Received Oct 15,2023, accepted Nov 10, 2023, date of publication Nov 11, 2023

Digital Object Identifier 10.46470/03d8ffbd.d488f834

Wireless Open Source Supremacy: Osmocom-based Networks for Indoor and Outdoor IoT Applications Deployment

SADIQ IQBAL[1], JEHAD M. HAMAMREH [2]

[1]WISLAB, Department of Electrical and Computer Engineering, Antalya Bilim University, Antalya, Turkey (e-mail:sadiq.zahid73@gmail.com)
[2]WISLAB, Department of Electrical and Electronic Engineering, Antalya Bilim University, Antalya, Turkey (e-mail:jehad.hamamreh@antalya.edu.tr)

Corresponding author: Sadiq Iqbal (Web: www.wislabi.com)

WISLAB (wislabi.com/solutions) offers solutions for building and deploying fully secure, cloud-based, and low-cost end-to-end 4G/5G networks along with providing consultations on helping companies reduce their networks CAPEX/OPEX cost and determine which solutions are best suited for their needs and use cases.

ABSTRACT The telecommunications landscape has undergone a remarkable transformation over the past three decades. From the early days of voice-only communication to today's era of high-speed data, video calls, and the Internet of Things (IoT), the journey has been nothing short of extraordinary. As we stand in the year 2023, it's essential to not only reflect on the past but also look ahead to what the future holds for telecommunications. As we move forward, the future of telecommunications holds exciting possibilities. The deployment of Fifth Generation (5G) networks promises unparalleled speed, low latency, and the ability to connect billions of devices simultaneously. The rise of AI, edge computing, network slicing, and IoT integration is reshaping industries, from healthcare and transportation to agriculture and smart cities. To address the unique requirements of IoT, 5G alone is not enough, and much more agile networks like 2G is required. Thus, it becomes imperative to look towards a new horizon with benefits that complement the limitations of IoT. So far, there are multiple directions to choose from to deploy IoT devices. However, in this work we focus on the use of the second generation of wireless technology (2G) for IoT applications, with more emphasis on the role of open source based architectures that can entirely shift the paradigm of connectivity as discussed in [1]. More specifically, we explore the 2G-based open-source software called Osmocom and its potential to build IoT systems and applications. First, we talk about 1G because it's important to know the start of the wireless trend; second, we move towards the actual 2G and introduce the IoT technology. This will provide the background knowledge that will help in understanding the concept of using Open Source wireless technology in deploying IoT applications. The paper also highlights the crucial role of Osmocom in enabling the transformation and convergence of these technologies to shape the future of telecommunications and IoT.

INDEX TERMS Osmocom, Open source 2G networks, IoT connectivity, Indoor IoT applications, Outdoor IoT applications, Wireless communication, Mobile networks, Cellular technology, Network infrastructure, IoT deployment, IoT solutions, IoT connectivity solutions, IoT network architecture, 2G technology, Osmocom framework, Wireless IoT connectivity, Network optimization, IoT deployment strategies, IoT network planning, Osmocom use cases.

I. INTRODUCTION

The telecommunications landscape has undergone a remarkable transformation over the years. From the early days of voice-only communication to today's era of high-speed data, video calls, and the IoT, the journey has been nothing short of extraordinary. As we stand in the year 2023, it's essential to not only reflect on the past but also look ahead to what the future holds for telecommunications.

A. THE FUTURE IS ALREADY HERE

As we move forward, the future of telecommunications holds exciting possibilities. The deployment of 5G networks promises unparalleled speed, low latency, and the ability to connect billions of devices simultaneously. The rise of edge computing, network slicing, and IoT integration is reshaping industries, from healthcare and transportation to agriculture and smart cities. However, to address the unique requirements of IoT, 5G alone is not enough although it can provide unlimited connections among IoT devices but the real restrictions occur due to the limited internal infrastructure of IoT devices.

Thus, it becomes imperative to look towards a new horizon with benefits that complement the limitations of IoT. So far, there are multiple directions to choose from to deploy IoT devices. However, we focus on Second-generation (2G) wireless technology, but with open source access that entirely shifts the paradigm as discussed in [1]. We will focus on First Generation (1G), 2G, Open-source, IoT, and Osmocom by discussing them briefly for the readers in this section. Firstly, we will talk about 1G because it's important to know the start of the wireless trend, secondly, we will move towards the actual 2G and then introduce the IoT. This will provide the background knowledge that will help the students in understanding the concept of using Open Source wireless technology in deploying IoT applications.

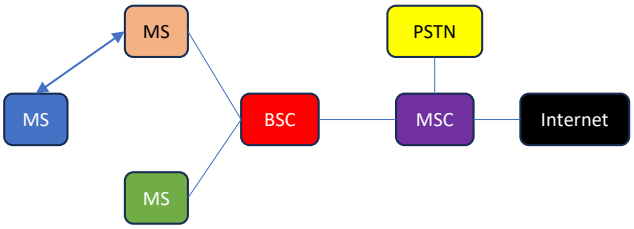

FIGURE 1. Architecture of AMPS.

B. 1G: THE GENESIS OF MOBILE COMMUNICATION

The advent of the 1G wireless networks marked a revolutionary moment in the history of communication technology. These networks, which emerged in the late 1970s and early 1980s, laid the foundation for the mobile communication era we know of today that were succeeded by 2G. The fundamental distinction between these two generations of mobile communication lies in their audio transmissions, with 1G networks employing analog signals and 2G networks exclusively utilizing digital signals. Various 1G cellular standards were developed and implemented across different countries. However, on a global scale the Nordic Mobile Telephone (NMT) and Advanced Mobile Phone System (AMPS) systems gained the most widespread acceptance. The architecture of an APMS system is shown in Fig. 1. The intrinsic advantages of digital technology over analog technology eventually led to the complete replacement of 1G networks by 2G networks. In developed economies, many 1G networks were deactivated by the year 2000. Nevertheless, in some regions, these networks continued to function into the 2010s [2].

1G networks operate on the principle of Frequency Division Multiple Access (FDMA). FDMA partitions the frequency band into numerous channels, with each channel being dedicated to a particular user for the duration of their call. This approach designates a unique frequency band for each user, preventing any overlap and interference between users.

1G networks lacked a standardized network protocol or data transfer rate, primarily focusing on voice communication. The pinnacle speeds and frequencies attainable in this generation were 2.4 kbps and 150 MHz, respectively [3].

Constraints of 1G Phones:

- Voice calls were prone to disruptions and unauthorized listening.
- Call quality could be adversely affected by weather conditions.
- These networks supported only a restricted number of users simultaneously.
- 1G networks had limited coverage distances.
- Mobile phones had poor battery endurance.
- The devices were large and inconvenient to carry.
- Roaming between similar systems was not feasible.

