Noname manuscript No.
(will be inserted by the editor)

Advances in Cloud Computing, Wireless Communications and the Internet of Things

Gopika Premsankar · Mario Di Francesco

Received: date / Accepted: date

Abstract There is a growing amount of data generated by a variety of devices of the Internet of Things (IoT). Sharing economy applications can leverage such data to provide solutions of high societal impact. Several technologies together enable the collaborative use of data through software services. This chapter describes the key developments in these technological areas. In particular, it describes advances in cloud computing that have resulted in new software architectures and deployment practices. Such improvements enable the rapid creation and deployment of new services on the cloud. Next, it highlights recent developments in wireless networks that allow heterogeneous devices to connect and share information. Furthermore, this chapter describes how IoT platforms are becoming interoperable, thus fostering collaborative access to data from diverse devices. Finally, it elaborates on how the described technologies jointly enable new sharing economy solutions through a case study on car sharing.

Keywords Internet of Things · IoT · cloud computing · mobile networks · LPWAN · LoRa

1 Introduction

Several advances in the field of Information and Communications Technology (ICT) have influenced how we are able to share our time, material and skills [46]. Indeed, there are many examples of online sharing platforms such as

G. Premsankar
Department of Computer Science, School of Science, Aalto University, Finland
E-mail: gopika.premsankar@aalto.fi

M. Di Francesco
Department of Computer Science, School of Science, Aalto University, Finland
E-mail: mario.di.francesco@aalto.fi

The original publication is available at www.springerlink.com DOI: `https://doi.org/10.1007/978-3-030-35032-1_6`

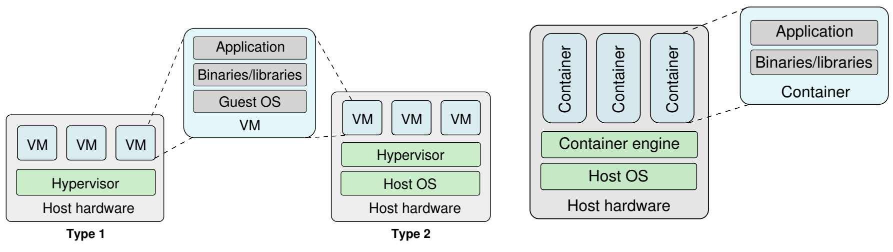
Airbnb, Uber, marketplaces and food sharing applications. The success of these applications depend on several factors. First, the ubiquitous availability of *Internet connectivity* enables us to access such services everywhere and at any time. Furthermore, the advent of new *cloud computing* service models enables application developers to quickly deploy new services and functionality. The economic barrier to entry has been lowered with the availability of pay-per-use, on-demand computing and data storage resources. Now, we are witnessing an increasing amount of machine-generated data from *Internet of Things* (IoT) devices such as sensors, household appliances, wearable devices, vehicles and much more. Indeed, sharing economy applications can make use of the wide variety of data and enable sharing in a more immersive and collaborative manner. The IoT enables new services that rely on seamless inter-operation between home networks, neighborhood networks and global suppliers of goods and services [35]. This chapter describes recent developments in the fields of cloud computing (Section 2), wireless connectivity (Section 3) and IoT (Section 4) that enable such new applications and services. Section 5 describes how these technologies together enable a car sharing application. Finally, Section 6 provides some concluding remarks.

2 Cloud Computing

Traditionally, software applications were deployed on bare metal servers. Developers typically had to invest in the infrastructure on which services were deployed, and manage the hardware in addition to the application itself [36]. Furthermore, they had to overprovision their deployments, i.e., make more computing resources available than required so as to meet the peak demand. However, this resulted in under-utilizing the hardware when the demand is low. The advent of *virtualization* enabled multiple virtualized servers (or instances) to run on a single physical machine. The virtualized instances, also known as Virtual Machines (VMs), run in isolation from each other and run their own operating system. This resulted in the initial wave of *cloud computing*, wherein cloud providers manage large data centers while developers (or service providers) deploy applications as VMs [31]. The main benefits of cloud computing are the availability of an infinite amount of resources (computing, network and storage) that can be used on-demand and released when no longer required [36]. This allowed software developers and organizations the flexibility to start with a lower level of resources and scale them as required. This also meant that computing resources could be treated as utility, which lowered the barrier for innovative services as a large initial commitment of resources was no longer required [48].

Cloud computing services are made available under different service models, described next.

– *Infrastructure-as-a-Service* (IaaS) allows the developers to deploy virtualized instances, typically VMs, within which custom software can be run.



(a) Hypervisor-based (b) Container-based

Fig. 1: Architecture of virtualized instances ((a) VMs and (b) containers) and host hardware.

In this paradigm, the end users manage the deployed instances including the OS, storage and networking [50].

- *Platform-as-a-Service* (PaaS) allows the end user to deploy applications on the cloud while the cloud provider manages the programming tools (software libraries, language runtimes), networking, storage and OS. Typically, the user controls the application settings and relies on the cloud provider for maintaining the underlying infrastructure [50].
- *Function-as-a-Service* (FaaS) [23] is a new paradigm that allows end users to deploy stateless functions on the cloud platform. The functions are executed only when explicitly invoked or triggered by an event (such as user input or database changes). The platform itself takes care of executing the functions and scaling them based on the demand. FaaS again relies on the cloud provider for managing the infrastructure; however, it differs from PaaS in that the functions are not billed when they are idle.
- *Software-as-a-Service* (SaaS) allows the end users to use applications provided by the cloud platform. The applications themselves are accessible through a thin client interface such as a web browser or an application programming interface (API). All aspects of the application and infrastructure are managed by the cloud provider [50].

2.1 Virtualization

Virtualization is the key enabler of cloud computing and its associated service models. The two most popular virtualization technologies are *hypervisor-based* and *operating system-based* (or *container*-based) virtualization.

Hypervisor-based virtualization

In hypervisor-based virtualization (Figure 1a), a software abstraction layer called a virtual machine monitor (VMM) or hypervisor lies between the VMs and the underlying physical hardware. The hypervisor manages the virtual machines and has full control of system resources. It provides an environment for execution that is identical to the underlying server [59]. The hypervisor provides complete isolation between VMs and also from the underlying hardware, thereby allowing multiple OSes to run at the same time. For instance, it is possible to run a VM with Windows OS on top of a machine with a Linux-based OS. There are two types of hypervisors: a *type 1* or *bare-metal* hypervisor runs directly on top of the host hardware, whereas a *type 2* or *hosted* hypervisor runs on top of the host OS. Examples of type 1 hypervisors include Xen [24] and VMWare ESX [51], and those of type 2 include KVM [43] and Oracle VirtualBox [75]. Hypervisor-based virtualization supports multitenancy and provides excellent isolation between VMs; however, it introduces an overhead that affects performance of the VMs [34, 56]. Moreover, the time taken to start a VM can be in the order of minutes as a complete OS needs to be started.

Container-based virtualization

Container-based virtualization (Figure 1b) is a lightweight form of virtualization that is becoming more prevalent on the cloud today [70]. This form of virtualization does not rely on a hypervisor and uses the host OS kernel-level features to provide isolation. Thus, virtualized instances (known as containers) do not need to run a separate OS [56]. Containers start faster than VMs and generally achieve a better performance [34]. Linux containers is one of the most popular implementations of this form of virtualization [34]. We refer to Linux containers as containers from now on. Containers use the following kernel features: *namespaces* and *cgroups* [34]. Namespaces provide isolation between containers so that one container has no visibility of objects outside it. Linux cgroups are used to limit the CPU and memory consumption of containers.

