interaction information, object, and agent behaviors.

The Internet of Things or Connected Things is a comprehensive environment to interconnect a large number of physical and virtual objects that contain technology and mechanisms for transmitting information (communication), and interact with their internal states and external environment to enhance the efficiency of the applications.

### C. IoT Software Technologies

Technologies are obsessed with software and hardware, and while these all play an important role, the market is interested only in the benefit that IoT can offer. The value lies in things and in data, not in the complexity of electronics and technology such as communication protocols or development platforms. IoT technology developers should focus on flexibility and simplicity of solutions to interact with, extract and process data. Customers do not buy or use over-complicated products and developers use those solutions to build software for those products that are simple, fun to work with and are commercially attractive. In the rapidly-growing technology markets, it is not easy to find robust solutions, and corporations often implement several technologies and solutions to develop a single product, which may then create additional problems and challenges

during the product development as well as increase the risk of delays in product release dates.

IoT projects have three characteristic cross-dependent technology elements: software, hardware, and connectivity. According to MachNation CTO [4], roughly 90% of edge complexity is related to software, while many vendors already provide hardware that adequately supports many IoT solutions.

## III. Data is the Valuable Resource

Sensory technology is deeply integrated into many devices. Modern mobile phones contain several sensors, modern cars contain more than a hundred sensors, and airplanes can contain thousands of sensors. Data from these sensors describe device and environmental states that are used to control and visualize devices, as well as being a source for predictive analytics in Machine Learning (ML) technology. In almost every industry, data have the ability to create new business cases in our daily lives. Today, data handling has become a new science and one of the most valuable resources. Nowadays, Artificial Intelligence (AI) is widely used in customer support, chatbots, fraud detection, and image recognition. The industry already uses many AI powered applications. Business derived from AI is predicted to reach almost $4 trillion by the year 2022 [5].

In Artificial Intelligence and Machine Learning, data are the essential source for training and increasing the quality of the neural network. One of the top reasons why the AI projects fail is the lack of data volume or data quality, because if there are not enough data or the data are incomplete or wrong, the neural network algorithms cannot be correctly tested. We might see the emergence of a new business around qualified data providers, where, similar to search engines, data collector engines would collect and distribute qualified data from every available public place. Today, it makes sense not to talk about big data, but about information, because this term emphasizes quality, instead of quantity .

So, IoT is the technology that connects Things and moves information from Things to data centers for further use and processing.

## IV. Computing Paradigms

As data move from connected devices to the cloud, it passes several nodes and computing processes. The standard IoT system involves three major technologies: constraint or edge systems, middleware and cloud services. Edge systems provide intelligence to the constraint devices, middleware provides interconnection of heterogeneous edge systems to the cloud, and the cloud provides comprehensive storage, processing and management mechanisms. The market offers following major IoT computing paradigms:

- Cloud computing (CC).
- Fog computing (FC).
- Edge computing (EC).
- Mist Computing (mist).

While the border of cloud computing is clear, fog, edge, and partially mist computing have many similarities and the borders are blurred. In the beginning, the term mist computing was an alternative for fog computing. Many companies offer edge computing technology, although the solutions are more related to fog computing. The reason for such an unclear delineation is that fog computing needs the support of all related edge computing technologies. In other words, the fog cannot be managed without the integration of edge computing technologies. That's why many have difficulties identifying whether they provide / need fog or edge computing technology.

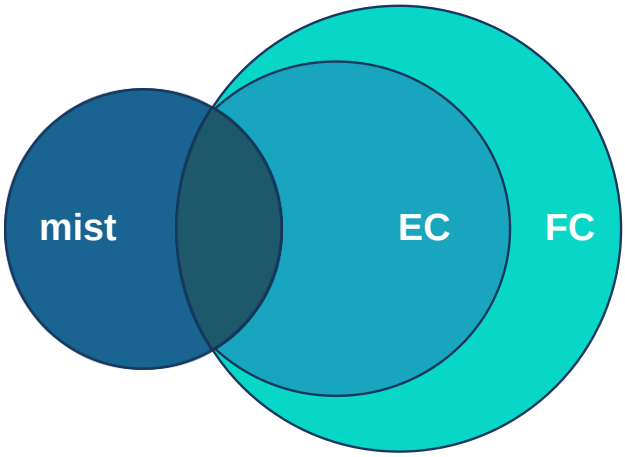


Fig. 1. Classification of scope of Mist Computing (mist), Edge Computing (EC) and Fog Computing (FC) paradigms and the intersection.

*Fig. 1* shows the intersection of mist, edge, and fog computing that includes real-time device control, data ownership and protection, data service, and communication.

Fog, edge and mist computing, all possess the common term "edge" and are combined in edge-centric computing paradigms. In the telecommunication industry the edge network usually refers to a 4G or 5G base station. In other networks, edge is the immediate first connection and network access point from IoT devices such as IoT gateways, and refers to local network where IoT sensors, actuators, and devices are located. If the computation is carried out directly on IoT devices, this computing paradigm refers to mist computing. For a better understanding of the benefits of these computing paradigms, they are examined in more detail in the following chapters.

## V. Cloud Computing

The National Institute of Standards and Technology (NIST) defines cloud computing as a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [7]. NIST characterizes the cloud computing model as a composition of three service models and four deployment models which can be categorized into cloud services and cloud types.

### A. Cloud Services

The cloud offers three types of services: infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS) [7]. These services differ by enabled characteristics, physical, and abstraction layers. Application

 

developers use these services depending on the needs of the application they develop (*Fig. 2*).

*Infrastructure as a Service (IaaS)*. A cloud infrastructure is a service model for accessing, monitoring and managing remote datacenter. The consumer rents or leases servers (as well virtual server) to compute or store data. The consumer does not manage or control underlaying cloud infrastructure, but can deploy and control any software component, including operating system and applications, control storage, and possibly have limited control over network components such as firewall.

*Platform as a Service (PaaS)*. A cloud platform is a collection of software libraries and services called middleware, and tools supported by provider to allow developers to create applications, support application lifecycle, and deploy onto the cloud created or acquired applications. The consumer does not manage or control the cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-hosting environment, including database management and scaling applications.

*Software as a Service (SaaS)*. A cloud software is a collection of applications running on a cloud infrastructure. Applications are accessed through a defined interface called Application Programming Interface (API) from various IoT devices and other applications, including a web browser, mobile application, or another web-service or cloud application. The consumer does not manage or control the underlying cloud infrastructure which includes network, servers, operating systems, storage, or even individual application capabilities, but has possible access to limited consumer-specific application configuration settings.

### B. *Cloud Types*

There are four types of cloud deployments: private cloud, community cloud, public cloud, and hybrid cloud [7].

*Private cloud*. The cloud infrastructure is designed for exclusive use by a single organization to ensure a high level of privacy and configurability, and is normally used to deploy proprietary applications. It may be owned, managed, and operated internally by the organization, third party, or combined. It may be hosted on- or off- premise.

*Public cloud*. The cloud infrastructure is designed as a typical model of cloud computing, where cloud services are offered by the cloud service provider. It may be free and provisioned for public use. It may be owned, operated and managed by business, academic, government organizations, or combined. It is hosted on the premises of the cloud provider.

*Community cloud*. The cloud infrastructure's purpose is to share resources between several organizations in a community that results from or in ownership decentralization. It may be owned, managed, and operated by one or more of the organizations in the community, third party, or combined. It may be hosted on- or off- premises.

*Hybrid cloud*. The cloud infrastructure is designed to be combined with more than one cloud type in order to maintain control over virtualized infrastructure that remain distinct entities but are bound together by virtualized interface models.

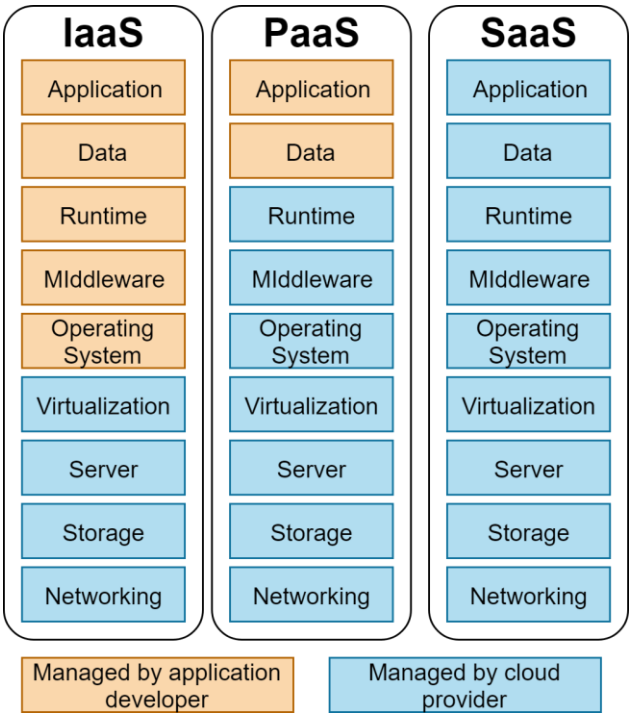


Fig. 2. Cloud services and the classification of application stack managed by developers and the service provider [20].

## VI. FOG COMPUTING

Traditional cloud-based IoT systems are challenged by the large scale, heterogeneity, and high latency observed in some cloud ecosystems. One solution is to decentralize applications, management, and data analytics in the network itself by enabling computing and data management closer to IoT edge devices using a distributed and federated compute model. This approach is known as fog computing. Fog computing (or fog networking) is an architecture that uses edge devices (IoT nodes) to carry out a substantial amount of computation, storage, networking, decision-making, and data management along the "IoT-to-Cloud" path as data moves from IoT edge to cloud.

The National Institute of Standards and Technology (NIST) released a definition of fog computing as a layered model for enabling ubiquitous access to a shared continuum of scalable computing resources. The model facilitates the deployment of distributed, latency-aware applications and services and consists of fog nodes (physical or virtual) that reside between smart end-devices and centralized (cloud) services. Fog computing minimizes the request-response time from/to supported applications and provides for end-devices, local computing resources and, when needed, network connectivity to centralized services [8].