1) 1.5G Network

1.5G mobile networks served as an interim evolution between 1G and 2G networks. This network technology emerged in the late 1980s and early 1990s with the primary aim of addressing the limitations of First Generation networks, particularly in terms of voice and data services. This transitional technology harnessed a blend of analog and digital components to deliver superior voice quality and enhanced data transfer capabilities that surpassed the capabilities of 1G networks. Additionally, 1.5G networks introduced innovations like time-division multiple access (TDMA) and code-division multiple access (CDMA) technologies. These advancements contributed to more efficient utilization of the available radio spectrum, consequently boosting network capacity.

A notable feature of 1.5G networks was the adoption of packet-switched data transmission. This approach marked a departure from the circuit-switched transmission utilized by 1G networks, resulting in faster data transfer rates and the optimized utilization of network resources. These enhancements laid the foundation for the development of

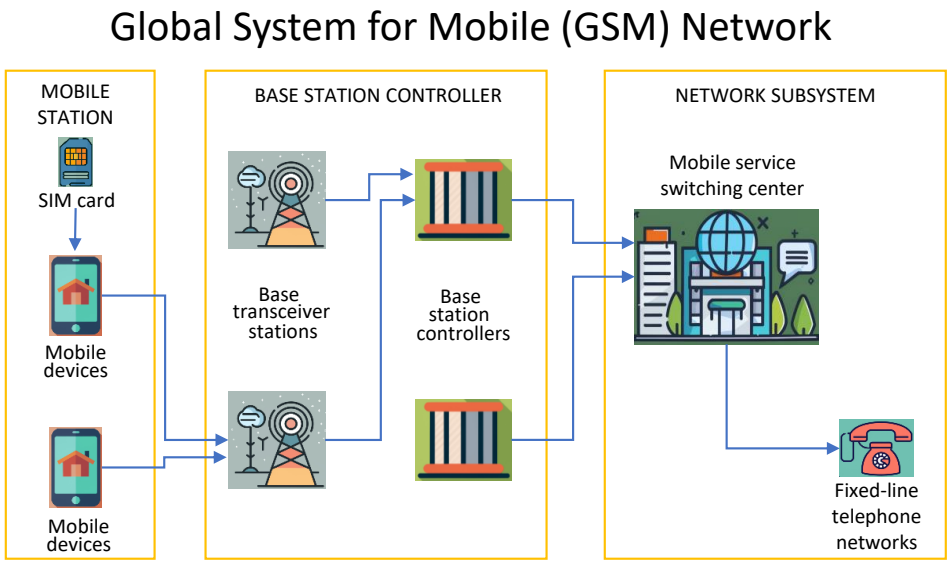

FIGURE 2. **GSM Network.**

advanced data services in subsequent network generations [3].

C. HISTORY OF 2G

Now lets talk about 2G networks or the second generation of mobile telecommunications technology, that made their debut in the 1990s. They harnessed standards like GSM (Global System for Mobile Communications), GPRS (General Packet Radio Service), or EDGE (Enhanced Data Rates for GSM Evolution) to facilitate the wireless transmission of digital signals.

This technology initiated the process by converting analog audio or multimedia data into a digital format before transmitting it from the sender to the receiver. By encoding analog signals into digital ones, 2G networks delivered substantial improvements compared to the earlier 1G networks. These enhancements encompassed superior call quality and the capability to transmit various forms of media, extending beyond mere voice [4].

In the early 1990s, 2G network technology came into existence with the development of the Global System for Mobile Communications (GSM) network standard by the International Telecommunication Union-Radiocommunication Sector (ITU-R). Subsequently, EDGE and GPRS were introduced as improvements to GSM.

Each of these technologies are discussed below.

1) GSM

GSM is a digital cellular network standard developed in Europe during the 1990s, designed to replace earlier analog cellular networks as illustrated in Fig. 2. GSM networks employ a combination of TDMA and FDMA techniques to facilitate the concurrent use of the same frequency band by multiple users. In a TDMA system, the available frequency band is divided into brief time slots, with each user allocated one or more time slots for data transmission. These short time slots, typically around 4.6 milliseconds in GSM, enable numerous users to share the frequency band simultaneously by transmitting and receiving during distinct time intervals.

Furthermore, GSM utilizes FDMA to subdivide the frequency band into smaller segments known as carriers. Each carrier is subsequently partitioned into time slots, which are then assigned to individual users. This dual-layered approach permits multiple users to coexist within the same frequency band by operating on different carriers and time slots [4].

2) General Packet Radio Service (GPRS) | 2.5G

GPRS, often referred to as 2.5G technology, serves as a transitional link between 2G and 3G networks. GPRS employs packet-switched technology to convey data across cellular networks, offering notable advantages over circuit-switched alternatives. Packet switching involves breaking data into small packets, each transmitted and received independently. This method optimizes the utilization of network resources. GPRS empowers users to engage in internet activities, exchange emails, and perform data-intensive operations on their mobile devices, complementing voice calls and text messaging functionalities. Similar to GSM, GPRS utilizes both TDMA and FDMA technologies to facilitate data

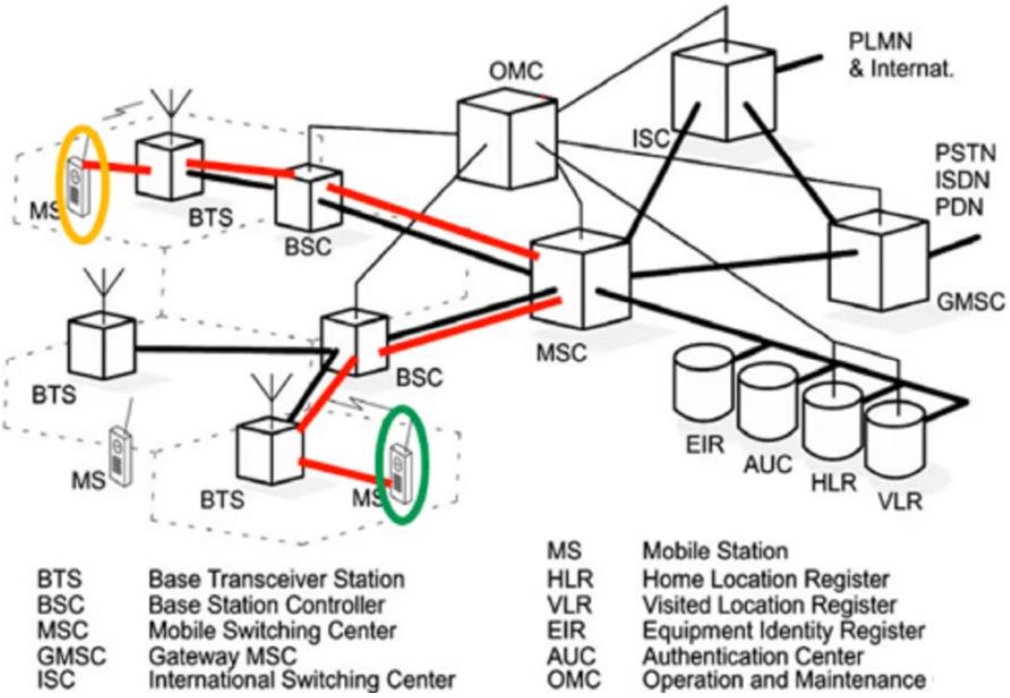

FIGURE 3. **2G Network Architecture [4].**

sharing among users [4].