For completeness, we also discuss Docker [13], a popular open source platform for building, deploying and managing containers. Docker containers are generally used for deploying applications[1] instead of complete machines as described earlier. Docker provides a set of tools that allows the simplified use of container technology [55]. For instance, Docker packages the software application and its dependencies into a standalone package called an *image* [12]. Containers can then be started or created from Docker images. Docker also provides *registries* through which Docker images can be easily shared [55]. Finally, Docker simplifies application development by providing portability between different machines, i.e., the same container can be run on different machines while still exposing the same execution environment to the application. Docker is being widely adopted by the software development community and is being actively developed with contributions from the key IT players such as Amazon, Microsoft and Google [55].

[1] https://docs.docker.com/engine/faq/

2.2 Application development on the cloud

The emergence of Docker containers has changed the way applications and services are developed. Traditionally, software applications were written as single standalone modules that contain all the logic required to run the application [38]. However, such a *monolithic architecture* does not work as the application size and complexity grows [54]. For instance, problems arise when software bugs have to be traced in a large codebase. Furthermore, even small changes in the application requires the complete application to be rebooted. Finally, such an architecture cannot fully utilize the scaling benefits of the cloud as the whole copy of the application instance needs to be created even if only a single component of the application requires more CPU or memory.

Microservices

With the increasing adoption of Docker containers, a more cloud-native form of architecture known as the *microservice* architecture emerged. In this approach, applications are partitioned into smaller independent components (or microservices), each performing some business logic [38]. Although Docker containers were not created specifically for microservices, they present an ideal way for deploying independent modules easily with low cost [52]. The microservices themselves should be easy to understand and able to scale independently [38]. Communication between the microservices occur through network calls, typically using lightweight Representational State Transfer (REST) application programming interfaces (APIs). There are several advantages with this approach as compared to a monolithic application [54]. First, the microservices can be developed with different technology stacks if required. Second, the individual microservices can be scaled as required. This is especially useful when different services have different requirements; for instance, some services may be more CPU-intensive than others and thus, scaled out faster. Next, such an architecture reduces the time required to deploy new services or functionalities. For instance, a microservice can be deployed with few changes and even rolled back if required without affecting other services. Finally, the functionality provided by a microservice can be reused by other components for different purposes. The many benefits listed above meant that several organizations such as Netflix, SoundCloud and Amazon have successfully used this architecture to build large-scale fault-tolerant systems. However, some of the disadvantages of the microservice architecture arise from its distributed nature and reliance on the network for service calls [54]. This requires careful design of the microservices and a way to manage consistency between distributed components.

Function-as-a-Service

Function-as-a-Service (*FaaS*) is a relatively new extension of the microservice architecture. As briefly discussed earlier, with FaaS applications are decomposed into multiple independent *stateless functions*. FaaS is also known as *serverless computing* as all operational concerns of the underlying infrastructure are abstracted away from the developers [65]. Again, the common approach for deploying functions is to run them in containers. The serverless platform itself takes care of executing the function and scaling it when required. Thus, software developers can concentrate on business logic and do not have to manage scaling, function runtimes and lifecycles of virtualized instances. Such an approach significantly reduces the time taken to develop and deploy applications, which will result in more innovative services [65]. However, there are a few disadvantages with this approach. First, there can be performance issues as functions (containers) are not running all the time. This implies that a container may need to first start and then install application dependencies before being able to respond to a request. Again, the distributed nature of FaaS-based applications requires careful consideration of consistency and network calls. Furthermore, as the FaaS approach is in a nascent stage, carrying out end-to-end tests is difficult as tooling for development, management and deployment are limited [65]. Finally, using a public cloud offering for serverless functions implies that there are certain limitations on the duration of function execution and supported language runtimes. However, there are several open source serverless frameworks (Fission[2], Kubeless[3] and OpenFaas[4]) that provide more flexibility in implementation.

Container orchestrators

Finally, we discuss the role of *container orchestrators* in application development. The overall software application consisting of multiple Docker-based microservices (or stateless functions) needs to be reliable, highly available and scalable. Container orchestrators represent an automated mechanism to create, manage and deploy distributed applications. Kubernetes [30] is one of the most widely-used container orchestrators. It has radically helped to increase the speed at which applications can be deployed due its features of *immutability*, *declarative configuration* and *online self-healing* [40]. First, Kubernetes relies on the immutable nature of Docker images. As described earlier, Docker packages both the application and its dependencies into an image that can be used to deploy the application. This greatly simplifies deployment; for instance, if an error occurs, it is simply possible to roll back to the older image. Second, Kubernetes uses a declarative configuration object: the developer only has to specify the desired state of the system and Kubernetes ensures that the desired state is fulfilled. An example of such a state is that the application module needs three running replicas (or identical running instances). Such a declarative approach is easy to understand as there is no ambiguity in the specification of the desired state. Finally, Kubernetes takes all the necessary

[2] https://fission.io
[3] https://kubeless.io
[4] https://www.openfaas.com

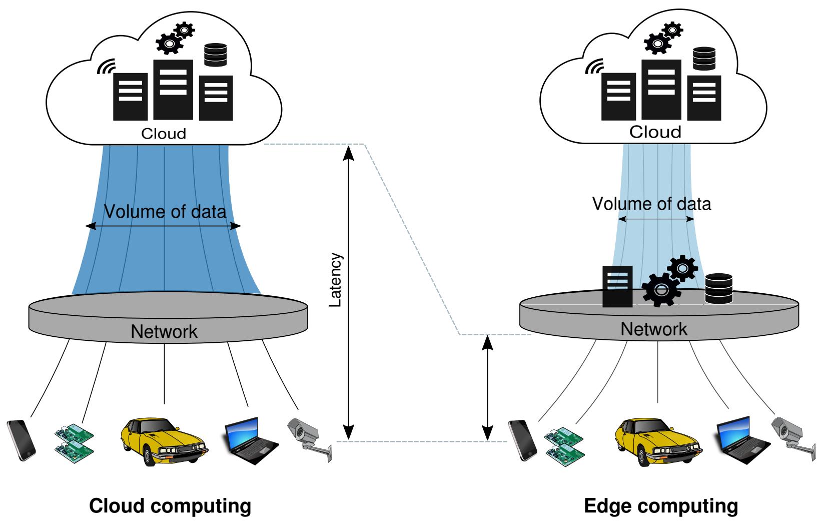


Fig. 2: The core concept of edge computing is to reduce latency of processing and volume of data sent to the cloud[5]

steps to maintain the desired state through online self-healing. This implies that it continuously monitors the state of the application and takes necessary corrective action. For instance, if a running instance of the application goes down, Kubernetes detects the change and brings up a new instance to maintain the desired state. Thus, Kubernetes abstracts away the operations tasks from software developers and is designed to give developers velocity, efficiency and agility [40].

2.3 Edge and fog computing

Edge [66] and *fog* [27] computing involve bringing cloud computing capabilities closer to the end devices. The main objective is to efficiently process the growing amount of data generated by Internet of Things (IoT) devices (described in Section 4) such as smart meters, connected cars, home and building automation systems. The number of such connections is expected to grow to 3.3 billion by 2021 [1]. Currently, the data from such devices are transferred to the cloud to obtain a meaningful analysis. Indeed, the elastic and on-demand nature of the cloud computing resources make it ideally suited for this purpose. However, cloud computing resources are usually available in large data centers located far away from the end devices generating the data. Thus, the large volume of data sent places immense stress on the backhaul links to the cloud. Moreover, there are several applications that require low-latency processing of

[5] Image adapted from http://www.ntt.co.jp/news2014/1401e/140123a.html

the data, such as vehicular safety applications [62], virtual/augmented reality applications [61] and real-time data analytics [67]. To address these issues, computing resources are made available at the edge of the network, i.e., closer to the end devices generating the data (Figure 2). This allows data to be processed with very low latency and removes the need for data to be sent to the distant cloud data centers. Moreover, such an approach addresses privacy concerns by processing the data at the edge and removing private information before sending to the cloud [67]. It is important to note that edge and fog computing are expected to co-exist with cloud computing. Some long-term forecasting and analytics can still be carried out on the cloud without having to send all data to it.