The fog nodes are components of fog computing architecture. They are either physical nodes such as gateways, servers, switches, routers, etc., or virtual nodes such as virtual machines, virtual switches, etc. Due to the location of fog nodes closer to IoT devices, analysis of and response to data generated

by connected devices are much quicker than from a centralized cloud service or data center.

Fog and cloud complement each other and one cannot replace the needs of the other. Fog extends cloud to the edge of the network. Federation between fog and cloud allows enhanced capabilities for data aggregation, processing, and storage. For instance, the fog can filter, pre-process, and aggregate big data from source devices, while heavy analytical processing and archiving can be carried out in the cloud [20].

The main characteristics of fog computing are as follows [21]:

- It extends cloud into the fog and edge domain and performs cloud functions in a single continuum.
- It applies cloud principles horizontally across different types of domains and interconnects IoT verticals to share technology, resources and data across domains in order to improve efficiency and productivity.
- It enables control and management of multiple fog domains called Fog Federation, comprising of edge devices, computers, networking, storage, and services in a distributed and consistent manner.
- It enables end-to-end security from the cloud to edge devices across IT/OT (integration of information technology (IT) systems used for computing with operational technology (OT) systems used to monitor events, processes and devices).
- It brings data processing and analysis closer to sources to enable data analytics of multiple interoperating edge devices for anomaly detection, failure prediction, and optimization of the ecosystem.

For some applications, fog computing is becoming an alternative. In addition, several studies conclude that centralized data centers consume a significant amount of energy compared to small distributed servers, or nano data centers [14].

## VII. EDGE COMPUTING

The increase of IoT devices at the edge of the network is producing a massive amount of data to store and compute in data centers. Edge computing is a distributed computing paradigm, which brings computation and data storage closer to the data origin in order to improve response times and save bandwidth [9]. It enhances processing, management, and storage of data generated by IoT devices. The aim of edge computing is to move computation from data centers toward the edge of the network in order to carry out tasks and provide services on behalf of the cloud. A fundamental part of edge computing is the strong and seamless integration between IoT devices and the cloud. It integrates IoT devices with the cloud by filtering, preprocessing, and aggregating IoT data via cloud services deployed close to IoT devices. It uses the power and flexibility of cloud services to run complex analytics, support decisions, and take actions [10].

Due to its proximity to IoT edge devices and end users, network latency in edge computing is sufficiently lower than in cloud computing. However, computation and data processing latency depend on the provision of computing power of edge units. Service availability in edge computing is also higher, since connected IoT devices and applications do not need to wait for centralized cloud platform for the provision of service.

In edge computing, the edge refers to computing infrastructure close to the source of data. Since it is the first connection and network access point of IoT devices, the computation can be carried out at device level. If fog computing focuses on the interaction between connected devices, in edge computing the focus is on the technology attached to connected things, such as industrial machines, WiFi access points, etc. [11].

In edge computing, services can operate with little or no internet connectivity, and devices can establish any form of connectivity such as LAN, WiFi, Bluetooth, ZigBee, etc. Edge computing works well with small data centers and applications can benefit from real-time communication.

## VIII. MIST COMPUTING

In the early days, mist computing was an alternative term for fog computing [12]. The mist computing paradigm is the distributing computing mechanism to the extreme edge where IoT devices are located (i.e. IoT devices themselves) in order to minimize the communication latency between IoT devices in milliseconds. It is the first computing location in the IoT-fog-cloud continuum and can be considered as “IoT computing” or “things computing”. It is proposed to grant the IoT devices the capability of self-awareness in terms of self-organizing, self-managing and several self-* mechanisms so that IoT devices become fully autonomous and be able to operate continuously, even in the event of no available internet or an unstable connection.

NIST defines mist computing as a lightweight and rudimentary form of fog computing that resides directly within and at the edge of the network fabric, thus bringing the fog computing layer closer to smart end-devices. Mist computing uses microcomputers and microcontrollers to feed into fog computing nodes and potentially onward toward centralized (cloud) computing services [8].

In mist computing, IoT devices may act not only as “thin clients”, but also as “thin servers” by enabling services accessed within network where the data can be processed directly on the device itself. Even if both battery and computing power is today more than it was 10-15 years ago, devices use up to five times more energy for communicating with wireless devices rather than for computers [19]. By processing data directly on an IoT device, mist computing helps to reduce power consumption.

The guiding principles of mist computing are [13]:

- The network of connected devices provides information, instead of simply data.
- The network delivers only information that has been requested, when it has been requested.
- Devices create a dynamic system that works together using a subscriber-provider model.

- Device applications are service providers that are accessible in the network to execute device specific or a narrow range of tasks.
- Devices adapt to network configuration and information and dynamically discover data providers and available services before executing an application.

This concept of system architecture sufficiently reduces the amount of streaming data in the network. *Fig. 3* demonstrates the relationship of mist-edge-fog-cloud computing [8]. In mist computing, connected things can directly communicate with each other, bypassing any other network node, remaining autonomous and becoming an extension of the fog network. The latency in mist computing is ultra-low.

Two examples of dynamic information and the provision of services in mist computing are movement detection and light sensors. In combination, they can provide light switch and/or alarm system services to be used by other devices and applications in the network, such as connected elevators, door lockers, lightning and alarm services in office environments to trigger actions in the event of an alarm. In the case of smoke detection, the alarm service distributes information that stops the movement of elevators, locks can become unlocked, evacuation doors can open, and exit paths can be lit in line with specific evacuation plans. Device functionality and any controlling applications can change if an alarm triggers a "security break", so that elevators can continue working or locking mechanisms might need to close some doors, instead of opening them. Depending on whether alarm services triggers "safety" or "security" information, the behavior of these same devices may change.

If fog computing is the extension of cloud to the edge, mist computing is the extension of fog to the extreme edge. Materially, mist computing perfectly integrates IoT devices into the fog infrastructure, where devices connect to fog not only as traditional clients, but as ultra-small servers that provide services to execute either device specific or a narrow range of tasks. In fact, IoT service-enabled embedded applications and a well-organized fog infrastructure can make IoT devices an integral part of fog, so that, in combination with cloud, form a *continuous nebula*. The key importance of such a model are capabilities of IoT applications to provide and discover services that can be used by other applications in the network. The

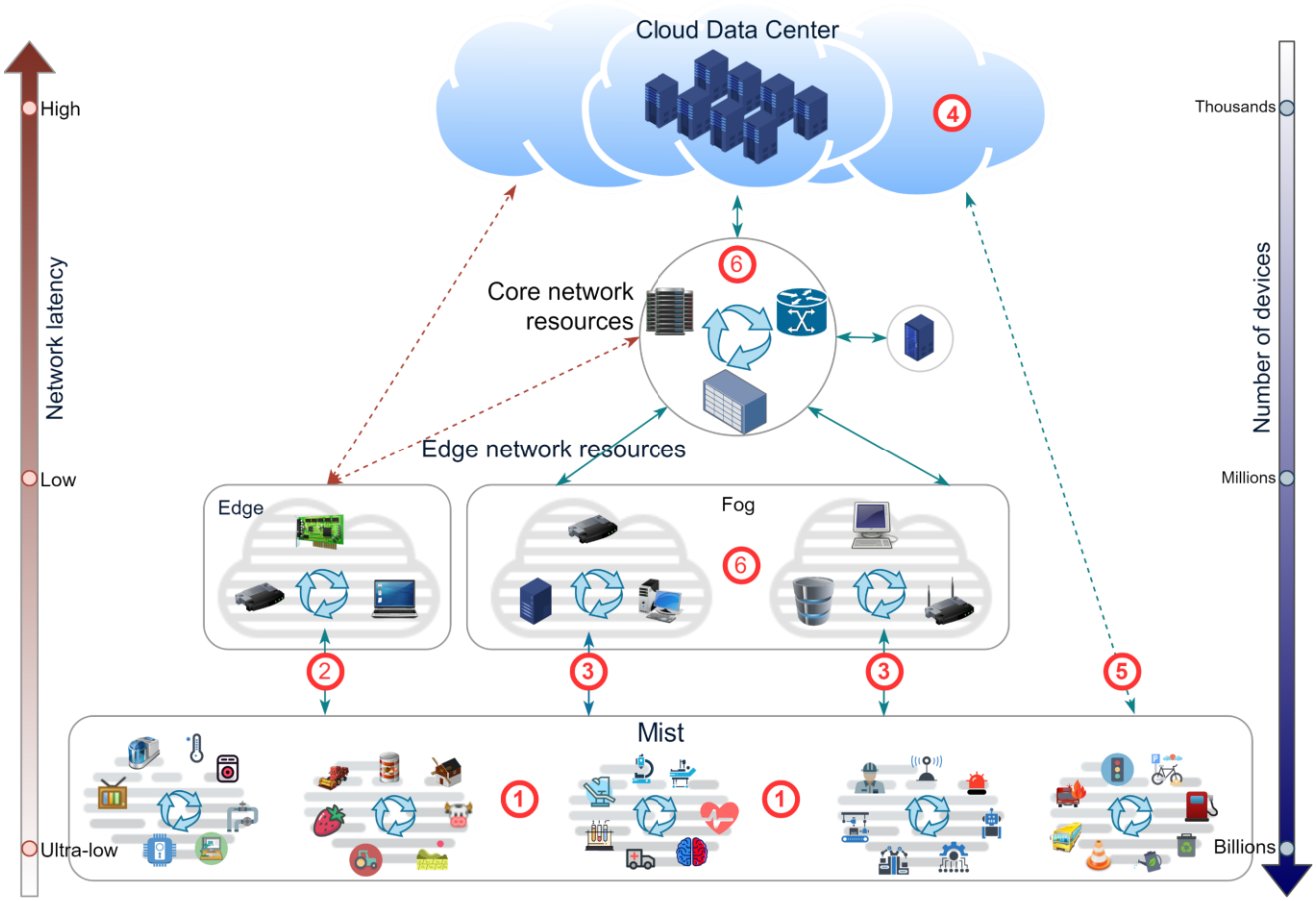


Fig. 3. Mist-to-cloud ecosystem. Interaction of mist, edge, fog, and cloud computing.