3) Enhanced Data Rates for GSM Evolution (EDGE) | 2.75G

EDGE, often referred to as the '2.75G network,' represents a mobile data technology that augmented the 2G GSM cellular network standard by introducing a novel modulation scheme called 8PSK (8 Phase Shift Keying). This innovation significantly boosted data transfer rates by allowing the transmission of more bits per symbol, resulting in an overall increase in data transfer speed. One of the primary merits of EDGE is its capacity to facilitate faster and more dependable data transmission, enabling users to access advanced data services like multimedia messaging, mobile internet browsing, and video streaming. An essential feature of EDGE is its backward compatibility with existing GSM networks. This means it can be utilized on devices supporting both 2G and 2.5G technologies, facilitating widespread adoption without necessitating extensive infrastructure upgrades [4].

4) 2G Network Architecture

The 2G network architecture comprises two primary components: the radio access network (RAN) and the core network. The RAN facilitates the exchange of voice and data between mobile devices and the network. It can be further categorized into two fundamental elements: the base transceiver station (BTS) and the base station controller (BSC) as shown in Fig. 3. The base transceiver station functions as the intermediary for transmitting and receiving signals to and from mobile devices. Each BTS covers a specific area, commonly referred to as a cell. A cluster of BTSs is under the supervision of a base station controller, responsible for managing radio channel allocation and call handovers between cells.

The core network serves as the central infrastructure of the 2G network, tasked with call routing and the delivery of supplementary services like Short Message Service (SMS) and voicemail. This critical component comprises various network elements, including the mobile switching center (MSC), home location register (HLR), visitor location register (VLR), and authentication center (AUC) as shown in Fig. 3. The MSC serves as the pivotal core of the 2G network, overseeing call routing and switching operations. In contrast, the HLR assumes the role of preserving subscriber data, encompassing details like phone numbers and location information. Concurrently, the VLR temporarily retains subscriber data when they roam into distinct network areas. Finally, the AUC takes on the vital task of authenticating subscribers, guaranteeing that network access remains exclusive to authorized users [4].

Until now, we have discussed 1G and 2G principles and architectures because they are a fundamental part of progressing forward and based on them, we will be building our own open-source 2G networks to deploy IoT applications.

5) 2G Mobile Phones

2G phones marked the introduction of packet-switched technology for signal transmission between transmitters and receivers. These devices operated on 2G networks built on standards like GSM and others. 2G phones brought a wave of transformative features to telecommunications,

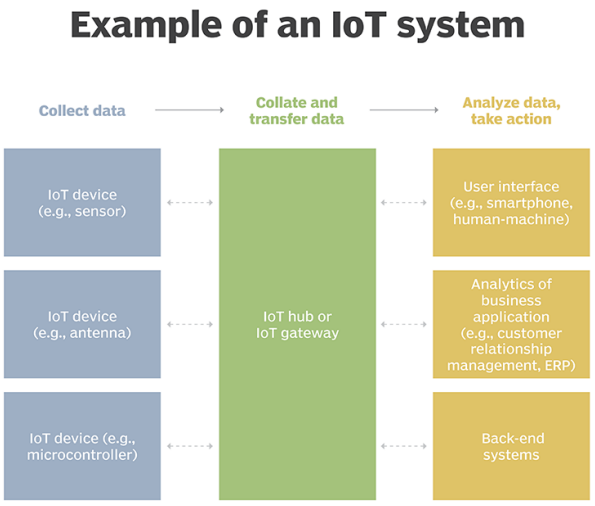

FIGURE 4. **An IoT System [5].**

including digital signal modulation, direct dialing, support for multiple access media transmission, and the introduction of messaging services.

Characteristics of 2G Phones:

2G phones incorporated a range of characteristics facilitated by the second-generation network:

- Initially, 2G phones operated on the GSM network standard, while later models utilized GPRS and EDGE.
- Modulation techniques like TDMA, FDMA, and CDMA were employed by 2G phones.
- These phones employed circuit-switched technology for transmitting data from sender to receiver.
- Digital signal modulation was adopted, replacing analog.
- Services such as SMS and MMS were enabled.
- Enhanced call quality was introduced.
- Bandwidth usage fell within the range of 30 to 200 kHz.
- Generally, 2G phones were more affordable and compact.
- 2G phones supported roaming capabilities.
- They provided clearer voice quality with reduced static.
- Efficient spectrum utilization was a hallmark.
- 2G phones exhibited lower power consumption.

Drawbacks of 2G Phones:

In contrast to more advanced generations of mobile phones, 2G had the following constraints:

- Sluggish Data Speeds: Data transfer speeds were notably slower compared to contemporary networks.
- Restricted Network Coverage: 2G networks provided limited coverage in comparison to modern counterparts.
- Absence of Advanced Features: 2G phones lacked cutting-edge features such as touchscreens, high resolution cameras, and mobile app stores.
- Security Vulnerabilities: While 2G networks did offer basic encryption for voice and data transmissions, they were susceptible to security vulnerabilities.
- Limited Battery Life: Generally, 2G phones had shorter battery life spans than their more modern counterparts.

With this the introduction of 1G and 2G is concluded, now we move towards open-source, IoT, and finally Osmocom the highlight of this article.

D. OPEN SOURCE

The phrase "open source" denotes a concept where individuals have the freedom to modify and distribute something since its design is available for public access.

Initially, this term was tossed in the realm of software development to describe a particular method of creating computer programs. Nevertheless, in present usage, "open source" encompasses a more extensive array of principles, encapsulated in what is termed "the open source way." Initiatives, products, or projects that adhere to open source values endorse and commend ideals like open sharing, cooperative engagement, swift prototyping, transparency, merit-based decision-making, and the development oriented towards community participation [6].

1) History of Open Source (Art of Sharing)

The practice of sharing technical information has a history that predates the Internet and personal computers by a significant margin. For example, during the early stages of automobile development, a group of wealthy monopolists held the patent rights to a 2-cycle gasoline engine patent initially filed by George B. Selden. Through their control of this patent, they effectively monopolized the industry, compelling automobile manufacturers to comply with their demands or face legal action [7].

In 1911, independent automaker Henry Ford successfully contested the Selden patent. This outcome rendered the Selden patent nearly worthless, and a new association, which would eventually evolve into the Motor Vehicle Manufacturers Association, was established. This new association introduced a cross-licensing agreement among all American automotive manufacturers. Under this arrangement, while each company continued to innovate and patent technology, these patents were openly shared among all manufacturers without any financial transactions. By the time the United States entered World War II, 92 Ford patents and 515 patents from other companies were freely exchanged among these manufacturers without monetary compensation or legal disputes [7].