Edge computing and fog computing differ in the approach to bring computing resources closer to the user. Edge computing relies on co-locating resource-rich servers along with access points, i.e., one hop away from end devices [66, 62]. Multi-Access Edge Computing (previously known as Mobile-Edge Computing) (MEC) follows this approach wherein software applications and cloud computing capabilities are made available in the radio access network [4]. MEC servers are typically deployed at wireless base station sites. This approach is standardized by the European Telecommunications Standards Institute (ETSI). Similarly, our previous work [62] considers the deployment of edge devices for vehicular applications. This scenario assumes that access points (called roadside units) deployed along the road are augmented with computing resources. The cars send data (such as location, direction, speed) every second to the roadside units; this data is processed immediately at the edge and a response sent back to the car. Such processing includes predicting whether collisions occur or deciding when autonomous vehicles can change lanes. Thus, the latency of processing and sending the response is critical for such applications. We demonstrate that even with a small amount of computing resources at the edge, such an approach is able to meet the computational demands of vehicular applications in a city without having to send data to the cloud.

On the other hand, fog computing utilizes the computing resources of heterogeneous devices, including the end devices themselves. For instance, the authors in [26, 41] describe a fog platform comprising of a diverse set of devices, including edge routers, access points, set-top boxes and smartphones. The OpenFog Consortium, an alliance of industrial companies and universities, has recently standardized the fog computing architecture [14]. In this approach, the fog consists of multiple layers or tiers, each with different levels of compute and storage resources. For instance, in a fog-based vehicular application scenario, the cars themselves are fog computing nodes which can process data on-board. The cars (fog nodes) can connect to other cars within the same tier as well as to fog nodes in other tiers. This allows the devices within a layer to carry out processing in case connectivity to the higher tier goes down. Each tier provides additional processing, networking and storage capabilities than the tier below it. Thus, data from each tier is aggregated and sent up to the next layer. The next layer (above the cars) consists of roadside

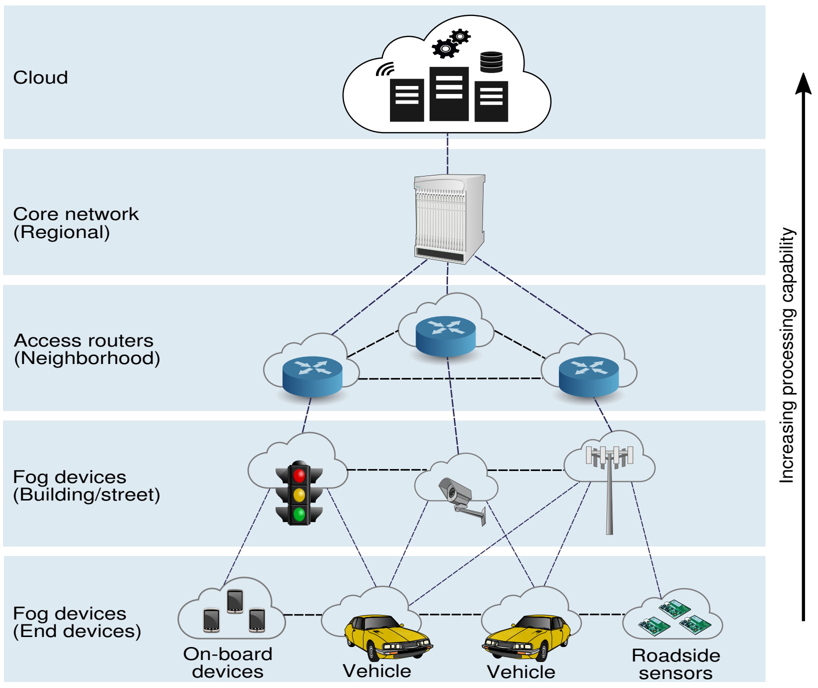

Cloud
Core network (Regional)
Access routers (Neighborhood)
Fog devices (Building/street)
Fog devices (End devices)
On-board devices
Vehicle
Vehicle
Roadside sensors
Increasing processing capability

Fig. 3: The multi-tier architecture of the fog [14].

devices such as roadside access points or traffic cameras. The layers above consist of neighborhood and regional fog tiers, each representing devices with increased capability. The key feature of this architecture is that interactions are possible between tiers as well as within the tier itself. Thus, the services should be available even if connectivity between the tiers is temporarily not available.

The approaches described above rely on virtualization to enable software applications to run seamlessly across different devices [47,66]. Furthermore, the cloud is still expected to play a role in managing the applications at scale. Together they can enable applications that require very low latency while still relying on the more resource-rich cloud for large-scale batch analytics.

3 Wireless connectivity

Today's wireless networks provide ubiquitous Internet connectivity accessible anywhere and at all times. The amount of Internet Protocol (IP) data handled by wireless networks has increased by a factor of 100 between 2010 to 2018 [20]. This section focuses on long-range wireless communications, i.e., cellular (or mobile) networks as well as wireless connectivity solutions specific to IoT devices.

3.1 Mobile networks

The demand for higher data rates and lower latencies are the driving forces behind the evolution of mobile (or cellular) networks. There has been a tremendous increase in data rates from second generation (2G) networks (the first to allow mobile data access) to the current fourth generation (4G) networks, with up to 100 Mbps supported today [74]. The adoption of mobile data was also driven by attractive flat-rate pricing schemes and the availability of new smartphone devices with high-resolution screens and better user interfaces [49]. The pervasive, always-on Internet connectivity has enabled users to access diverse mobile services whenever they want. Indeed, it is hard to imagine the success of taxi-sharing applications such as Uber without having a fast and reliable mobile Internet connection.

The advancement in mobile Internet connectivity was made possible due to improvements in the end-to-end network, comprising of the radio and the core network. On the radio side, 2G systems known as Global Systems for Mobile Communication (GSM) used a combined Frequency-Division Multiple Access (FDMA)/Time-Division Multiple Access (TDMA) system [44]. In such a system, the radio spectrum is divided into frequency sub-bands and within each sub-band, time is divided into frames and slots. This network supported mostly voice calls and very low data rate Internet. However, there was a growing demand for high data rates required by multimedia communication. To meet this requirement, 3G networks relied on a different technique known as Direct Sequence Wideband Code Division Multiple Access (DS-WCDMA) within TDMA [44], wherein time slots are available on multiple frequencies. This allowed for a growth in the capacity of mobile networks and a data rate of up to 14 Mbps. As the demand for data services over mobile networks increased, the next generation of networks, 4G Long Term Evolution (LTE), introduced the orthogonal frequency division multiplexing (OFDMA) technique [44]. In such a system, each mobile node is allocated time slots in one or more radio channel frequencies. The OFDMA technique ensures that interference between signals sent on different frequencies is minimal. Along with other innovations in the radio network, the capacity of such networks increased tremendously and the maximum data rate increased to 100 Mbps in the downlink and 50 Mbps in the uplink [44].

At the same time, the core network has evolved from circuit-switched networks to the current packet-switched Evolved Packet System (EPS). Packet switched networks were introduced in 2G networks and removed the need for dedicated end-to-end circuit-switched connections. The current 4G EPS networks are completely IP-based; this allows for fewer protocol conversions and thus a higher performance [15]. This architecture also separated the control plane and data plane elements. The control plane transports signaling messages (related to mobility and management), whereas the data plane is responsible for handling user data packets. The separation of these planes in the mobile core network allows the network operators to scale the data plane and control plane elements independently and better meet the demands of end users.