① - Mist computing and infrastructure.
② - Edge computing and network resource.
③ - Fog computing and network resource.
④ - Cloud computing and data center.
⑤ - Interaction of IoT devices with cloud.
⑥ - The next level resources in the network.
— - The next level of computing extension.
- - - Traditional IoT to cloud connection.
- - - Optional connection to network nodes.

connectivity frameworks that can help to implement service-enabled applications are introduced further in this paperwork.

## IX. Cloud vs Fog vs Edge vs Mist

*Fig. 3* visualizes mist computing (1), edge computing (2), fog computing (3), cloud computing (4), interaction of IoT devices and cloud (5), and capabilities further enhanced by next level resources in the network (6). In this chapter we compare these paradigms.

### A. Cloud vs. Fog Computing

Compared to traditional cloud computing, with a centralized data center, the nodes in fog computing are generally more decentralized and located closer to the network edge. This sufficiently reduces latency. The decentralized nature of fog computing allows devices to either serve as fog computing nodes themselves (e.g. each IoT device provides a service and acts as a fog node) or use fog resources as traditional clients of the server. Fog nodes are widespread and geographically available in large numbers. Security is provided at the edge or at dedicated locations in the fog network [20].

The major difference in cloud computing is the availability of high computing and storage resources which requires a relatively high level of power consumption, whereas fog computing requires relatively lower power consumption [14].

Since the hardware for fog computing is located closer to the user and is more limited in resources, fog computing utilizes small servers, gateways, routers, switches, and data centers, whereas cloud computing traditionally utilizes large data centers.

In fog computing, fog-based services do not require a continuous internet connection, since services are accessed through connected devices in the network and services work independent of an internet connection. In cloud computing, services are available only when an internet connection is established.

### B. Mist vs. Fog and Edge Computing

Regardless whether fog and edge computing both move the computation power closer to the network edge where the data is generated, these paradigms are not identical. According to the OpenFog Consortium, fog computing is often erroneously called edge computing, but there are key differences. Fog works with the cloud, whereas edge is defined by the exclusion of the cloud. Fog is hierarchical, whereas edge tends to be limited to a small number of layers. In addition to computation, fog addresses networking, storage, control, and acceleration, while edge computing is limited to computing at the edge [15].

*Fog computing* is a network architecture that consists of various components such as gateways, routers, and services. Fog computing is the extension of the cloud to the edge of the network. It allows decentralized computing through the distribution and processing of data at the fog nodes, which can be any device capable of storage, computing, and network connectivity. Therefore, fog computing runs applications in a multi-layered architecture that decouples and meshes the hardware and software functions. Fog computing can be useful for smart cities and smart manufactures, where various devices use real-time data to perform tasks. It can be used in autonomous vehicles and in the entertainment industry to process data and perform in real-time. Multiple business applications that require instant data processing can use fog computing.

Fog computing enables real-time data analysis, which in turn can make IoT applications work quicker. By processing data at fog nodes, businesses can reduce both storage and computing costs. Confidential data can be secured as it can be stored at the fog node. It can also be used to develop low latency networks between analytics endpoints and devices. Using such networks can lead to reduced bandwidth requirement when compared to cloud computing. In contract to edge computing, fog computing can process large volumes of data as it is capable of processing real-time requests. The disadvantage of fog computing is that it is dependent on multiple links for transferring data from the physical asset chain to the digital layer, which can then result in potential points of network failure [16].

*Edge computing* is used for processing data directly on devices or gateways that have attached sensors or are close to the sensors. It enables devices to process data in real-time or in near real-time mode without relying on the cloud or fog, and it reduces overheads at the centralized cloud. Edge computing runs specific applications in a fixed location and provides service. Edge computing can be used in smart homes to perform tasks such as the control of domestic heating and lights in real-time. In smart manufacturing, it can simplify predictive maintenance by sending instant alerts about possible defects in equipment.

Edge computing simplifies internal communication by wiring physical assets to IoT devices for the collection and processing of crucial data. After processing the necessary data, IoT devices can determine data to be stored locally and to be sent to the cloud for analysis. In this manner, sensitive data can be stored discreetly at its source. The disadvantage of edge computing is that it is less scalable when compared to fog, and supports little interoperability, which might render IoT devices incompatible with certain cloud services and operating systems. It also does not support resource pooling [16].

*Mist computing* is utilized at the extreme edge of a network which consists of embedded devices such as microcomputers, microcontrollers, and sensors. It harvests resources with the help of computation and communication capabilities available on the sensor and uses the communication infrastructure of microcontrollers and microcomputers to transfer data to fog computing nodes as well as to the cloud. Mist computing uses network infrastructure to manage arbitrary computations on the sensors, microcontrollers, and microcomputers, making devices autonomous. Mist computing can be useful in public transportation [17] and manufacturing as devices may not be stationary and may serve a singular purpose.

Mist computing enables local decision-making with the help of microcomputers, microcontrollers, and sensors. It helps in conserving bandwidth and battery power as only essential data is transferred to the gateway, server, or router. Additionally, mist computing enables utilization of data access control mechanisms that can ensure data privacy at a local level. The

disadvantage of mist computing is that its infrastructure can only be used for lightweight data processing and a narrow range of tasks [16].

## X. Complementarity of Cloud, Fog and Mist

Cloud, fog, and mist computing are complementary to each other and are meant to work together [18]. Each paradigm has advantages and disadvantages that, through correct use, we ensure any weaknesses are minimized and systems and applications efficiency when scaling a network of connected devices are maximized (*Fig. 3*). The use of these paradigms at the same time depends on each business case. For instance, for automated agricultural irrigation or the monitoring of applications that do not require internet, it can be sufficient for data to be stored at the premises and therefore no cloud is required. However, there are many other cases where these paradigms solve various problems creating effective, secured and stable systems. For a better insight into fog, mist, and cloud computing complementarity. Let us review an example of autonomous vehicles.

As stated, fog is a computing paradigm that decentralizes computing by distributing and processing data at the fog nodes, which can be any device between edge of network and cloud, and capable of storage, computing, and network connectivity. Mist computing puts computing power at the extreme edge of the network onto actual IoT devices and smart sensors. For this reason, processing capability is more limited. With all the benefits of fog and mist computing, cloud computing offers the largest amount of computing capability.

An autonomous vehicle is an example of a complementary computing of mist, fog, and cloud. The job of sensors is to record data from the environment and transfer the information to data storage. An autonomous car relies on data received from sensors. Having all data processed at the cloud is a no-go, since the car especially needs safety, rescue, and security-relevant results as quickly as possible, and in almost real time.

Data transfer uses much more battery power than an equivalent computing process. By having computing power on the smart sensor or smart IoT device, data can be processed, preconditioned, and optimized first before being stored. The resulting data is smaller and consumes less battery power in the transfer. Several studies have shown that moving processing power closer to the data origin and balancing such ratios as active and idle time, as well as the number of downloads and uploads, could yield energy efficiency improvements [14]. Some authors conclude that communication consumes up to five times more power in embedded microcontrollers than computation [19]. By aggregating, preprocessing, fusing, and preconditioning data at the vehicle, less data are sent to the gateway, server, and cloud. This conserves power and bandwidth more efficiently, which are crucial characteristics for moving an autonomous vehicle.

Having a gateway device that can process the data right away is crucial for the car to function properly. In addition, the gateway device must be able to filter and find the relevant data to distribute in the network of nearby connected vehicles or send to the cloud for further analysis.

However, to improve the performance of the vehicle, the collected data needs to be analyzed. Since analyzing the data requires a huge amount of computational power and it is not a time-sensitive operation, cloud computing is the best way, since it is capable of storing, sharing and processing huge amounts of data.

One further characteristic is enhanced data security on the system. In reviewed computing architectures, data is processed locally before being sent to the remote server. By processing data at mist or fog, sensitive data can be removed or encrypted, reducing the amount of security threats the system has to deal with.

## XI. Related Edge Computing Paradigms

This paper focuses on Fog Computing (FC), Edge Computing (EC), Mist Computing (mist), and touches on Cloud Computing (CC). However, industry and academia offer Mobile Computing (MC), Mobile Cloud Computing (MCC), Multi-access Edge Computing (MEC), Mobile ad hoc Cloud

TABLE I. FEATURES OF IOT COMPUTING PARADIGMS.

| **Table Head** | **Table Column Head** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | ***CC*** | ***FC*** | ***EC*** | ***mist*** | ***MC*** | ***MCC*** | ***MACC*** | ***MEC*** | ***CoT*** |
| Heterogenity support | ✓ | ✓ | ✓ | ✓ | × | ✓ | × | × | × |
| Infrastructure need | ✓ | ✓ | ✓ | ✓ | × | ✓ | × | ✓ | ✓ |
| Heterogenity support | × | ✓ | ✓ | ✓ | × | × | × | ✓ | ✓ |
| Geographically distributed | × | ✓ | ✓ | ✓ | × | × | × | ✓ | ✓ |
| Location awareness | × | ✓ | ✓ | ✓ | ✓ | × | ✓ | ✓ | ✓ |
| Ultra-low latency | × | ✓ | ✓ | ✓ | × | × | × | ✓ | ✓ |
| Mobility support | × | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Real-time application support | × | ✓ | ✓ | ✓ | × | × | × | ✓ | ✓ |
| Large-scale application support | ✓ | ✓ | ✓ | ✓ | × | × | × | ✓ | × |
| Standardized | ✓ | ✓ | ✓ | × | ✓ | × | × | ✓ | × |
| Multiple IoT applications | ✓ | ✓ | × | ✓ | × | × | × | × | ✓ |
| Virtualization support | ✓ | ✓ | × | × | × | × | × | ✓ | ✓ |

Computing (MACC), and Cloud of Things (CoT) paradigms [20]. For a brief understanding of the features of various IoT computing paradigms, please refer to *TABLE I* of this paper. The table also demonstrates commonalities and differences between fog, edge, and mist computing.