Early instances of freely sharing source code include IBM's release of the source code for its operating systems and other software during the 1950s and 1960s, as well as the formation of the SHARE user group, which aimed to facilitate the exchange of software. In the 1960s, researchers

FIGURE 5. Open Source is King.

involved with ARPANET utilized an open "Request for Comments" (RFC) process to encourage feedback in the development of early telecommunication network protocols. This contributed to the inception of the early Internet in 1969. The sharing of source code on the Internet commenced during a period when the Internet was relatively rudimentary. Software was distributed through channels like UUCP, Usenet, IRC, and Gopher. For instance, the distribution of BSD was initially widespread through posts on comp.os.linux in the Usenet, where its development was also discussed. The Linux operating system followed a similar model [7].

What is open source software?

It is a software accompanied by source code that is accessible for scrutiny, alteration, and improvement by anyone. The "source code" constitutes the software section that typically remains hidden from the view of most computer users; it serves as the code that computer programmers can modify to alter the operation of a software component, whether it's a "program" or an "application." Programmers who possess access to a software program's source code can enhance the program by introducing new features or rectifying segments that may exhibit occasional malfunctions.

How is open source software different?

Some software is characterized by its source code, which is exclusive to the individuals, teams, or organizations that created it. This type of software is often referred to as "proprietary" or "closed source" software as shown in Fig. 6. With proprietary software, only the original authors possess the legal rights to copy, inspect, and modify the software.

FIGURE 6. Open Source Vs Close Source.

To use proprietary software, computer users typically need to agree to a license agreement displayed when they run the software for the first time. This license agreement stipulates that users must not engage in any activities with the software that the software's authors have not expressly permitted. Notable examples of proprietary software include Microsoft Office and Adobe Photoshop [6].

Open-source software, on the other hand, follows a different paradigm. The creators of open-source software make their source code accessible to anyone who wishes to view it, copy it, learn from it, make alterations, or share it with others. Software like LibreOffice and the GNU Image Manipulation Program (GIMP) exemplify open-source software. However, users of open source software must also accept the terms of a license, but the legal conditions

outlined in open source licenses are quite different from those found in proprietary licenses [6].

Open source licenses significantly impact how people can utilize, study, modify, and distribute software. In general, open-source licenses grant computer users the freedom to use open-source software for any purpose they desire. Certain open-source licenses, often referred to as "copyleft" licenses, require that anyone who modifies an open-source program must also release the source code for that modified program. Additionally, some open-source licenses dictate that anyone who makes alterations to a program and shares it with others must provide that program's source code without imposing a licensing fee. The design of open-source software licenses encourages collaboration and sharing. They empower computer programmers to access, inspect, and modify open-source software at their convenience, with the condition that they extend the same privileges to others when they share their work [6].

Is open source software only for programmers?

No, open-source technology and the principles of open-source thinking hold value for both programmers and non-programmers alike. The Internet itself, a creation of early innovators, was largely constructed using open-source technologies such as the Linux operating system and the Apache Web server application. Consequently, anyone who uses the Internet today benefits from open-source software. Every time, individuals browse web pages, manage emails, engage in chats, stream music online, or participate in multiplayer video games, their computers, mobile devices, or gaming consoles connect to a global network of computers through open-source software. This software efficiently directs and transmits data to the "local" devices in front of users. Typically, these computers responsible for such critical operations are situated in distant locations, unseen and inaccessible to users. These computers are often referred to as remote computers.

Increasingly, people rely on remote computers to perform tasks they would traditionally execute on their local devices. For instance, individuals may utilize online word processing, email management, and image editing software without installing and running these applications on their personal computers. Instead, they access these programs on remote computers through web browsers or mobile applications. This engagement in "remote computing" is sometimes synonymous with "cloud computing" because it involves activities like file storage, photo sharing, or video streaming, which encompass not only local devices but also a global network of remote computers forming a digital "cloud" around users. Cloud computing is becoming increasingly integral to daily life with internet-connected devices. Some cloud computing applications, such as Google Apps, are proprietary, while others, like ownCloud and Nextcloud, embrace the open-source philosophy. These cloud computing applications operate atop additional software that facilitates their smooth and efficient functioning. It is also often said that the software running "beneath" cloud computing applications serves as a "platform" for these applications. These cloud computing platforms can be either open source or closed source, with OpenStack serving as an example of an open source cloud computing platform [6].

Why choose open source software?

People lean toward open-source software over proprietary alternatives for a variety of reasons, including:

- *Control:* Open source software provides users with a heightened sense of control. They have the ability to scrutinize the code, ensuring it operates according to their preferences and lacks any undesired functionalities. Moreover, users can modify aspects of the software to align it with their requirements. This extends the benefit to non-programmers who can utilize open-source software for diverse purposes, unrestricted by predefined usage constraints.
- *Education:* Open source software appeals to those aiming to enhance their programming skills. As open-source code is publicly accessible, students find it conducive to studying and learning from it, fostering their proficiency in software development. Sharing work with others allows for constructive feedback and critique, contributing to skill refinement. Identifying errors in open-source software's source code also enables knowledge sharing to prevent others from repeating the same mistakes.
- *Security:* Some individuals perceive open-source software as more secure and dependable than their proprietary counterparts. Since anyone can inspect and modify open-source code, potential errors or omissions can be identified and rectified by the community, reducing vulnerabilities. The collaborative nature of open source allows for swift error correction, updates, and enhancements compared to proprietary software.
- *Stability:* Open source software finds favor among users engaged in critical, long-term projects. Publicly available source code assures users that their essential tools won't become obsolete or neglected if the original creators discontinue development. Additionally, open-source software tends to adhere to and implement open standards, ensuring compatibility, and longevity.
- *Community:* Open source software often fosters the formation of a dedicated user and developer community. While many popular applications attract user meetups and groups, open-source communities are distinct. They aren't merely fanbases; they actively participate in producing, testing, using, advocating for, and influencing the software. This sense of ownership and collaboration sets open source communities apart [6].

What is "open source" in context to beyond software?

At Opensource.com, the focus extends beyond the realm of software, and they explore how open-source values and principles can be applied to various aspects of life. Open source isn't just a method for developing and licensing computer software; it embodies a broader mindset. Embracing

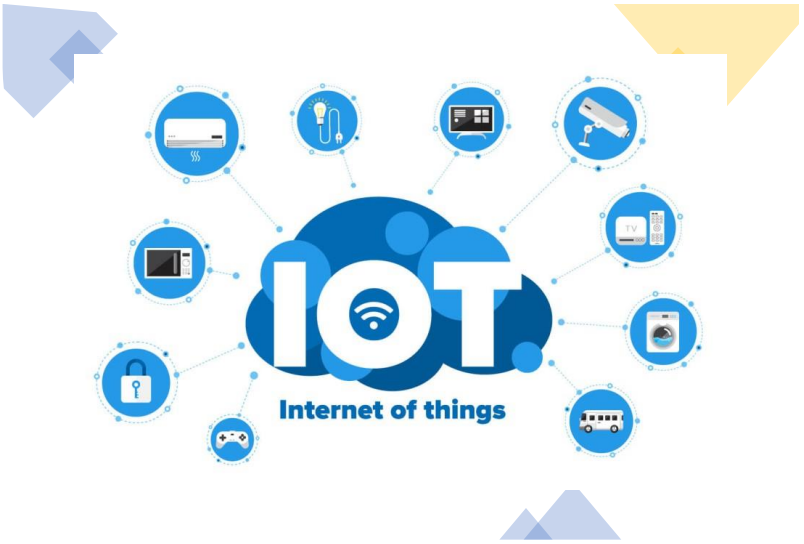

FIGURE 7. IoT connecting the world.