The next step in the evolution of mobile networks, i.e. the fifth generation (5G), is expected to support a growing amount of data from mobile and Internet of Things (IoT) devices with low latency communication. Specifically, 5G networks will incorporate technologies needed for low latency, energy-efficient and reliable communications from heterogeneous devices. In the radio network, modification of the radio frame structure, millimeter wave [20] and non-orthogonal multiple access (NOMA) [58] are among the key features proposed. NOMA achieves better spectral efficiency than before by allowing multiple users to share the same radio resources (frequency, timeslots and spreading code). Furthermore, the use of a different mmWave spectrum (30-300 GHz range) supports massive bandwidth and ultra-low latency applications such as virtual and augmented reality [20,58].

Another interesting development is the softwarization of the network through *Network Function Virtualization* (NFV) [33] and *Software Defined Networking* (SDN) [57]. Currently the mobile core network consists of proprietary hardware designed to meet the high performance requirements of such networks [60]. This implies that mobile network operators need to dimension their networks and plan for peak loads as upgrading of such infrastructure is expensive and slow. NFV utilizes the virtualization techniques described in Section 2.1 to enable the deployment of mobile network elements as software modules on general-purpose hardware [33]. This technique brings the benefits of cloud computing, namely scalability and reduced expenses, to the core network. The core elements can be scaled out or in depending on the actual demand. Furthermore, NFV aims to bring new innovative services to the mobile network as software-based deployments have a shorter deploy cycle than hardware-based implementations. Finally, as the elements are deployed as virtualized instances, they can be dynamically moved to the location most suitable for low-latency communication [71].

SDN involves the virtualization of networking itself [42] and is complementary to NFV. The main features of SDN are: the separation of the control and data plane, centralized network intelligence at a programmable controller, and standardized application programming interfaces (APIs) [57,42]. By separating the control plane from the data plane, the complexity of network devices such as switches are greatly simplified. The devices only need to receive instructions from the central SDN controller and forward data packets based on these instructions. The centralized controller allows network operators to have an overview of the entire network. Programs running on the SDN controller can make real-time changes to any part of the network. Finally, the standardized APIs abstract away the networking infrastructure from the applications. The APIs also enable the management of devices from multiple vendors.

SDN and NFV together enable the flexible management and programming of complex networks. These technologies make it easier for network operators to react to changes in the network. Furthermore, the time to deploy services is considerably reduced from hardware-based implementations.

5G networks are expected to support several new applications and services [18]. For instance, multiple person video communication is expected to become pervasive. This will enable collaboration at a scale not seen before. This will be further enhanced by the support for augmented and virtual reality. Sensor- and user-generated data can be used to replicate the movement and gestures of people from a physical setting to a virtual one [61]. Furthermore, applications that rely on tactile signals and haptic feedback will become reality with the low latency communication offered by 5G networks. In the core network, edge computing (described in Section 2.3) has been proposed to move computing closer to the end user and thereby meet the low latency requirements of such applications. This will allow remote control of machinery and robots, thereby enabling applications in remote health care as well.

3.2 Low Power Wide Area Networks

Low power wide area networks (LPWANs) are a class of networks specifically targeted for resource constrained and battery-powered IoT devices. Such networks offer low power, long range connectivity (in the range of kilometers) and support only low data rates. Thus, LPWANs are highly suited for smart city and machine-to-machine applications, including smart metering, smart grid and agricultural monitoring [64,22]. They support a class of applications and devices that cannot be otherwise served by existing wireless technologies. For instance, mobile networks (described in Section 3.1) are not energy efficient as they require much more complex processing on the end devices [64]. Moreover, this increased complexity would increase the cost of end devices. On the other hand, LPWANs promise a battery lifetime of ten years and a communication range of several kilometers [64]. It is important to note that these technologies are designed for a specific class of applications that require only low data rates (in the range of kilobits per second) and can tolerate some latency in communication [64]. This section describes the following LPWAN technologies: LoRa [68,45], narrowband-IoT [63] and Sigfox [9].

LoRa networks consist of two main components, LoRa and LoRaWAN. LoRa refers to the proprietary physical layer developed by Semtech [68], whereas LoRaWAN [45] corresponds to the medium access control (MAC) and network layers of the protocol stack. The physical layer achieves long distance communication by using the *chirp spread spectrum modulation* technique, wherein the signal is encoded into *chirp* pulses spread over a wide spectrum [64]. Chirp pulses can go from low to high frequencies (up-chirp) or vice-versa (down-chirp) over time. This modulation technique makes the signal robust to interference from other transmissions. This is very useful as LoRa operates in the unlicensed sub-GHz band. Each LoRa transmission can be configured with the following parameters: carrier frequency, bandwidth, coding rate, spreading factor and transmission power [32]. These parameters affect the communication range, the data rate and the occurrence of collisions. Thus, it is possible to assign the parameters in such a way to minimize collisions between transmissions. For instance, LoRaWAN specifies an Adaptive Data Rate (ADR) algorithm that dynamically manages the communication parameters to increase the capacity of the network and maximize the battery life of the end devices [69]. LoRaWAN also specifies the architecture of LoRa networks. The end devices or LoRa nodes communicate with *gateways* over the LoRa physical layer. Gateways simply relay the messages received from the nodes to a central *network server*. Nodes are not associated with a single gateway; this implies that gateways can receive and process messages from all nodes within its communication range. The network server manages the network and further sends the messages to the required *application server*. Such an architecture allows to increase the capacity of the network by increasing the number of gateways [64]. For instance, The Things Network [73] (TTN) is an open, community-driven LoRa network which allows the general public to place gateways of their own and thereby expand coverage. TTN itself provides the network server and the means to integrate applications to the network. Such a community-driven network allows end users to develop applications that can use the coverage provided by deployed gateways. Moreover, several mobile network operators (such as KPN, Orange, Swisscom, Softbank and others) have started deploying LoRaWAN networks to meet the growing demand for services that rely on such networks [53].

Narrowband-IoT (NB-IoT) is standardized by the 3GPP and is based on the LTE technology (described in Section 3.1). It operates in the licensed radio spectrum and thus will be available through telecom service providers. Furthermore, there are no duty cycle restrictions as the spectrum is licensed. In contrast to LoRa, NB-IoT uses the *narrowband modulation* technique, wherein the signal is encoded in a low bandwidth which also minimizes the noise level [64]. This technique also ensures that the spectrum is efficiently utilized by all the links. NB-IoT reuses several concepts from LTE including the frequency-division multiple access (FDMA). This implies that the end device needs to synchronize to the carrier frequency. Thus, the complexity of devices increases as compared to LoRa. However, this also implies that NB-IoT can ensure a higher quality of service (QoS) and lower communication latency than LoRa. At the time of writing there are 58 commercial networks[6] that use NB-IoT.

Sigfox is an LPWAN solution provider that operates its eponymous network based on a proprietary technology [9]. Sigfox networks use the *ultra narrowband modulation* technique, wherein the signal is encoded into a very narrow bandwidth (of less than 100 Hz) [64]. This reduces the amount of noise resulting in higher receiver sensitivity and thus, long distance communication. However, this is achieved at the expense of the data rate which is limited to 100 bps [64]. The devices communicate with proprietary Sigfox base stations using a random access MAC protocol and thus devices are not complex. Sigfox networks are deployed in partnership with other service providers (including

[6] https://www.gsma.com/iot/mobile-iot-commercial-launches/

telecommunication service providers) and 62 countries[7] are expected to be covered by the end of 2018.

Each LPWAN solution has its own modulation scheme and relies on a separate network architecture. However, the goals of each network is similar: to provide low-cost, long-range, low-power network connectivity to resource constrained devices. The value of connecting these devices to the network will be more apparent from the discussion in the next section.