For a brief overview, here is short description of the general use cases of IoT computing paradigms [20]:

- *Cloud computing:* Scalable data storage, virtualized apps, distributed computing for large data sets (Google MapReduce).
- *Mobile computing:* Mobile sales transactions, location-dependent queries (travel recommendations), multimedia applications on mobile devices.
- *Fog computing:* IoT, connected vehicles, smart factory, smart grid/smart city, health care, smart delivery (high-scale package drone delivery), real-time subsurface imaging, video surveillance.
- *Edge computing:* Local video surveillance, video caching, traffic control.
- *Mobile cloud computing:* Social networking, sensor data processing, health care (tele-monitoring and tele-surgery).
- *Mobile ad hoc Cloud Computing:* Networking and computing for disaster relief, group live video streaming, unmanned vehicular system.
- *Multi-access edge computing:* Content Delivery, Video analytics, connected vehicles, health monitoring, augmented reality.
- *Cloud of things:* Optical character recognition (OCR), wearable cognitive assistance (Google Glass), environment continuous monitoring.
- *Mist computing:* Parallel computation on IoT devices, autonomous vehicles, smart factory, smart office, health care, privacy-preserving local processing.

It is important to understand the characteristics of IoT computing paradigms while some may suit particular use cases better than others. However, because of its comprehensive definition scope, generality, and extensive presence, fog computing in combination with mist computing are considered as more general forms of computing when compared to other solutions.

## XII. ARCHITECTURE

The cloud and fog possess an awareness of the global environment, whereas the mist is more attuned to the local environment. Together they are responsible for executing IoT applications. Data from billions of connected devices and a high number of requests to process data cause huge network traffic, high service latency, and result in the consumption of enormous amounts of energy. Today, data centers alone consume 2% of the world's produced energy [39]. To realize the full benefit of IoT, it is necessary to provide sufficient networking and computing infrastructure to support low latency and fast response times for IoT applications. As stated earlier, being far from users and devices, cloud-based IoT solutions face several challenges such as high response times and heavy loads. In the era of Big Data, sending extraordinarily large amounts of data to the cloud is inefficient and expensive.

When considering the weaknesses and incompleteness of cloud, fog, and mist, several researchers independently propose

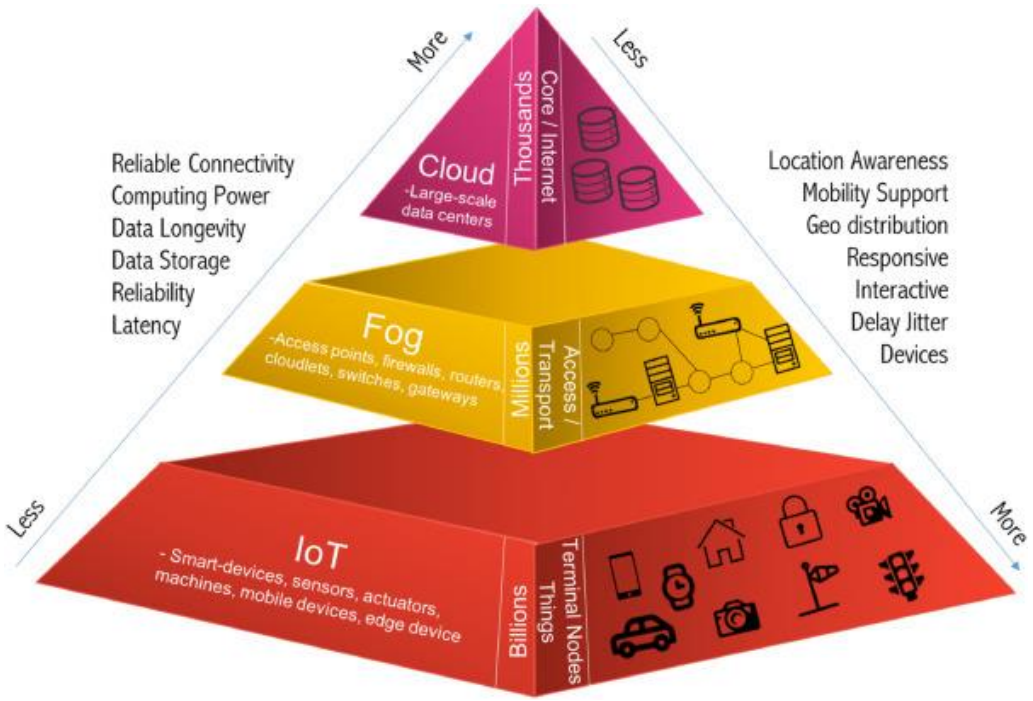


Fig. 4. IoT Three-layer architecture, proposal from A. Yousefpour et al. [20].

a three-layered architecture for IoT-fog-cloud ecosystems. The proposition considers IoT-to-cloud and fog-to-cloud interaction, as well as fog-to-fog communication in order to reduce service delay by load-sharing [22], and IoT-to-IoT for service effective re-use provided directly on devices. The three-layers are as follows:

- IoT layer, where the IoT and end-users are located;
- Fog layer, where fog nodes are placed;
- Cloud layer, where distributed cloud servers are located.

The way in which IoT and fog nodes as well as cloud servers operate and interact is as follows:

- IoT nodes can process requests locally, request service from known IoT nodes in a neighborhood, and send requests to a fog node or to the cloud.
- Fog nodes can process and forward requests to other fog nodes in the same domain or forward the requests to the cloud.
- Cloud servers process requests and send the response back to the fog or IoT nodes.

The aim is to minimize service delay for IoT devices based on fog and mist computing. Because the fog layer lies between IoT devices and the cloud, it can handle a majority of IoT service requests in order to reduce the overall service delay.

*Fig. 4* outlines a three-layered IoT architecture proposal from A. Yousefpour et al. [20]. Considering the ability of fog and mist computing to process requests and provide services, the proposed architecture can be the continuous nebula solution outlined earlier, where IoT within a complete stack of network can form a distributed computing system to share data resources, reduce response times, save energy consumption, distribute computation, and reduce application workloads.

## XIII. Network Modeling

The communication network modeling in framework and middleware is a very important factor for architecture. Network modeling has an impact on application performance, task distribution, data transaction, as well as the nature of error detection. Traditionally, IoT uses the following types of communication network modeling: point-to-point, client-server, and publish-subscribe. In this paper, we also suggest an additional multicast routing model.

### *A. Point-to-Point Communication Model*

Point-to-point is the simplest communication model, where two nodes establish direct connection and high-bandwidth data exchange is initiated. TCP is a point-to-point network protocol. The benefit of point-to-point model is that by having a direct connection, two applications can stream large amounts of data. Point-to-point modes do not consider multiple connections.

### *B. Client-Server Communication Model*

The client-server is a scalable communication model of point-to-point connection where one node has a role of server and other nodes are clients. The client-server model is a many-to-one architecture (*Fig. 5*), where one server connects and

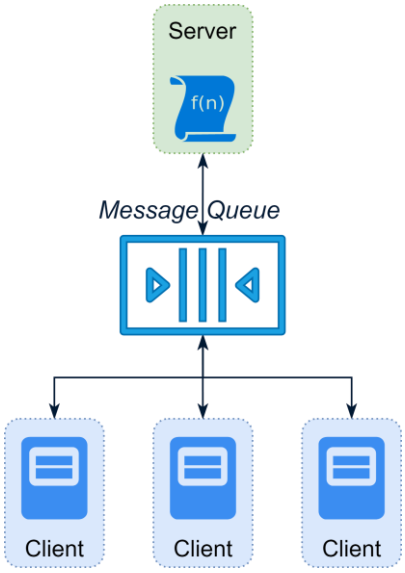


Fig. 5. Client-server communication model.

communicates simultaneously with many client nodes capable of scaling. Normally, IoT devices connect as clients to a remote server in order to send data. The client-server network model is used when information is centralized, for example in database and file servers. However, the server can distribute collected data from several connected client nodes. The benefits of the client-server model are that clients can be scaled; clients can use more powerful resources of the server to execute tasks and store large data. Since servers and clients are connected via particular IP-addresses and port numbers, client-server modeling is not considered to create a network mesh, because with this model dynamic network creation is a challenging task.

### *C. Publish-Subscribe Communication Model*

Publish-subscribe (PubSub) is a communication model where messages are passed between application nodes called “publisher” and “subscriber”. In this model, the publisher is a node that writes the data called topic, the subscriber is a node running in the same or another application to read the data, and data is then distributed via a physical databus service, in some systems called broker. In the PubSub model, messages flow out of a centralized server and the nodes are distributed in the network. PubSub is useful for distributing large quantities of information in the network. The benefits of PubSub are scalability and fault tolerance, the decoupling of publishers and subscribers so that publishers are not aware of subscribers, the ability to distribute data in the network, and the creation of a mesh of publishers and subscribers. In PubSub, since publishers and subscribers are not aware of each other and communication is provided via topics (including request-response mechanisms), publishers may send a lot of unnecessary messages where a subscriber may not exist. because no subscriber of a certain topic exists in the network. In this case, it may not be the most optimal resource consumer. Some PubSub solution providers optimize this issue by storing combined lists of publishers and subscriber topics locally. With this solution, network communication is optimized at the expense of local machine resources, which in some cases can

be unjustified. In addition, due to intermediate remote broker node, PubSub is not the fastest communication model.

### D. Multicast Routing Communication Model

Multicast routing is a communication model where nodes are connected to each other logically and can communicate through one or more physical message routers. One application can have more than one node, where every node is a software component. One node can be connected to several message routers, but two nodes communicate through one router. Each connection with a message router defines the communication channel. There can be several channels and channels can differ by priority, protocol, and physical network.