"the open source way" entails a readiness to share, engage in transparent collaboration (allowing others to observe and participate), view failure as a stepping stone to improvement, and actively encourage others to do the same.

It also involves a commitment to actively contribute to the betterment of our world, a goal achievable only when everyone has access to the mechanisms shaping that world. Our world is governed by "source code" in various forms such as blueprints, recipes, and regulations that influence and mold our thoughts and actions. All of us firmly believe that this foundational code, regardless of its nature, should be open, accessible, and shared to enable widespread participation in its enhancement.

On this platform, the creators share stories illustrating the influence of open-source values across diverse domains, encompassing fields such as science, education, government, manufacturing, healthcare, law, and organizational dynamics. We all are a community dedicated to demonstrating that the open source way is the most effective path forward, recognizing that, like any valuable knowledge, the true power of open source is fully realized when shared [6].

E. IoT: CONNECTING THE DOTS IN A SMART WORLD

IoT has emerged as a transformative force in the digital landscape, connecting devices, data, and people like never before. It's a concept that is reshaping industries, homes, and cities, promising a future where everyday objects communicate and collaborate to enhance our lives as depicted in Fig. 7.

1) History

The history of the IoT is characterized by a gradual evolution from early concepts to its eventual realization. While discussions about adding sensors and intelligence to everyday objects date back to the 1980s and 1990s, progress was initially slow due to technological limitations. During this period, there were a few pioneering projects, such as the development of an internet-connected vending machine. However, the key challenge lay in the fact that the necessary technology had not yet matured. Processors were too large and power-hungry, and there was no efficient means for objects to communicate [8].

The breakthrough came when cost-effective, compact processors that could function almost as disposable components became available. These processors were essential for connecting billions of devices economically. Additionally, the adoption of Radio-Frequency Identification (RFID) tags, which are low-power chips capable of wireless communication, helped address some of the technology's limitations. The growing availability of broadband internet and the expansion of cellular and wireless networking also played crucial roles [8].

Another pivotal development was the adoption of IPv6, a protocol that provides a vast number of IP addresses, ensuring there are enough for every conceivable device globally. This was a necessary step to enable the IoT to scale and accommodate the extensive network of connected devices. The term 'Internet of Things' was coined by Kevin Ashton in 1999. However, it took roughly another decade for technology to advance sufficiently to realize the IoT's full potential [8].

2) How big is the Internet of Things?

The IoT is substantial and continuously expanding, with more connected devices than there are people worldwide. According to the tech analysis firm IDC, it is projected that by 2025, the total number of connected IoT devices, often referred to as 'things,' will reach a staggering 41.6 billion. IDC also emphasizes that the industrial and automotive sectors offer the most significant opportunities for connected 'things.' However, it anticipates robust adoption of smart home and wearable devices in the near future [8].

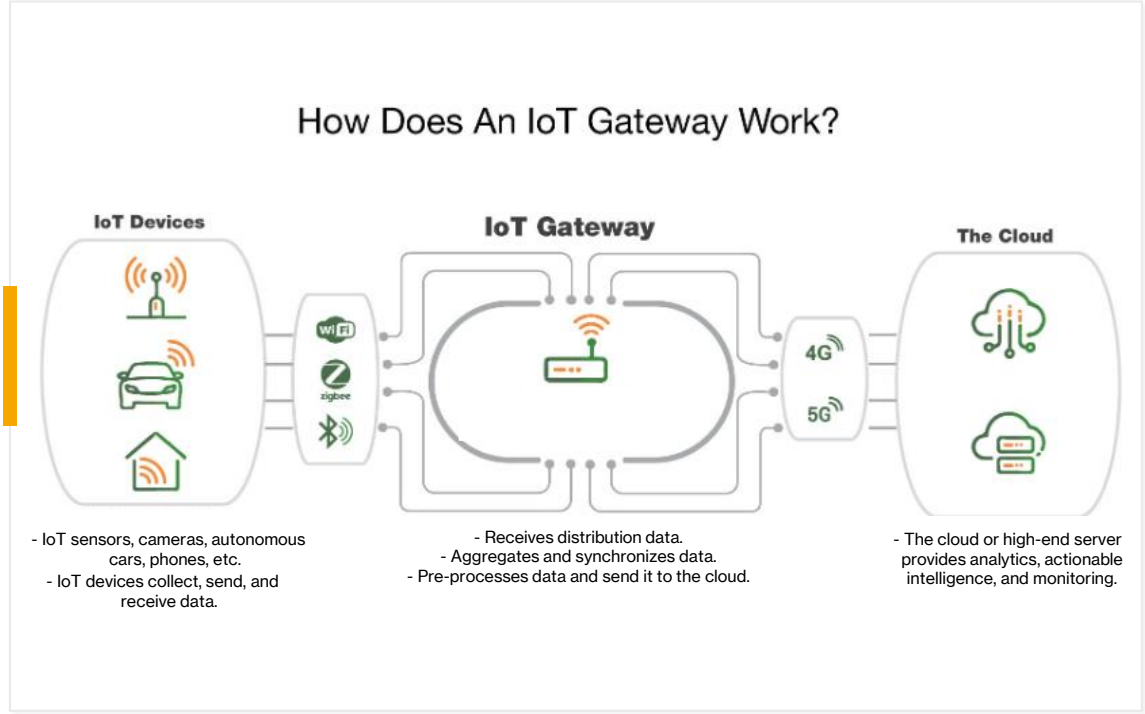

FIGURE 8. **IoT, how does it function?**

Another tech analyst, Gartner, predicts that in the current year, the enterprise and automotive industries will account for 5.8 billion connected devices, marking a nearly 25% increase from 2019. The utilities sector is poised to become the most extensive user of IoT devices due to the ongoing deployment of smart meters. Following closely behind are security devices, such as intruder detection systems and web cameras, which will represent the second most prevalent use of IoT devices. The fastest-growing sectors within IoT adoption include building automation, notably connected lighting, along with automotive (connected cars) and healthcare (monitoring chronic conditions) [8].

Another report [9], says the end-use industry has segmented into different sections globally as shown in Fig. 9, representing the total growth world wide in 2022.

Benefits of the IoT for businesses: The advantages of the IoT for businesses are contingent on the specific use case, but typically, enhanced agility and efficiency stand out as primary considerations. The fundamental concept revolves around providing organizations with increased access to data related to their products and internal systems, along with the capacity to implement necessary changes based on this data [8].