4 Internet of Things

The Internet of Things (IoT) comprises of billions of Internet-connected devices equipped with sensors and actuators that are able to interact and cooperate with each other to achieve a common goal [21]. Indeed, we already see examples of the IoT today: RFID sensor-based tracking of shipments, routing of vehicles based on GPS data and real-time control of home devices through home assistants [76]. The rapid increase in the number of deployed devices is due to the availability of more efficient and low cost IoT devices, improved wireless connectivity (described in Section 3) as well as advances in the cloud (described in Section 2).

4.1 IoT devices

The devices forming the IoT can range from low-cost, low-complexity devices such as temperature/humidity sensors to full-fledged connected cars equipped with sophisticated sensors. This section highlights the main features of the IoT devices [39,17,19]. One of the most important features is that the device can *sense* its environment and collect measurements. Examples of these measurements include temperature, humidity, pressure, location, motion, light and sound. An on-board sensor can measure the data and convert it to a machine-friendly, digital representation. Certain devices are also equipped with *actuators*, i.e., components that produce a physical effect (such as motion or an electromechanical signal) in response to an input. A *processing unit* (such as a microcontroller, microprocessor, CPU, FPGA, etc.) handles several tasks, including processing sensed data and managing the remaining systems on the device. The capability of the processing unit depends on the requirements of the specific IoT application. Next, the device should be able to *communicate* to other devices or a network gateway or controller. Thus, the device typically contains a wireless transceiver for the chosen connectivity option. Again, here, the choice of connectivity standard depends on the requirements of the application. Finally, the device needs a *power source* to be able to function. Typically, the devices are battery-powered; but they can sometimes scavenge energy from other sources such as solar cells [19].

[7] https://www.sigfox.com/en/coverage/become-so

The growing availability of low-cost, low-power devices is leading to the pervasive deployment of IoT devices [16,39]. For instance, micro-electro-mechanical systems (MEMS) are an attractive method to package sensors and actuators as they can integrate these elements on a very small scale at low costs. Thus, sensors can be easily added to everyday objects. Moreover, the on-board processors are getting more powerful while simultaneously becoming smaller. Finally, the power consumption of such devices are further reduced when using IoT-specific connectivity options such as LoRa.

4.2 Connecting IoT devices

Next, we discuss how IoT devices communicate with each other or to the network. We have already discussed the available physical layer protocols in Section 3. Above this, there are several options for networking and transport protocols designed for IoT devices. At the network layer, IoT devices typically rely on the Internet Protocol (IP) and specifically the IPv6 protocol for networking [39]. However, there are several practical concerns to this approach especially for resource-constrained devices and unreliable wireless networks. For instance, some devices may not have sufficient energy resources (being battery-operated) to run the whole IP networking stack. Furthermore, some wireless networks such as LPWANs can have high latency and packet losses. In such scenarios, an *IoT gateway* acts as an intermediary on the communication path between the device and the application [39,72]. These gateways translate non-IP packets from the devices to IP-based packets that can be sent to the application server. To this end, the Internet Engineering Task Force (IETF), an Internet standards organization, has developed standardized protocols to incorporate the resource-constrained non-IP devices into an IP-based network. The interested reader can refer to the IETF working groups[8] (*6lo, 6tisch, lpwan, ipwave*) for the specifics of the protocols for IoT networks.

At the transport layer, IoT networks may use either the Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) [39]. The choice of the protocol depends on both the upper and lower layers of the networking stack as well as the capability of the devices. TCP ensures reliable communication through a connection-oriented scheme (a session is established between the sender and receiver) and several error correction mechanisms including retransmission of packets. On the other hand, UDP is a connectionless protocol wherein data packets are sent between sender and receiver without error control. Next, at the application layer, there are several options, including generic web-based HTTP protocols, existing messaging protocols (XMPP) and more lightweight IoT-specific protocols such as CoAP and MQTT. CoAP and MQTT aim to support messaging for resource-constrained devices that may operate in networks with packet loss and low bandwidth [17]. A detailed description of application protocols is available in [17,39]. Again, the choice of

[8] https://datatracker.ietf.org/wg/

protocol relies on the requirements of the specific application and capability of the devices.

4.3 Inter-operating networks

The growing availability of smart, connected IoT devices has resulted in their rapid adoption in several areas, such as smart homes, industrial settings and transportation. The discussion above highlights the variety of communication protocols, which are decided based on the device capability and application requirements. Thus, the IoT solutions usually operate in their own vertical silos with specific communication protocols for the domain they operate in [25, 29]. Although such IoT solutions bring tremendous benefits within the particular area of operation, they do not meet the original vision of smart devices being able to cooperate with each other to meet a common goal. For instance, consider a scenario wherein IoT devices are deployed at bus stops across a city. These devices can detect the number of people currently waiting for a bus. This data can be used by the public transport provider to send more buses when there is a surge in demand. A truly IoT solution would also allow this data to be accessed by different services, such as a registered car pooling service that can also serve this demand. Such a scenario requires that devices can be seamlessly *discovered* and their sensed data *accessed* across different domains and architectures [72].

There are several approaches for the discovery of IoT devices. They can categorized into three main categories [10, 28]. One class relies on small- or medium-range wireless communication for discovering devices nearby. For instance, Google and Apple both use Bluetooth Low Energy for their UriBeacon (now part of the Eddystone project) and iBeacon projects that enable devices to discover and interact with each other. A second category of discovery relies on searching for device endpoints in a network. For instance, *multicast DNS*[9] (mDNS) is a distributed version of DNS service discovery (DNS-SD) that translates the IoT service/host name to an IP address within small networks. Finally, a third category relies on querying of centralized directories and thus scales to larger networks. The CoRE Resource Directory[10] is an example of such an approach. This approach allows services from other domains and networks to discover resources based on attributes; for instance, all devices matching a certain criteria (type or interface) can be listed by querying this directory [72].

Another important aspect of interoperability is that the data should be sent in a standardized format so that different applications and services can extract useful information from it [17]. To this end, the IPSO Alliance [2], a global body of multiple IoT companies, aims to enable IoT device interoperability by specifying open standards for semantics, security, device identity

[9] https://www.ietf.org/rfc/rfc6762.txt
[10] https://core-wg.github.io/rd-dns-sd/#resource-directories

and other protocols. They have specified a data model known as Smart Objects [3] to describe IoT device resources. For instance, a temperature sensor could be represented as `3300/0/5700` where 3300 represents that it's a temperature sensor, 0 represents the 0th instance of the sensor and 5700 refers to the most recent reading. This abstraction allows the software application to use simple APIs to access and read the IoT device resources. The model is designed to work on top of any REST-based protocol. The Web of Things Thing Description, a standard developed by the World Wide Web Consortium (W3C), also describes a formal model for representing IoT devices [11]. In this model, device resources are typically represented in JSON-LD format and different application layer protocols such as MQTT, CoAP and HTTP are supported. A listing of other data models is available at [10]. Another approach relies on using a separate entity, i.e., a *data broker*, between the devices and the application server [39]. The broker converts the data from multiple devices to a common format that can be accessed by authorized applications. Such an approach is suitable for non-IP based devices as well and where an application protocol is not used.

The approaches highlighted above are a step towards achieving the goal of inter-operable IoT networks that will further enable new and innovative services.

5 Towards a sharing economy

The increasing maturity of cloud-based services, wireless connectivity solutions and IoT devices creates a growing opportunity for data-driven solutions for the sharing economy. Such solutions can have a societal impact through the collaborative use of data from multiple providers, including individuals and public or private organizations [37]. To this end, a cloud-based IoT platform represents an ideal way to aggregate data from multiple sources and expose this information to different service providers. There are several commercial and open source IoT platforms available today [17]. OpenMTC [7] is a prominent example of an open source platform built to enable shared data access across several application domains. We briefly describe the key features of this platform and how it enables sharing economy solutions.