Physically, the message router is a server and every application is a physical connected client. Two or more applications communicate via the message router without awareness of the IP-address of each other. Applications in such a model provide services as logical servers and the network can have an unlimited number of logical servers. An application at the same time can be a server and a client without setting up any physical connection except with the message router. Similar to a traditional client-server connection, the client's logical connection is authorized on a logical server. Based on these characteristics, the multicast routing model allows any application to use and provide multiple services as well as create a mesh of authorized client-server logical connections. This allows to create a mesh of services, applications scale dynamically, and messages are distributed in the network and routed directly from the source to multiple targets without knowing the physical location of each target(s). *Fig. 6* demonstrates prioritization of two communication channels in multicasting routing model. *Fig. 9* demonstrates multicasting message router and messaging of two applications.

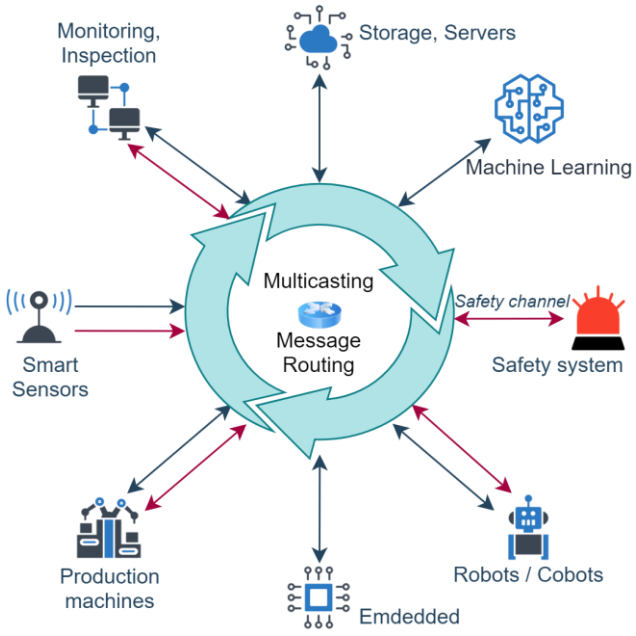


Fig. 6. Multicasting message routing with two communication channels connecting various nodes and services in a smart factory.

The benefits of using multicast routing are connection abstraction, connection authorization, a higher degree of privacy, network mesh, application scalability, fault tolerance, dynamic connection, message distribution, and optimized message streaming due to databus absence and streaming by request. Communications via message router is not the fastest, since unlike other client-server direct peer-to-peer connections, messages are passed through additional routing nodes.

### E. Networking Model Summary

*TABLE II* shows a comparison of characteristics of Point-to-Point (P2P), Client-Server (C/S), Publish-Subscribe (P/S), and Multicast Routing (MCR) network modeling.

Summarizing the network model, multicasting routing combines the key features of client-server and publish-subscribe networks, which encompass connection abstraction, connection scalability, data distribution, data-optimized streaming, service meshing. Data privacy is higher due to the possibility of service access authorization and access to services on-demand. The important fundamental characteristic of network communication optimization is data distribution on demand. The use of local resources depends on the solution provider.

## XIV. CONNECTIVITY FRAMEWORKS

S. Yacoub and H. Ammar correctly noted "*The most difficult part of building software is not coding; it is the decisions you make early at the design level. Those design decisions live with the system for the rest of its lifetime.*" [23]. It is very important that application and system architects choose the technology, architecture, and application frameworks suitable for the business. Often, such decisions are made during the application development phase and therefore

TABLE II. NETWORK MODELING CHARACTERISTICS.

| Feature List | Network models | | | |
|---|---|---|---|---|
| | ***P2P*** | ***C/S*** | ***P/S*** | ***MCR*** |
| Communication speed | ✓ [a] | ✓ [a] | ✓ [b] | ✓ [b] |
| Connection abstraction | × | × | ✓ | ✓ |
| Connection scalability | × | O | ✓ | ✓ |
| Messaging on demand | ✓ | ✓ | × | ✓ |
| Data access authorization | ✓ | ✓ [c] | × | ✓ |
| Data privacy | ✓ | O [d] | × | ✓ |
| Message distribution | × | O [e] | ✓ | ✓ |
| Service meshing | × | × | ✓ | ✓ |
| Fault tolerance | × | O | ✓ | ✓ |

a. Communication is fast if no additional broker node exists.
b. Communication may be delayed up to a few milliseconds due to additional node.
c. Centralized data access authorization on server side.
d. Since data is collected on server side, no client can guarantee data privacy.
e. Tasks can be distributed on client nodes via centralized server.

- ✓ — feature is strong;
- O — feature is average;
- × — feature is poor or absent.

subsequent changes of chosen architecture and technology become more difficult and expensive, and sometimes impossible.

In January 2020, IoT Analytics reported that there are 620 publicly known IoT development platforms [6]. For objective reasons, assuming that the majority of solutions are focused on cloud solutions or specific computing solutions such as fog or edge computing, there are few that are able to provide solutions simultaneously for fog, edge and mist (edge-centric computing).

This paper reviews service-driven solutions to introduce existing application development platforms suitable for mist, edge, and fog computing. Edge-centric platforms can be grouped into data-centric, action-centric, and interface-centric servicing models. As examples of service-driven application development frameworks, we consider and compared AREG IoT SDK (suggestion of AREGtech) [25], Connext DDS [26], gRPC [27], ROS [28], WCF and MSMQ [29].

The communication protocol is a system of rules expressed by algorithms and data structures that allow two or more entities of a communication system to transmit information. There are studies and papers comparing different features and performances in several benchmarks of widely used protocols in the IoT industry such as OPC UA, ROS, DDS, and MQTT. Based on research results, the protocols have an effect on software performance [24]. Analysis of these communication protocol is not included in the scope of this paperwork.

### A. *Data-Centric Messaging*

The Data Distribution Service (DDS) for real-time systems is an Object Management Group (OMG) [30] machine-to-machine (M2M) standard that aims to enable dependable, high-performance, interoperable, real-time, scalable data exchanges using a publish-subscribe (PubSub) pattern. DDS is a networking middleware that implements a publish-subscribe pattern (also called Databus) for sending and receiving data, events, and commands among the nodes. Nodes that produce information called "publishers". Publishers create information called "topics" (e.g., temperature, location, pressure, etc.) and publish values of "samples" (instances). DDS delivers the samples to subscribers that declare an interest in that topic [41]. Any node can be a publisher, subscriber, or both simultaneously. *Fig. 7* is a visualization of publisher and subscriber relationship via Data Distribution Service.

DDS addresses the needs of applications that require real-time data exchange, and the publish-subscribe model is for distributed computation. The key benefit is that applications that use DDS for communications are decoupled, i.e. applications communicate with the DDS databus rather than with each other. In particular, applications never need information about other participating applications, including their existence or locations. DDS transparently handles message delivery without requiring intervention from user applications, determining who should receive the messages, where recipients are located, and what happens if messages cannot be delivered [41].

DDS connectivity-based applications are data-centric. Each application connects to a DDS databus to write or read data. The

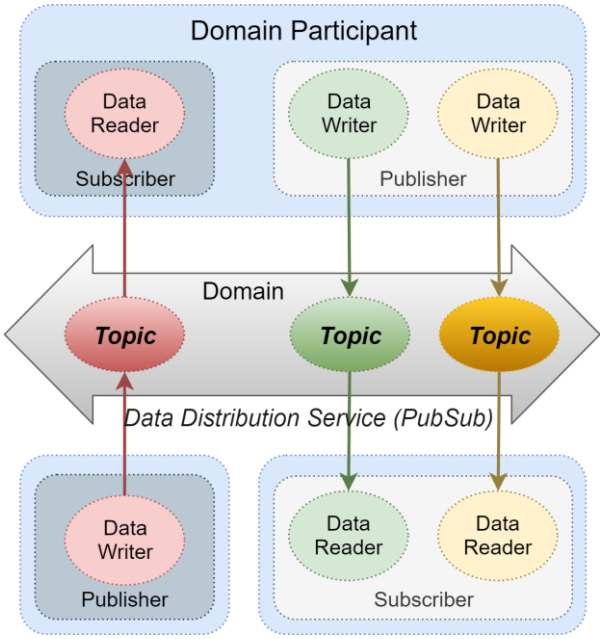


Fig. 7. Publish-subscribe data-centric communication architecture.

DDS applications are fault tolerant, fully decoupled, and distributed in the network. Applications are bound together in the DDS domain with unique identifiers. There can be more than one domain in the network. Bind data writer and data reader must have the same topic and data type. Applications in one DDS domain cannot subscribe to data published in a different DDS domain. Multiple DDS domains are allowed to have multiple virtual distributed systems over the same physical network [26].

The publish-subscribe model provides anonymous, transparent, many-to-many communications. Each time an application sends a DDS sample of a particular topic, the middleware distributes DDS sample to all the applications that want that topic. The publishing application neither needs to know how many applications receive the topic, nor where the applications are located. Similarly, subscribing applications do not specify the location of the publications. In addition, new publications and subscriptions of the topic can appear at any time, and the middleware will automatically interconnect them. Thereby, the model creates a mesh of publishers and subscribers within each domain.

Different providers have implemented different DDS mechanisms. One widely used mechanism is a data broker (or data server) as a separate physical node for keeping the list of applications to distribute subscribed topics. Another mechanism allows each application to keep a list of applications subscribed to per same topic. In addition, DDS keeps the list of applications and publications for each topic to which the application has subscribed. This database of information is used in the domain participant discovery process to determine if an application has a matching data writer and reader of a particular topic.

The example frameworks of using DDS and publish-subscribe networking models are Connext DDS [26] and Robot

Operating System (ROS) [28]. Both frameworks provide a wide list of useful features and can run on PC and embedded systems with multitasking operating systems. User manuals and API documentation can be found on their website. Other popular publish-subscribe messaging providers are Message Queuing Telemetry Transport (MQTT) based middleware solutions.