Manufacturers are incorporating sensors into their product components to enable data transmission regarding their performance. This proactive approach aids companies in identifying potential component failures and replacing them before they cause harm. Moreover, the data generated by these sensors facilitates the optimization of systems and supply chains, providing more accurate insights into operations. This comprehensive real-time data collection and analysis can significantly enhance the responsiveness of production systems [8].

Enterprise adoption of the IoT can be categorized into two segments: industry-specific solutions such as sensors in power plants or real-time healthcare location devices, and versatile IoT devices applicable across all industries, like smart climate control or security systems. Although industry-specific products will lead initially, Gartner predicts that by 2020, cross-industry IoT devices will reach 4.4 billion units, while vertical-specific devices will amount to 3.2 billion units. While consumers buy more IoT devices in quantity, businesses outpace spending. In the past year, consumer expenditure on IoT devices reached approximately $725 billion, while business expenditure reached $964 billion. The combined spending on IoT hardware by businesses and consumers is projected to approach $3 trillion by 2020 [8].

Global spending on the IoT was anticipated to reach $745 billion in 2019, marking a 15.4% increase from the $646 billion spent in 2018, as per IDC's projections. It was predicted to surpass the $1 trillion milestone by 2022. The leading industries for IoT were expected to include discrete manufacturing ($119 billion in spending), process manufac-

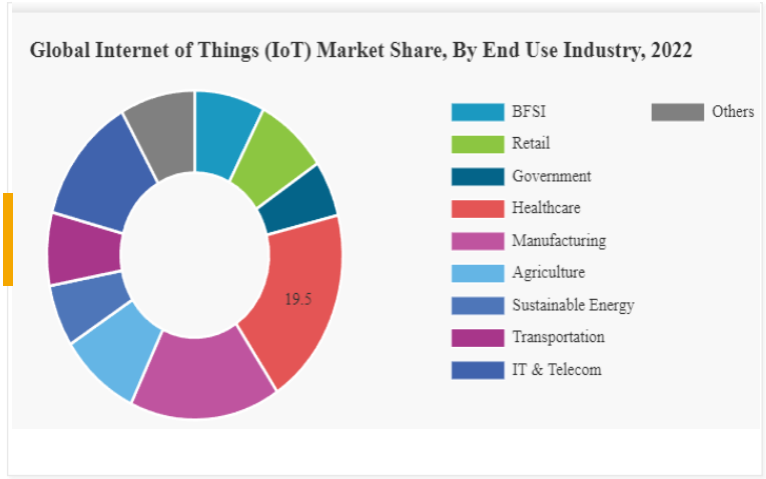

FIGURE 9. **IoT world and how big is it?**

turing ($78 billion), transportation ($71 billion), and utilities ($61 billion). In manufacturing, the focus would be on projects supporting asset management, while transportation would prioritize freight monitoring and fleet management. The utilities sector would see substantial IoT spending in smart-grid projects for electricity, gas, and water [8].

Consumer IoT spending was projected to reach $108 billion, ranking as the second-largest industry segment. Key areas for consumer IoT spending included smart home solutions, personal wellness devices, and connected vehicle infotainment. In terms of use cases, the major areas of investment were expected to be manufacturing operations ($100 billion), production asset management ($44.2 billion), smart home applications ($44.1 billion), and freight monitoring ($41.7 billion) [8].

3) Unraveling the IoT

The IoT represents a network of interconnected devices that establish communication and share data among themselves and with cloud-based systems. These IoT devices are often equipped with sensors and software, encompassing both mechanical and digital machines and various consumer objects. Across diverse industries, organizations are progressively embracing IoT to streamline their operations, elevate customer service quality, enhance decision-making processes, and augment the overall value of their businesses. IoT facilitates the transfer of data across networks without necessitating direct human-to-human or human-to-computer interactions. Within the IoT, a 'thing' encompasses a wide range of entities, including individuals with heart monitor implants, livestock with biochip transponders, vehicles equipped with sensors to notify drivers of low tire pressure, and various other natural or man-made objects. These objects can be assigned an Internet Protocol address, enabling them to transmit data over a network [5].

4) How IoT Functions:

The IoT operates through an ecosystem of web-enabled smart devices. These devices are equipped with embedded systems, encompassing components such as processors, sensors, and communication hardware. They are designed to collect, transmit, and respond to data obtained from their surroundings. IoT devices transmit the sensor data they gather by establishing connections with an IoT gateway. This gateway serves as a central hub through which IoT devices can relay their data as shown in Fig. 8. Prior to transmission, the data may also be directed to an edge device, where it undergoes local analysis. This localized data analysis serves to reduce the volume of data that needs to be transferred to the cloud, consequently minimizing bandwidth usage [5].

In certain scenarios, these devices engage in communication with other interconnected devices and take action based on the information they exchange. The devices perform most tasks autonomously, although individuals can interact with them. For instance, users may set up these devices, issue instructions, or access the data they generate. The choice of connectivity, networking, and communication protocols employed by these internet-enabled devices predominantly hinges on the specific IoT applications being employed. Furthermore, IoT has the capacity to harness artificial intelligence and machine learning to simplify and enhance data collection processes, making them more dynamic [5].

Having discussed the fundamentals of IoT and its operational aspects, let's delve into what sets it apart and highlights significance.

5) Why is IoT Significant?

IoT enhances the way people lead their lives and conduct business, introducing heightened efficiency and convenience. For instance, consumers can incorporate IoT-equipped devices, such as automobiles, smartwatches, or

thermostats, into their daily routines to elevate their lifestyles. For instance, upon a person's arrival home, their car could communicate with the garage to open the door, their thermostat could adjust to a predefined temperature, and their lighting could be configured to a softer intensity and color scheme. Moreover, IoT holds immense importance in the realm of business. It furnishes organizations with real-time insights into the functionality of their systems, offering invaluable information ranging from machinery performance to the optimization of supply chain and logistical processes [5].

IoT empowers machines to undertake repetitive tasks independently, enabling companies to automate operations, reduce labor expenditures, minimize waste, and enhance service delivery. This technology streamlines the manufacturing and delivery of goods, while simultaneously providing transparency into customer transactions. IoT stands as one of the most pivotal technological advancements, and its evolution persists as an increasing number of businesses recognize the potential of interconnected devices to maintain their competitiveness [5].

6) Advantages and Disadvantages of IoT

IoT offers several advantages, including:

- *Universal Accessibility:* Facilitates access to information from any location, at any time, and through any device.
- *Enhanced Connectivity:* Improves communication among interconnected electronic devices.
- *Efficient Data Transfer:* Enables the transfer of data packets across connected networks, resulting in time and cost savings.
- *Data Abundance:* Gathers extensive data from numerous devices, benefiting both end-users and manufacturers.
- *Edge Data Analysis:* Analyzes data at the edge, reducing the volume of data that must be transmitted to the cloud.
- *Task Automation:* Automates tasks, enhancing service quality and reducing the reliance on human intervention.
- *Continuous Healthcare:* Allows for continuous and more effective patient care in the healthcare sector.