5.1 OpenMTC

OpenMTC is an open source implementation of the standardized IoT platform architecture by oneM2M[11]. The goal of such a platform is to provide a horizontal layer for IoT devices from different domains (such as healthcare,

[11] oneM2M (http://www.onem2m.org) is a global standards initiative comprising of eight regional ICT standards organizations and over 200 companies.

transport and utilities) to communicate with the application layer. The platform was released in 2012 and made open source in 2017. The architecture follows a hierarchy of three layers, described next.

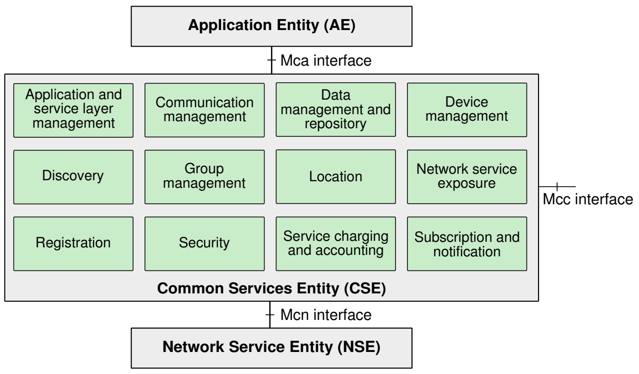

Application Entity (AE)
Mca interface
Application and service layer management
Communication management
Data management and repository
Device management
Discovery
Group management
Location
Network service exposure
Mcc interface
Registration
Security
Service charging and accounting
Subscription and notification
Common Services Entity (CSE)
Mcn interface
Network Service Entity (NSE)

Fig. 4: oneM2M entities.

– The *application layer* consists of *application entities* that implement the service logic. Examples of such entities include applications to monitor vehicle fleets or to track power consumption.
– The *common services layer* provides the core functionality of the platform including data and device management, service subscription management, discovery and others (illustrated in Figure 4). The functionality of each entity is detailed in [5].
– The *network services layer* provides transport and connectivity services to the devices connected to the platform.

The interfaces between the different layers (Mca, Mcn and Mcc) are also standardized. Thus, the overall architecture is designed to be protocol-agnostic and thereby support devices from different domains. A node in the OpenMTC platform can implement one or more of the entities from the different layers (described above) and expose standardized interfaces to other nodes in the network. The *gateway* and *backend* are the main components of the OpenMTC platform. The gateway interconnects devices from different domains. *Protocol adapters* are used to provide inter-connectivity with other IoT platforms. Furthermore, the platform provides a *software development kit* (SDK) to allow application developers to write applications compliant with oneM2M [8]. The source code of all components are available at [8] along with the respective Docker images.

5.2 Case study: Car sharing

A car sharing service is ideally suited for urban users who prefer not to own a car, but still have the convenience of using a car when required. When the user needs a car, he/she simply goes to the nearest available shared car, unlocks it and drives it to the preferred destination. The user then parks the car in a designated spot and has to pay only for the duration of the trip. The success of such a service depends on the seamless use of the cars with minimal user intervention. This can be achieved by leveraging data from IoT devices and other service providers (such as insurance companies). The following discussion is based on the oneM2M specification [6], which describes how IoT platforms (such as OpenMTC) can enable car sharing applications.

Cars today are equipped with a variety of sensors, including door control sensors, tire pressure sensors, fuel level sensors and GPS. The rich set of data from these sensors can be used to automate and simplify the process of utilizing shared cars for the end user. Sensor data is communicated to a cloud-based *IoT platform* via a *smartphone* which acts as the gateway. Other devices such as access points deployed along the road [62] can also behave as gateways. However, it is feasible to offer the car sharing service as a smartphone application and thus, the phone can behave as a gateway. Furthermore, the smartphone itself has additional sensors that can provide useful data, for instance, for navigation. The IoT platform collects the status and configuration information from the vehicles as well as the *service providers*. Service providers include the car sharing provider itself, insurance companies and gas stations. Providers are assumed to have an agreement to provide a unified service.

Figure 5 illustrates how the data from multiple sources can be exchanged for a car sharing service takes place.

1. First, the service provider applications register to the IoT platform. Each application subscribes to the specific information it requires. Such a subscription allows the service provider application to monitor updates or changes to the subscribed information. Examples of such information include location, health and fuel status of the car. This step also requires the IoT platform to ensure that only authorized applications are granted access to the information.
2. When a user intends to use a shared car, he/she obtains the location of the nearest available car from the smartphone application.
3. Next, he/she proceeds to the nearest car (pointed to by the application), opens the car door and starts the car through the application. The smartphone can interact with the car through Bluetooth or NFC. The application ensures that the user has a valid subscription and is authorized to use the car. The car's on-board sensors report that the car is in use to the IoT platform via the smartphone (or gateway).
4. The IoT platform communicates the status of the car as occupied to the backend of the car sharing application. This status update can be used to trigger an update to the front-end smartphone application or website to mark

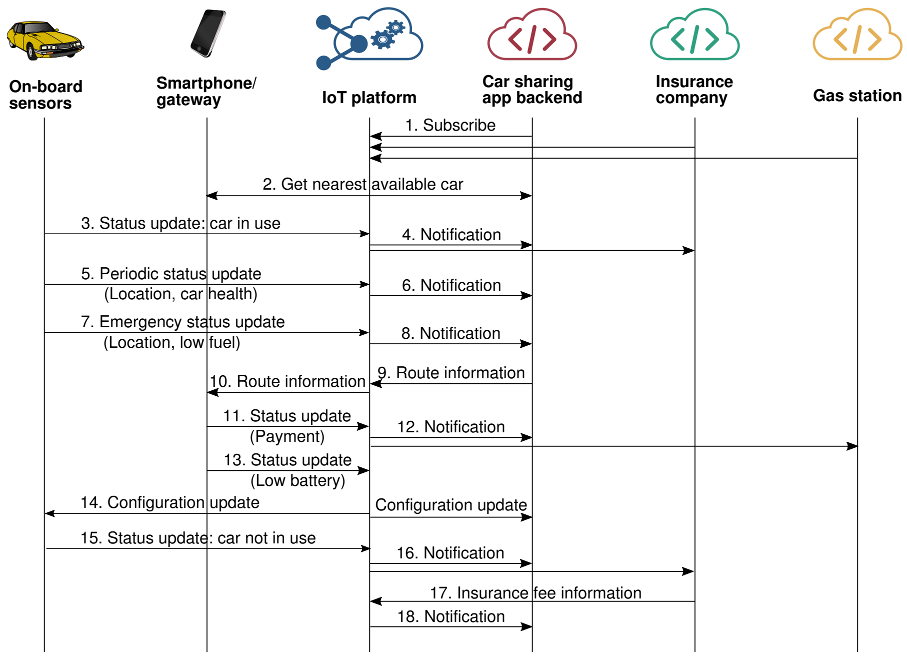


Fig. 5: Information flow for a car sharing application.

the car as unavailable to other users. The insurance provider is also informed of the status of the car.

5-6. The car periodically reports its status – including location, fuel status and health – to the IoT platform. The IoT platform reports the information to the applications subscribed to the particular data.

7-8. In case of an emergency, for instance, if the fuel is low, the car sends an update with an "urgent" status. This ensures that the IoT platform immediately notifies the car sharing backend application.

9-10. The backend application computes the location of the nearest gas station (with which it has a service agreement) based on the most recent location of the car. The application then communicates the route to the nearest gas station to the IoT platform, which relays this to the smartphone.