*B. Action-Centric Messaging*

Remote Procedure Call (RPC) is a protocol-based paradigm of Inter-Process Communication (IPC) used in distributed computing to cause a function called in one process to be executed in another target process running on the same or different computers of the same network. Since processes have different address space, the procedures cannot be called programmable. Instead, an RPC is initiated by the client, which sends a request message to a known remote server to execute a specified procedure with supplied parameters. The remote server then responds to the client's request. While the server is processing the request, the client can be either blocked or unblocked. A blocked or unblocked state of the client application defines a synchronous or asynchronous communication model. Some RPC connectivity frameworks may provide both communication models. An important difference between remote procedure calls and local calls is that remote calls can fail because of unpredictable network problems. Callers must deal with such failures without knowing whether the remote procedure was actually invoked [31].

Applications written using RPC are action-centric and normally use a client-server networking model. There are many variations of RPC implementations and request-response communication protocols.

Irrespective of the fact that request-response protocols in distributed computing existing has been around since the late 1960s, there is no single standard in protocols. However, the sequence of events is normally as follows [32]:

- The client calls a local procedure call of a stub, with parameters pushed on to the stack in the normal way.
- The client stub packs the parameters into a message and makes a system call to send the message.
- The client's local operating system sends the message from the client machine to the server machine.
- The server's local operating system receives incoming packets and passes them to the server stub.
- The server stub unpacks the parameters from the message and calls the server's request procedure.
- The server's reply traces the same steps in the reverse direction.

RPC is a technique for constructing client-server-based applications, when the called procedure need not exist in the same address space as the calling procedure. The two processes may be on the same or different systems with a network connecting them. The transport independence of RPC isolates the application from the physical and logical elements of the data communications mechanism and allows the application to use a variety of transports [33].

Different providers have implemented different RPC mechanism. Some are using direct peer-to-peer connection where each client is directly connected to the server application and the server application has an internal message queue to receive and process messages [34]. Others provide message-queue service where clients and servers communicate via special queueing service as is visualized in *Fig. 8*.

An example framework using peer-to-peer client-server RPC communication is gRPC [28]. An example framework of using synchronous RPC communication is ROS [28]. An example framework using additional message queue service for RPC communication is WCF (Windows Communication Foundation) [29]. Connext DDS [26] as well uses request-response protocol and executes it via a publish-subscribe model.

Another solution is the Distributed Component Object Model (DCOM) Remote Protocol, which exposes application

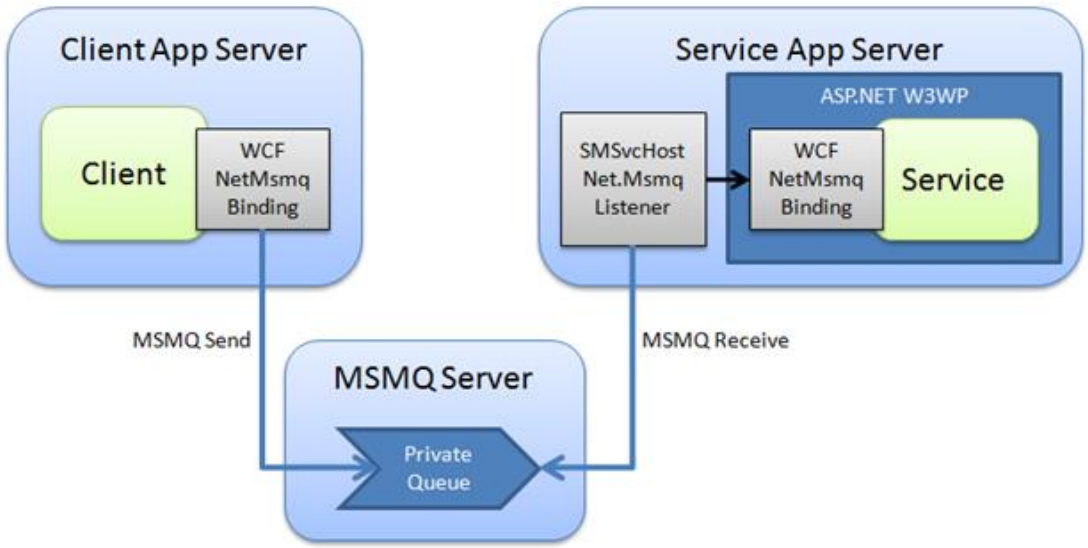


Fig. 8. Message queue server and communication in WCF [34].

objects via remote procedure calls (RPCs) [35]. This is considered in more detail in the next topic.

### C. Interface-Centric Messaging

Object Remote Procedure Call (ORPC) is a concept of remote procedure call whose target is an interface on an object. The target interface (and therefore the object) is identified by an interface identifier. The ORPC is a protocol for exposing application objects by way of RPCs. It is built on top of the RPC, and relies on its authentication, authorization, and message integrity capabilities. Applications obtain remote object reference and make ORPC calls. ORCP is a technique of constructing meshed network of distributed services, where applications are enabled to provide reusable services. *The programmable (logical) client objects request services from programmable (logical) server objects without having to understand where the server object is running in the network.*

There is no restricted client or server application in such a network. The network is a mesh of logical client and server objects communicating through designed service interfaces. A server component is an implementation of one and more service interfaces (objects). Client objects request activation of server components, and once a component is available in the network, they invoke predefined methods (operations) on server components. When a method is called, the system sends serialized operation with arguments in ORPC request buffer to be executed remotely. The remote server object replies with ORPC response buffer containing serialized arguments.

Applications using ORPC communication are interface-centric and use objects of predefined interfaces, similar to object-oriented programming. It is possible to instantiate multiple instances of the same object and the system is able to handle them in a transparent manner so that a message can be dispatched to call an exact function of an exact object. Interface-centric messaging solutions are normally based either on a multicast routing network communications model or on a client-server network communication model. There is no protocol restriction, although the communication must be bi-directional to have the possibility of sending messages to all connected nodes in the network.

Different providers offer different mechanisms of object instantiation, method invocation, and data management and we provide here short introductions to AREG IoT SDK [25] and DCOM [35]. Despite similarities in the provision of distributed services, each framework differs by agenda. These solutions are described separately.

#### 1) AREG IoT SDK

Traditionally, IoT devices are connected clients of cloud or fog servers to stream data from sensors for further processing. Since data is generated at collected devices level, it makes sense to provide network accessible (public) services directly on things. Such a concept changes not only the role of connected devices, but also primarily solves the following challenges:

- It significantly increases data privacy, which is a very important factor for sensitive data.
- It decreases data streaming, which is a fundamental condition for optimizing network communications that has a direct impact on energy consumption.
- It renders devices more autonomous and more intelligent by providing network services directly in the environment of data origin.

The benefits of service-enabled devices are described in the mist computing section of this paper. AREG IoT SDK (or AREG SDK) is the suggestion of AREGtech. AREG IoT SDK is an asynchronous communication framework that is based on a multicasting communication model and helps to develop a service network mesh being distributed to any application at any node. The framework helps to create applications to run on embedded systems and high-end machines with 32- or 64-bit multitasking operating system [25].

AREG SDK consists of two main component parts: a middleware framework and a message multicast router, which is a service running in the network (can run on gateway

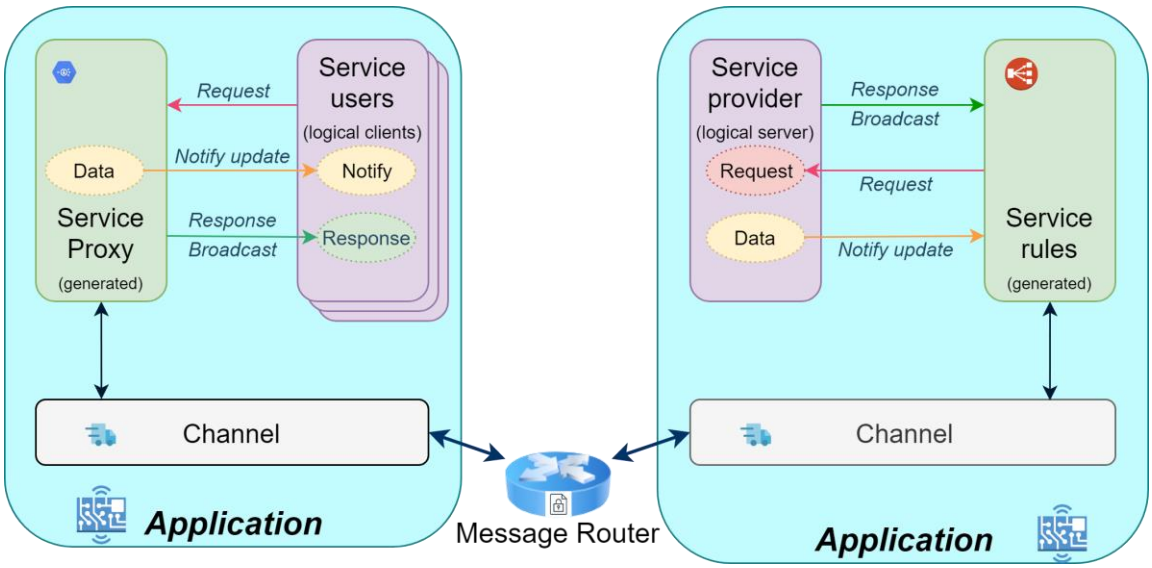


Fig. 9. Message routing, client and server objects, and ORPC calls in interface-centric AREG IoT SDK [25].

hardware) to route ORPC messages between applications in real-time communication mode. Applications can connect to one or more routers, where every connection is a communication channel with high or normal priority. Channels may differ by priority, protocol, and network. Applications based on AREG IoT SDK automatically create a mesh of services without additional programming effort.