However, it also presents some disadvantages, such as:

- *Increased Vulnerability:* Expands the attack surface as the number of connected devices grows, potentially exposing confidential information to hackers.
- *Complex Device Management:* Managing a massive number of IoT devices can become challenging for organizations, leading to data collection and management complexities.
- *Compatibility Challenges:* Compatibility issues arise due to the lack of international standards, making it difficult for devices from different manufacturers to communicate effectively.
- *System Bugs:* System bugs or vulnerabilities in IoT devices can potentially compromise the functionality of other interconnected devices."

From here onwards, we will talk about **IoT Standards and Frameworks.**

Several prominent organizations are actively engaged in shaping IoT standards [5]. These include, International Electrotechnical Commission (IEC), Institute of Electrical and Electronics Engineers (IEEE), Industrial Internet Consortium, Open Connectivity Foundation, Thread Group, and Connectivity Standards Alliance.

Examples of IoT standards encompass:

- *IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN):* An open standard established by the Internet Engineering Task Force (IETF). It enables various low-power radios, including 804.15.4, Bluetooth Low Energy, and Z-Wave, to connect to the internet. Beyond home automation, relates to the applications in industrial monitoring and agriculture.
- *Zigbee:* This low-power, low-data rate wireless network is commonly used in home and industrial settings. It is based on the IEEE 802.15.4 standard. The ZigBee Alliance introduced Dotdot, a universal language for IoT, ensuring smart objects can securely function on any network and communicate with one another.
- *Data Distribution Service (DDS):* Developed by the Object Management Group, DDS serves as an industrial IoT (IIoT) standard for real-time, scalable, and high-performance machine-to-machine (M2M) communication.

IoT often relies on specific communication protocols for device interactions. These protocols dictate how IoT device data is transmitted and received [5].

Notable IoT protocols include:

- *Constrained Application Protocol (CoAP):* An IETF-designed protocol tailored for low-power, resource-constrained IoT devices, facilitating their operations which is shown in Fig. 10.

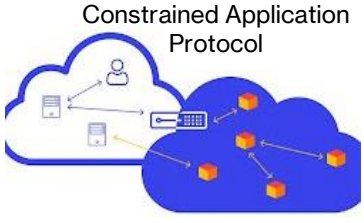

FIGURE 10. CoAP process.

- *Advanced Message Queuing Protocol (AMQP):* An open-source, standardized approach for asynchronous messaging. AMQP enables secure and interoperable messaging between applications and organizations as

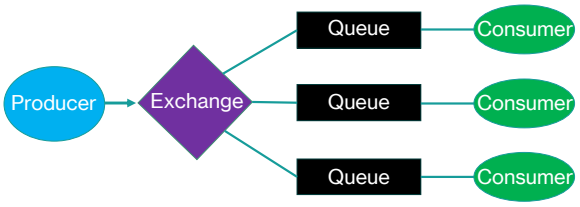

FIGURE 11. AMQP process.

shown in Fig. 11. It is used in both client-server messaging and IoT device management.

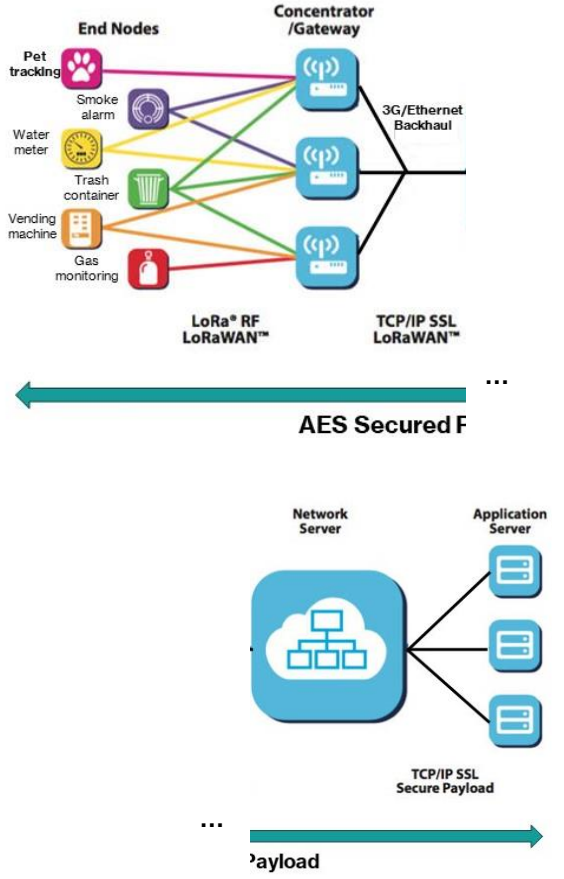

FIGURE 12. LoRaWAN process.

- *Long-Range Wide Area Network (LoRaWAN):* Designed for wide area networks, LoRaWAN supports extensive deployments like smart cities, featuring millions of low-power devices and services that are shown in Fig. 12.

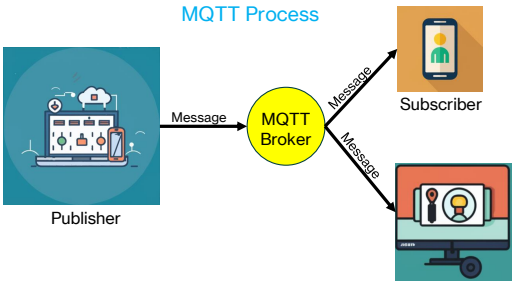

FIGURE 13. MQTT process.

- *Message Queuing Telemetry Transport (MQTT):* A lightweight protocol suitable for control and remote monitoring applications, particularly for devices with limited resources as shown in Fig. 13.

Next is IoT frameworks which encompass:

- *Amazon Web Services (AWS) IoT:* An Amazon cloud computing platform tailored for IoT. It simplifies the connection of smart devices to the AWS cloud and other connected devices, ensuring secure interactions.
- *Arm Mbed IoT:* An open-source platform for developing IoT applications based on Arm microcontrollers. It aims to provide a scalable, secure, and connected environment for IoT devices, integrating Mbed tools and services.
- *Microsoft Azure IoT Suite:* A suite of services facilitating interaction with IoT devices, data reception, multi-dimensional analysis, transformation, aggregation, and visualization. It is designed to serve various business needs.
- *Calvin:* An open-source IoT platform developed by Ericsson, intended for building and managing distributed applications that enable device-to-device communication. Calvin offers both a development framework for application developers and a runtime environment for managing running applications.

7) IoT Architecture

The IoT is projected to contribute between $5.5 trillion and $12.6 trillion to the global economy by 2030. This substantial value encompasses a wide spectrum of IoT devices, ranging from smart home appliances like light bulbs to critical sensors employed in power stations. While this economic potential is considerable, it presents the challenge of ensuring seamless cooperation among this diverse array of devices. This is where IoT architecture plays a pivotal role, encompassing its various layers, systems, and devices.