10. At the gas station, the user can make the payment through the smartphone's NFC interface. The car sharing application also sends a status update to the IoT platform.

11-12. The IoT platform communicates the payment details to both the car sharing backend and the gas station provider. This step could also allow the car sharing provider to directly make the payment to the gas station without having the user to pay.

13. Next, the smartphone reports its low battery status to the IoT platform.

14. The IoT platform changes the configuration of the status updates to minimize battery usage. For example, the platform can increase the time interval between status updates or configure only urgent notifications to be sent. The platform then informs both the smartphone and the car sharing backend application of the new configuration.

15-16. When the user arrives at the destination, the user stops the car and turns off the ignition. All backlog data (if the configuration was changed) is reported to the IoT platform along with the notification that the car is no longer in use. The platform, in turn, notifies the backend application and the insurance provider.

17-18. The insurance provider application sends a message containing the insurance fee information to the IoT platform, which is then communicated to the car sharing application backend.

6 Conclusion

This chapter reviewed the major developments in cloud computing, wireless connectivity and IoT in the context of collaborative consumption. First, cloud computing has significantly lowered the economic barrier for deploying software applications. The software development process is further simplified by the growing popularity of Docker-based microservices and stateless functions. This in turn results in faster time to market for new services. Next, improvements in wireless connectivity have resulted in the ubiquitous availability of Internet connectivity. Low power wireless solutions specific to IoT devices have also emerged in the past few years. This allows a large number of low-cost IoT devices to connect to the network and share their sensed data. Thus, new sharing economy solutions can utilize the large volume of data from a diverse set of IoT devices to provide services in an automated and collaborative manner. Furthermore, edge and fog computing are expected to support a new class of applications that require processing of data with a very low latency.

References

1. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2016–2021 White Paper. `http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.pdf`. (Accessed on 16/07/2018)
2. IPSO Smart Objects. `https://www.omaspecworks.org/develop-with-oma-specworks/ipso-smart-objects/`. Online; accessed 23.07.2018
3. IPSO Smart Objects. `https://github.com/IPSO-Alliance/pub`. Online; accessed 23.07.2018
4. Mobile-Edge Computing (MEC); Service Scenarios. `https://www.etsi.org/technologies-clusters/technologies/multi-access-edge-computing`. (Accessed on 24/07/2018)
5. oneM2M Functional Architecture, ETSI Standard TS-0001-V3.11.0. `http://www.onem2m.org/technical/published-drafts`. Online; accessed 07.08.2018
6. oneM2M Vehicular Domain Enablement, Draft Technical Report TR-0026-V4.1.0. `http://www.onem2m.org/technical/published-drafts`. Online; accessed 07.08.2018

7. OpenMTC. http://www.open-mtc.org/index.html. Online; accessed 07.08.2018
8. OpenMTC. https://github.com/OpenMTC/OpenMTC. Online; accessed 08.08.2018
9. Sigfox. https://www.sigfox.com/en. Online; Accessed on 18.07.2018
10. Web of Things – Technology Landscape. http://w3c.github.io/wot/landscape.html. Online; accessed 23.07.2018
11. Web of Things (WoT) Thing Description. https://www.w3.org/TR/wot-thing-description/. Online; accessed 23.07.2018
12. What is a container? https://www.docker.com/what-container. (Accessed on 16/07/2018)
13. What is Docker? https://www.docker.com/what-docker. (Accessed on 16/07/2018)
14. Ieee approved draft standard for adoption of openfog reference architecture for fog computing. IEEE P1934/D2.0, April 2018 pp. 1–175 (2018)
15. 3GPP: The Evolved Packet Core. http://www.3gpp.org/technologies/keywords-acronyms/100-the-evolved-packet-core. Online; Accessed on 04.07.2018
16. Akyildiz, I.F., Su, W., Sankarasubramaniam, Y., Cayirci, E.: Wireless sensor networks: a survey. Computer networks **38**(4), 393–422 (2002)
17. Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., Ayyash, M.: Internet of things: A survey on enabling technologies, protocols, and applications. IEEE Communications Surveys & Tutorials **17**(4), 2347–2376 (2015)
18. Alliance, N.: 5g white paper. Next generation mobile networks, white paper pp. 1–125 (2015)
19. Anastasi, G., Conti, M., Di Francesco, M., Passarella, A.: Energy conservation in wireless sensor networks: A survey. Ad hoc networks **7**(3), 537–568 (2009)
20. Andrews, J.G., Buzzi, S., Choi, W., Hanly, S.V., Lozano, A., Soong, A.C., Zhang, J.C.: What will 5G be? IEEE Journal on selected areas in communications **32**(6), 1065–1082 (2014)
21. Atzori, L., Iera, A., Morabito, G.: The internet of things: A survey. Computer networks **54**(15), 2787–2805 (2010)
22. Augustin, A., Yi, J., Clausen, T., Townsley, W.M.: A study of LoRa: Long range & low power networks for the internet of things. Sensors **16**(9), 1466 (2016)
23. Baldini, I., Castro, P., Chang, K., Cheng, P., Fink, S., Ishakian, V., Mitchell, N., Muthusamy, V., Rabbah, R., Slominski, A., et al.: Serverless computing: Current trends and open problems. In: Research Advances in Cloud Computing, pp. 1–20. Springer (2017)
24. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: ACM SIGOPS operating systems review, vol. 37, pp. 164–177. ACM (2003)
25. Bello, O., Zeadally, S., Badra, M.: Network layer inter-operation of device-to-device communication technologies in internet of things (iot). Ad Hoc Networks **57**, 52–62 (2017)
26. Bonomi, F., Milito, R., Natarajan, P., Zhu, J.: Fog computing: A platform for Internet of Things and analytics. In: Big Data and Internet of Things: A Roadmap for Smart Environments, pp. 169–186. Springer (2014)
27. Bonomi, F., Milito, R., Zhu, J., Addepalli, S.: Fog computing and its role in the Internet of Things. In: Proceedings of the first edition of the MCC workshop on Mobile cloud computing, pp. 13–16. ACM (2012)
28. Bröring, A., Datta, S.K., Bonnet, C.: A categorization of discovery technologies for the internet of things. In: Proceedings of the 6th International Conference on the Internet of Things, pp. 131–139. ACM (2016)
29. Bröring, A., Schmid, S., Schindhelm, C.K., Khelil, A., Kabisch, S., Kramer, D., Le Phuoc, D., Mitic, J., Anicic, D., Teniente López, E.: Enabling iot ecosystems through platform interoperability. IEEE software **34**(1), 54–61 (2017)
30. Burns, B., Grant, B., Oppenheimer, D., Brewer, E., Wilkes, J.: Borg, Omega, and Kubernetes. Queue **14**(1), 10 (2016)
31. Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I.: Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation computer systems **25**(6), 599–616 (2009)