The service providers are the logical server objects, and the service consumers are logical client objects. In AREG SDK a single component simultaneously can be a client and a server. Client objects find and connect to their corresponding server objects via an automated Service Discovery module integrated in each process, which is the key instance in the process of forming a mesh of services. Clients receive connection and service availability notifications only if the appropriate server object accepts a connection. A client connection can be programmed to be rejected on the server side. The network automatically drops any ORPC message sent by a non-authorized client object.

Unlike many other frameworks, the messages in AREG SDK are not dispatched in centralized points. The servicing components are instantiated, and the messages are dispatched in thread context so that the ORPC messages are personalized and they are delivered to an exact application, exact thread, and exact target object to raise the method. Thereby, AREG SDK is thread safe and it is guaranteed that all messages are executed in the thread context of service. There is no need to manually serialize, deserialize, and dispatch messages, and these procedures are automated in generated codes created from the service interface document. The service interface document can be created in a graphical tool, provided that a service interface code generator can generate the following objects (*Fig. 9*):

- a client-proxy to cache automatically updated data and provide client-server communication;
- a server stub, which contains rules for controlling requests and subscriptions from clients, as well as providing skeletons of request services for implementation;
- client base objects to extent and overwrite appropriate response, multicast information, data update or request fail notifications;
- service-specific events for personalized messaging;
- service-specific data types.

The service interfaces are containers of attributes (data), requests, responses, and information, which are special multicasting methods for delivering several objects in one call. Similar to a traditional client-server model, clients call requests and the result is replied via asynchronous responses. AREG SDK provides similar publish-subscribe messaging features whereby clients can dynamically subscribe to certain attribute value update notifications as well as to multicasting information of the provided service interface. Additionally, the client can subscribe to particular responses without a preliminary call request, in order to receive a response of a request triggered by another client object.

Provided services within a network must have unique names, and services must be accessed by those names. One service in a single network can be instantiated more than once and each must differ by name. Visibility is decided by service interface type during design time and there are three types of service interfaces supported in AREG SDK (*Fig. 10*):

- *Remote Service.* Visible within a single network and accessed by any client object running on any machine in the network. Thus, service provider components must have unique names within the network. The same service provider component can be instantiated more than once if service names differ.
- *Local Service.* Visible only within a single process and is not accessed externally. Normally, it is used to protect private data that is not supposed to be shared. The components have unique names within a single process. The clients must run in the same process.
- *Internet Service.* Components that implement internet service interfaces, are not web-service providers. This

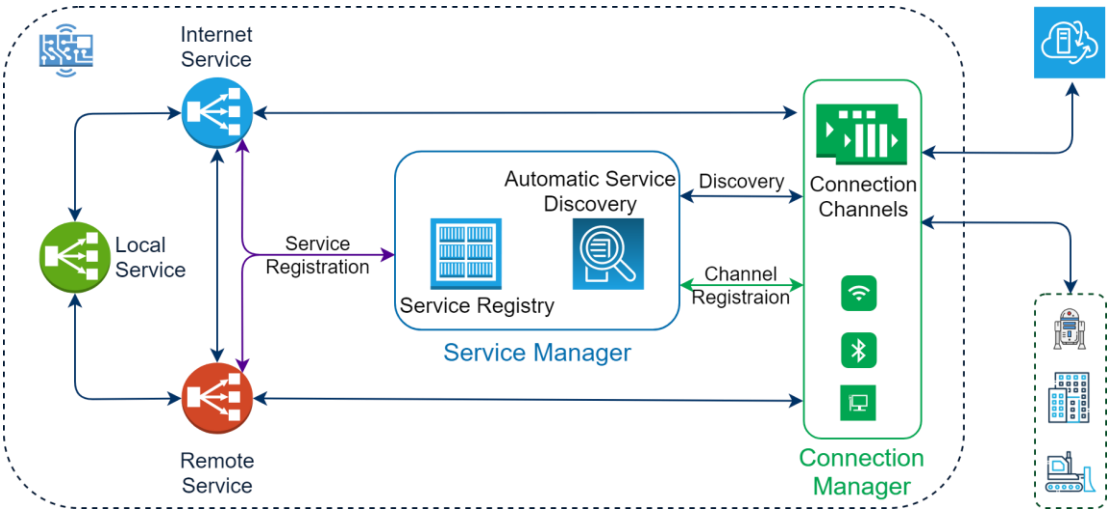


Fig. 10. Local, public and internet services, service discovery, connection manager and message flow in interface-centric AREG IoT SDK [25].

interface is designed to support traditional web communication as a client application and is normally used by application objects to send data to the web.

The listed service interface categories make AREG IoT SDK good industry framework for developing multitasking embedded applications, service-enabled applications at mist and fog, and being able to be an IoT enabler of connected things.

*2) DCOM*

Distributed Component Object Model Remote Protocol (DCOM) extends the Component Object Model (COM) over a network by providing facilities for creating and activating objects, and for managing object references, object lifetimes, and object interface queries [35].

Applications use the DCOM client to obtain object references and make ORPC calls on the object. The DCOM client in turn uses the RPC Protocol Extensions to communicate with the object server. The object server constitutes an object resolver service and one or more object exporters. Objects are contained in object exporters. Objects are the target of the ORPC calls from the client. The activation is an act of creating or finding an existing DCOM object via class factory. Activation consists of sending the following to the object activation service on the remote machine [35]:

- A class identifier (CLSID)
- One or more interface identifiers (IID)
- Optionally, an initialization storage reference

The CLSID identifies the class of the object to be created. The IIDs identify the interfaces on the newly created object that the client is asking for and, if specified, the storage reference identifies persistent store with which the newly created object is to be initialized after creation. Activation returns object references to the client application. The client application can send or receive object references as part of ORPC calls [35].

An object exporter is a container where objects are created, called, and released. An object is required to be contained within a single object exporter and is required not to span multiple object exporters. An object exporter can be a thread, a process, or a machine. The DCOM uses reference counts to manage object lifetimes. To ensure that object resources are recovered in the event of machine failures or network failures, the DCOM incorporates a garbage collection mechanism. The object resolver service performs activation, resolution, garbage collection, and server aliveness tests [35].

*3) Other frameworks and technologies*

A few others known interface-centric technologies and frameworks not reviewed in this paper are as follows:

- CORBA (Common Object Request Broker Architecture) [36]. It is a standard defined by the Object Management Group (OMG) and is designed to facilitate the communication of systems that are deployed on diverse platforms.
- Some/IP [37]. It is a known middleware in the Automotive industry that was designed to be integrated into AUTOSAR 4.x releases. CommonAPI [38]. It is a known framework in the Automotive industry designed by GENIVI Alliance.
- Windows Communication Foundation (WCF) [29]. It is a runtime and a set of APIs in the .NET Framework for building connected, service-oriented applications.

There is an increasing number of other frameworks in the market designed with various network communication models and all designed to bring new ideas and business cases for fulfilling the needs of network edge.

### *D. Connectivity Solutions Summary*

All three data-centric, action-centric, and interface-centric messaging models have both advantages and disadvantages. Whereas data-centric messaging provides more abstraction, action-centric messaging exhibits faster communication by using simple RPC and peer-to-peer connection. Interface-centric messaging, similar to multicasting routing solutions, may combine features of data- and action-centric solutions. However, due to additional routing nodes, it is still not the fastest means of communication, and there may be a message delay up to several milliseconds, depending on router load. Some interface-centric solutions by other providers can still consume more resources than other solutions, and therefore they are not intended to be a solution for embedded development with limited resources. Such tasks can be carried out by AREG IoT SDK that is designed with a focus on embedded systems and by frameworks like Some/IP or CommonAPI targeted to the Automotive industry.

For a better understanding of frameworks' differences, let us consider some simple and very common use cases. Often during product development, there comes a moment when an existing API requires updates. Let's assume, an IoT application provides API of Data1 and Data2. In a new version of the product, there is a need to provide new API of Data3, and for any reason aggregate all three data to send at once. For example, in order to perform an action, all 3 data should change the value. Instead of saving the "data change" flag for each value on target, it is easier if the source notifies and sends all 3 values.

In the case of data-centric solutions, the manufacturer still needs to support the old version of Data1 and Data2. Otherwise, the existing connected applications (especially those provided by third parties) will not work. Therefore, the manufacturer has to support old and new APIs. In traditional data-centric solutions, since nodes are decoupled, the application streams three data to broker (Data1, Data2, and a structure of combined data). As was outlined earlier, network streaming can consume more energy power than CPU. Therefore, the provision of new API of combined data may be not an optimal solution.

In the case of action-centric solutions, optimization depends on the version of connected applications. For applications that are using an older version of API, will still be two request calls to set Data1 and Data2. For applications are using a new version of API, then all three datasets can be set by one request call. So that, the product should support both versions of API. This neither optimizes nor regresses the communication, and it depends on API version used by connected applications. However, by that stage, additional problems relating to

controlling versions, APIs, and software business logic would arise. Nevertheless, when compared to data-centric solutions, action-centric solutions don't execute unnecessary streaming in the network.

In interface-centric cases, there are two different solutions and each solution changes the situation. In the example of AREG SDK, here are the presented solutions:

- *Solution #1:* manufacturer provides data as a part of service interface attribute so that there are Data1 and Data2 attributes, and the new attribute is a container of all three data. Similar to action-centric solutions, the old applications can receive Data1 and Data2 by subscribing to appropriate attribute changes, and new applications can subscribe new API. So that, consumers receive data based on used API version.
- *Solution #2:* instead of providing data as an attribute of service, use multicasting information and pass data on the s parameter list. The new API is nothing more than a new parameter on the parameter list. The applications with old API consume first two parameters listed in information, and the 3rd will be automatically ignored. It is automated and does not require any refactoring. Applications with new API use all data.

As was seen earlier, interface-centric solutions provide more possibilities for optimizing communication. In this simple use case, the number of messages were not changed. Instead, it changed the size of the ORCP message. Therefore, network communication is still optimal.