32. Croce, D., Gucciardo, M., Mangione, S., Santaromita, G., Tinnirello, I.: Impact of lora imperfect orthogonality: Analysis of link-level performance. IEEE Communications Letters **22**(4), 796–799 (2018)
33. ETSI: Network Functions Virtualisation - An Introduction, Benefits, Enablers, Challenges, Call for Action. Tech. rep. (2012)
34. Felter, W., Ferreira, A., Rajamony, R., Rubio, J.: An updated performance comparison of virtual machines and linux containers. In: Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium On, pp. 171–172. IEEE (2015)
35. Fischer, J.E., Colley, J.A., Luger, E., Golembewski, M., Costanza, E., Ramchurn, S.D., Viller, S., Oakley, I., Froehlich, J.E.: New horizons for the iot in everyday life: proactive, shared, sustainable. In: Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct, pp. 657–660. ACM (2016)
36. Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I.: Above the clouds: A Berkeley view of cloud computing. Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS **28**(13), 2009 (2009)
37. Garcí, J.M., Fernández, P., Ruiz-Cortés, A., Dustdar, S., Toro, M.: Edge and cloud pricing for the sharing economy. IEEE Internet Computing **21**(2), 78–84 (2017). DOI 10.1109/MIC.2017.24
38. Garriga, M.: Towards a taxonomy of microservices architectures. In: A. Cerone, M. Roveri (eds.) Software Engineering and Formal Methods, pp. 203–218. Springer International Publishing, Cham (2018)
39. Hanes, D., Salgueiro, G., Grossetete, P., Barton, R., Henry, J.: IoT Fundamentals: Networking Technologies, Protocols, and Use Cases for the Internet of Things. Cisco Press (2017)
40. Hightower, K., Burns, B., Beda, J.: Kubernetes: Up and Running: Dive Into the Future of Infrastructure. " O'Reilly Media, Inc." (2017)
41. Hong, K., Lillethun, D., Ramachandran, U., Ottenwälder, B., Koldehofe, B.: Mobile fog: A programming model for large-scale applications on the internet of things. In: Proceedings of the second ACM SIGCOMM workshop on Mobile cloud computing, pp. 15–20. ACM (2013)
42. Jain, R., Paul, S.: Network virtualization and software defined networking for cloud computing: a survey. IEEE Communications Magazine **51**(11), 24–31 (2013)
43. Kivity, A., Kamay, Y., Laor, D., Lublin, U., Liguori, A.: kvm: the Linux Virtual Machine Monitor. In: Proceedings of the Linux symposium, vol. 1, pp. 225–230. Dttawa, Dntorio, Canada (2007)
44. Kurose, J.F., Ross, K.W.: Computer networking: a top-down approach: international edition. Pearson Higher Ed (2013)
45. LoRa Alliance: LoRaWAN Specification (V1.0.3). `https://www.lora-alliance.org/resource-hub/lorawantm-specification-v103` (2018). Online; Accessed on 18.07.2018
46. Malmborg, L., Light, A., Fitzpatrick, G., Bellotti, V., Brereton, M.: Designing for sharing in local communities. In: Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems, pp. 2357–2360. ACM (2015)
47. Marín-Tordera, E., Masip-Bruin, X., García-Almiñana, J., Jukan, A., Ren, G.J., Zhu, J.: Do we all really know what a fog node is? current trends towards an open definition. Computer Communications **109**, 117–130 (2017)
48. Marston, S., Li, Z., Bandyopadhyay, S., Zhang, J., Ghalsasi, A.: Cloud computing—the business perspective. Decision support systems **51**(1), 176–189 (2011)
49. Mcqueen, D.: The momentum behind lte adoption [sgpp lte]. IEEE Communications Magazine **47**(2), 44–45 (2009)
50. Mell, P., Grance, T., et al.: The nist definition of cloud computing (2011)
51. Muller, A., Wilson, S.: Virtualization with VMware ESX server (2005)
52. Nadareishvili, I., Mitra, R., McLarty, M., Amundsen, M.: Microservice architecture: aligning principles, practices, and culture. " O'Reilly Media, Inc." (2016)
53. Navarro-Ortiz, J., Sendra, S., Ameigeiras, P., Lopez-Soler, J.M.: Integration of LoRaWAN and 4G/5G for the Industrial Internet of Things. IEEE Communications Magazine **56**(2), 60–67 (2018)
54. Newman, S.: Building microservices: designing fine-grained systems. " O'Reilly Media, Inc." (2015)

55. Nickoloff, J.: Docker in Action, 1st edn. Manning Publications Co., Greenwich, CT, USA (2016)
56. Nider, J.: A comparison of virtualization technologies for use in cloud data centers. IBM Research Report H-0330 (HAI1801-001) (2018)
57. ONF: Software-defined networking: The new norm for networks. ONF White Paper (2012)
58. Parvez, I., Rahmati, A., Guvenc, I., Sarwat, A.I., Dai, H.: A survey on low latency towards 5g: Ran, core network and caching solutions. IEEE Communications Surveys & Tutorials (2018)
59. Popek, G.J., Goldberg, R.P.: Formal requirements for virtualizable third generation architectures. Communications of the ACM **17**(7), 412–421 (1974)
60. Premsankar, G., Ahokas, K., Luukkainen, S.: Design and implementation of a distributed mobility management entity on openstack. In: Cloud Computing Technology and Science (CloudCom), 2015 IEEE 7th International Conference on, pp. 487–490. IEEE (2015)
61. Premsankar, G., Di Francesco, M., Taleb, T.: Edge computing for the Internet of Things: a case study. IEEE Internet of Things Journal **5**(2), 1275–1284 (2018)
62. Premsankar, G., Ghaddar, B., Di Francesco, M., Verago, R.: Efficient placement of edge computing devices for vehicular applications in smart cities. In: NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium. IEEE (2018)
63. Ratasuk, R., Mangalvedhe, N., Zhang, Y., Robert, M., Koskinen, J.P.: Overview of narrowband iot in lte rel-13. In: Standards for Communications and Networking (CSCN), 2016 IEEE Conference on, pp. 1–7. IEEE (2016)
64. Raza, U., Kulkarni, P., Sooriyabandara, M.: Low power wide area networks: An overview. IEEE Communications Surveys & Tutorials (2017)
65. Roberts, M., Chapin, J.: What is Serverless?: understanding the latest advances in cloud and service-based architecture. " O'Reilly Media, Inc." (2017)
66. Satyanarayanan, M., Bahl, P., Caceres, R., Davies, N.: The case for VM-based cloudlets in mobile computing. IEEE Pervasive Computing **8**(4), 14–23 (2009)
67. Satyanarayanan, M., Simoens, P., Xiao, Y., Pillai, P., Chen, Z., Ha, K., Hu, W., Amos, B.: Edge analytics in the Internet of Things. IEEE Pervasive Computing **14**(2), 24–31 (2015)
68. Semtech: What is LoRa? `https://www.semtech.com/technology/lora/what-is-lora`. Online; Accessed on 18.07.2018
69. Slabicki, M., Premsankar, G., Di Francesco, M.: Adaptive configuration of lora networks for dense iot deployments. In: 16th IEEE/IFIP Network Operations and Management Symposium (NOMS 2018), pp. 1–9 (2018)
70. Strauss, D.: Containers–not virtual machines–are the future cloud. The Linux Journal **228**, 118–123 (2013)
71. Taleb, T.: Toward carrier cloud: Potential, challenges, and solutions. Wireless Communications, IEEE **21**(3), 80–91 (2014)
72. Tanganelli, G., Vallati, C., Mingozzi, E.: Edge-centric distributed discovery and access in the internet of things. IEEE Internet of Things Journal **5**(1), 425–438 (2018)
73. The Things Network: The Thing Network Mission. `https://github.com/TheThingsNetwork/Manifest/blob/master/Mission.md` (2015). Online; accessed 18.07.2018
74. Wang, C.X., Haider, F., Gao, X., You, X.H., Yang, Y., Yuan, D., Aggoune, H., Haas, H., Fletcher, S., Hepsaydir, E.: Cellular architecture and key technologies for 5G wireless communication networks. IEEE Communications Magazine **52**(2), 122–130 (2014)
75. Watson, J.: Virtualbox: bits and bytes masquerading as machines. Linux Journal **2008**(166), 1 (2008)
76. Woetzel, J., Remes, J., Boland, B., Lv, K., Sinha, S., Strube, G., Means, J., Law, J., Cadena, A., von der Tann, V.: Smart cities: digital solutions for a more livable future. McKinsey Global Institute San Francisco (2018)