In summary, here are the strengths of the aforementioned frameworks (we're excluding DCOM, because, when compared to others, it was not intended for small systems and therefore requires more resources):

- *ROS* was developed for robotics. It offers design tools and geometry libraries. The framework is data- and action-centric so that applications can either stream data or provide request services.
- *Connext DDS* is a data-centric architecture with automated topic discovery. Unlike other data-centric solutions, data streaming is more optimized due to lack of service brokers. The application nodes keep a list of topics, publishers and subscribers, so that the system can optimize streaming. In addition, it has an API designing tool and code generator.
- *gRPC* is an action-centric solution with partially-automated service discovery. Due to peer-to-peer connection, there is no intermediate node between clients and a centralized server, so that the communication is faster. It uses Protocol Buffers (Protobuf) developed by Google, so that applications based on gRPC can benefit from the services and used code generators provided by Google.
- *AREG IoT SDK* is an interface-centric solution with automated service discovery. The applications based on AREG IoT SDK can benefit network of service mesh and easy integration into a fog network. Developers using AREG SDK can benefit from automated, personalized and thread-oriented messaging that invokes methods in thread safe mode so that it eases multitasking development. Interface design and code generator tools are an additional help to save development time.

When considering the frameworks, it should be noted primarily that every solution inherits networking model features. Secondly, frameworks have properties that evolve, so in the future some of them will take on new and more practical forms. Before using a certain framework, decision makers must consider the architecture and communication model.

## XV. CONCLUSION

The Internet of Things promises a vast potential to accelerate digital transformation and provide massive benefits to many industries. Due to the wide range of benefits, the number of connected IoT devices, networks, and big data centers are all on the rise. This whitepaper considered different edge-centric computing paradigms, networking models, frameworks, and benefits of combining technologies with the cloud. Fog with mist are extending cloud to the edge and are promising solutions for minimizing communication latency down to milliseconds and handling data produced by IoT devices. The multicasting routing is a solution for creating a multi-channel network service mesh that can combine publish-subscribe features and client-server communication models. The interface-centric messaging frameworks can be a combination of request-response and publish-subscribe patterns. The interface-centric frameworks for embedded systems can be an integral part of fog that, in combination with cloud, can form a continuous nebula that can be regarded as a viable and fully functional new concept in IoT in the future. AREG IoT SDK, described in this paperwork, is an interface-centric framework based on a multicasting communications model that can be used in IoT for service-enabled applications development for embedded and high-end systems. At a time of technology complexity, effective application development platforms provide less complication and more automation, design tools, and code generators to ease the development process and help to reduce risks as well as overall application development costs.

Today, data centers consume 2% of the world's produced energy. By 2030, this number is forecast to reach 8%. According to EirGrid, date centers and other large users in Ireland alone will consume 29% of Ireland's electricity by 2028 [39]. This fact provides an additional incentive for providing more computing at the IoT network edge. Fog computing, mist computing, and interface-centric solutions for embedded devices can optimize systems for effective handling of big data, which are often time-sensitive and security-critical. This optimized solution helps to reduce data streaming, and therefore, reduces energy consumption and costs.

## REFERENCES


[1] "Cisco Visual Networking Index: Forecast and Trends, 2017–2022 White Paper", Document ID 1551296909190103, whitepaper.

[2] B. Zhang, N. Mor, J. Kolb, D. S. Chan, N. Goyal, K. Lutz, E. Allman, J. Wawrzynek, E. Lee, and J. Kubiatowicz, "The Cloud is Not Enough: Saving IoT from the Cloud", University of California, Berkeley, 2015, whitepaper.

[3] J. Hyde, "IoT Forensics: Where the Internet of Things and Digital Evidence Intersect", M.S. in Computer Forensics, George Mason University: February 18, 2019: presentation [Workshop #6, "The Impact of Emerging Technologies on Digital & Multimedia Forensics", The American Academy of Forensic Sciences (AAFS) 71st Annual Scientific Meeting, Baltimore, MD: workshop, February 18, 2019].

[4] D. Tokar, "MachNation rates 16 IoT edge vendors in first-of-its-kind ScoreCard", MachNation, December 6, 2017, in press.

[5] C. Pettey and R. van der Meulen, "Gartner Says Global Artificial Intelligence Business Value to Reach $1.2 Trillion in 2018", Stamford, Conn., April 25, 2018, in press.

[6] K. L. Lueth. "IoT Platform Companies Landscape 2019/2020: 620 IoT Platforms globally". IoT Analytics, December 23, 2019.

[7] P. Mell and T. Grance. "The NIST Definition of Cloud Computing", (pp. 1-3, Rep.). Gaithersburg: National Institute of Standards and Technology. Special Publication: 800-145, September 2011.

[8] M. Iorga, L. Feldman, R. Barton, M. J. Martin, N. Goren, C. Mahmoudi. "Fog Computing Conceptual Model, Gaithersburg": NIST U.S. Department of Commerce, NIST Special Publication 500-325, March 2018.

[9] E. Hamilton. "What is Edge Computing: The Network Edge Explained". cloudwards.net., 2019-05-14.

[10] A. Reale. "A guide to Edge IoT analytics", IBM, February 2017

[11] "What is Edge Computing?", General Electric blog.

[12] C. Chang, S. Narayana S. Buyya and R. Buyya. "Fog and Edge Computing: Principles and Paradigms", chapter 1, "Internet of Things (IoT) and New Computing Paradigms", Wiley STM.

[13] M. Kumar Yogi, K. Chandrasekhar, G. Vijay Kumar. "Mist Computing: Principles, Trends and Future Direction" , SSRG International Journal of Computer Science and Engineering (SSRG-IJCSE), volume 4 Issue 7, July 2017.

[14] F. Jalali, K. Hinton, R. Ayre, T. Alpcan, R. S. Tucker. "Fog Computing May Help to Save Energy in Cloud Computing", IEEE Journal on Selected Areas in Communications, Vol. 34, No. 5, May 2016.

[15] "OpenFog Reference Architecture for Fog Computing" Produced by the OpenFog Consortium Architecture Working Group, February 2017, p.3.

[16] N. Joshi. "Fog vs Edge vs Mist computing. Which one is the most suitable for your business?", Allerin Tech PVT Ltd, 18 July, 2019.

[17] N. Joshi. "How IoT can transform public transportation", Allerin Tech PVT Ltd, 26 May, 2019.

[18] R. Mahesa. "How cloud, fog, and mist computing can work together", IBM, March 6, 2018, article at IBM Developer.

[19] J. Preden. "Mist computing - IoT on the Edge", publisher: B. Sindar, IoT-Inc., August 2016 (an interview with J. Preden, Thinnect).

[20] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, J. P. Jue. "All one needs to know about fog computing and related edge computing paradigms: A complete survey", Journal of Systems Architecture, Vol. 98, September 2019, pp 289-330.

[21] "Fog vs Edge Computing", Nebbiolo Technologies Inc., Milpitas, CA, whitepaper, v. 1.1.

[22] A. Yousefpour, G. Ishigaki, R. Gour, J. P. Jue. "On Reducing IoT Service Delay via Fog Offloading", Networking and Internet Architecture (cs. NI), IEEE Internet of Things Journal, vol. 5, no. 2, pp. 998-1010, April 2018.

[23] S. Yacoub and H. Ammar. "Pattern-Oriented Analysis and Design: Composing Patterns to Design Software Systems". Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003 (preface).

[24] S. Profanter, A. Tekat, K. Dorofeev, M. Rickert, A. Knoll. "OPC UA versus ROS, DDS, and MQTT: Performance Evaluation of Industry 4.0 Protocols" In Proceedings of the IEEE International Conference on Industrial Technology (ICIT), Melbourne, Australia, 2019.

[25] AREG IoT SDK, product of AREGtech. [Source: https://aregtech.com].

[26] Connext DDS, product of Real-Time Innovations, Inc. (RTI). [Source: https://rti.com].

[27] gRPC, product of Google LLC. [ Source: https://grpc.io].

[28] ROS, open source project. [ Source: https://ros.org].

[29] "Windows Communication Foundation", Microsoft Corporation. [Source: https://docs.microsoft.com/en-us/dotnet/framework/wcf/].

[30] "Data Distribution Service (DDS)", Object Management Group (OMG), specification v. 1.4, April 2015. [Source: www.omg.org/spec/DDS/1.4]

[31] "Remote Procedure Call", an article at Wikipedia.org, [Source: https://en.wikipedia.org/wiki/Remote_procedure_call].

[32] R. H. Arpaci-Dusseau and A. C. Arpaci-Dusseau. "Distributed Systems", Arpaci-Dusseau Books, 2014.

[33] Dr D. Marshall. "Remote Procedure Calls (RPC)", a tutorial on Open Network Computing Remote Procedure Call (ONC RPC), Cardiff University, January 5, 1999.

[34] T. Hollander. "MSMQ, WCF and IIS: Getting them to play nice (Part 1)", Microsoft Corporate, personal blog at Microsoft MSDN, July 2008.

[35] "[MS-DCOM]: Distributed Component Object Model (DCOM) Remote Protocol", Microsoft Corporation, Open specifications. [Source: https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-dcom/].

[36] "Common Object Request Broker Architecture (CORBA)", a standard defined by the Object Management Group (OMG). [Source: https://corba.org].

[37] Dr. Lars Völker. "Scalable Service-Oriented Middleware over IP (SOME/IP)", an open source project. [Source: https://some-ip.com].

[38] CommonAPI, GENIVI Alliance, Inc., and open source project. [Source: https://docs.projects.genivi.org].

[39] R. Carroll. "Why Irish data centre boom is complicating climate efforts", The Guardian, an article at The Guardian, January 2020.

[40] K. L. Lueth. "IoT 2019 in Review: The 10 Most Relevant IoT Developments of the Year", IoT Analytics GmbH, January 2020.

[41] "Data Distribution Service", an article at Wikipedia. [Source: https://en.wikipedia.org/wiki/Data_Distribution_Service].