IoT architecture serves as the framework that enables internet-connected devices to establish communication with

one another. Commonly, IoT architecture models consist of three to seven distinct sets of functional components, often referred to as "layers." These layers encompass crucial aspects such as perception (e.g., sensors), transport (e.g., Wi-Fi), and application (e.g., software) layers [10].

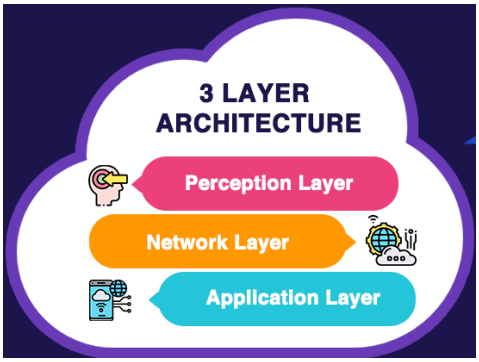

FIGURE 14. IoT Architecture with three layers [11].

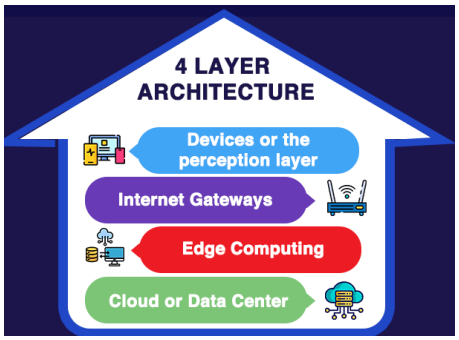

FIGURE 15. IoT Architecture with four layers [11].

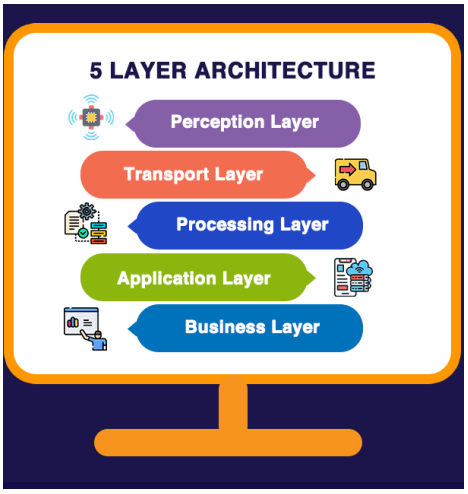

FIGURE 16. IoT Architecture with five layers [11].

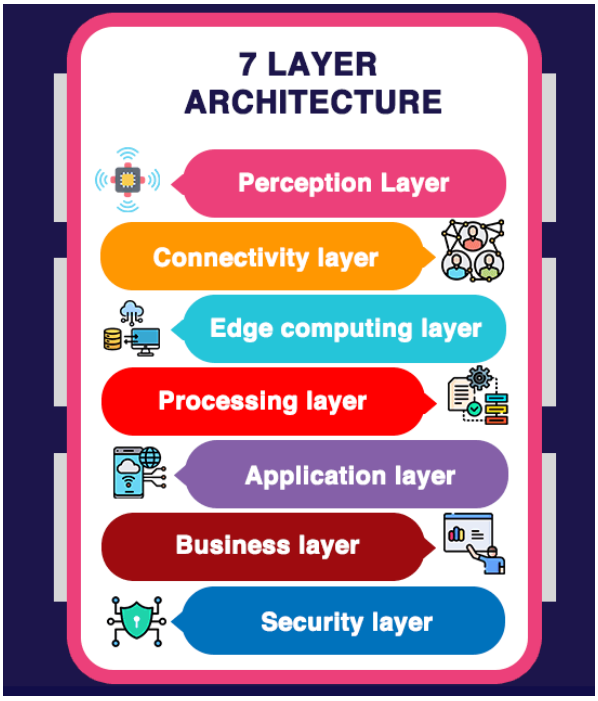

FIGURE 17. IoT Architecture with seven layers [11].

What is the IoT Architecture?

IoT architecture encompasses the various configurations and structures employed in IoT devices to fulfill specific user requirements. These IoT system components are organized into layers, typically ranging from 3 to 7 layers, each assigned a specific role within the architecture as illustrated in Fig. 17. It's important to note that IoT architecture lacks standardized protocols, which can introduce challenges related to compatibility, security, and other critical aspects.

McKinsey's projections indicate that the global count of IoT devices will surpass 43 billion by 2023. These myriad devices are already driving transformative changes across various sectors, enabling activities like remote patient monitoring in healthcare and spill prevention in the oil industry. The foundation for this extensive growth is provided by IoT architecture [10].

Layers of IoT Architecture?

IoT architecture typically encompasses a range of layers, which can number up to seven, including the perception, transport, edge, processing, application, business, and security layers.

- **Perception Layer:** The first layer in IoT system architecture, known as the perception layer or device layer, comprises various components such as sensors, cameras, actuators, and similar devices. These elements are responsible for data collection and performing specific tasks. For instance, consider an IoT sensor employed in an automotive assembly line. This sensor can carry out quality control checks on nearby robots. When a robot assembles a fuse box, the IoT sensor examines whether the fuse has been correctly placed by detecting the color coding of different fuses [10].
- **Transport Layer:** Within the IoT system architecture, the transport layer is responsible for conveying data from various devices, including on-site sensors, cameras, and actuators, to an on-premise or cloud-based data center. To initiate this process, IoT gateways play a crucial role in converting incoming analog data into a digital format. Subsequently, the gateway utilizes a selection of data transfer protocols (DTPs) to send this data to the on-premise or cloud data center [10].
  Significant factors that influence the selection of a data DTP include:
  - The volume and nature of the data to be transmitted.
  - Desired speed and frequency of data transmission.
  - Network connection reliability.
  - Power consumption during data transfer.
  - Data and network security.
  - Communication between edge devices.

  Various DTPs used in IoT networks offer distinct advantages and disadvantages in relation to these factors. Here are some of the most diverse and widely utilized IoT protocols:

  – *MQTT*: This is a lightweight protocol featuring publish/subscribe interaction models, originally developed by IBM. It has gained popularity as the most widely used protocol in the IoT domain, owing to its open-source nature and suitability for devices in remote areas with limited internet connectivity.
  – *Modbus*: Initially designed for use with Modicon's programmable logic controllers (PLCs, now Schneider Electric), the Modbus data communications protocol is a preferred choice for connecting a supervisory computer to remote terminal units (RTUs) in IoT systems, following the supervisory control and data acquisition (SCADA) model.
  – *AMQP*: Led by JP Morgan Chase, one of the largest U.S. banks, AMQP was primarily developed for data transmission within the financial services sector. AMQP boasts an integrated security framework employing components like transport layer security (TLS) and a simple authentication and security layer (SASL).
  – *PROFINET (Process Field Network)*: PROFINET is an Ethernet-compatible protocol, is developed and supported by PROFIBUS & PROFINET International (PI), an automation community based in Germany. It has gained widespread adoption in industrial automation systems requiring communication among multiple edge devices, machinery, and software systems.
  – *CAN (Controller Area Network) Bus*: Originally crafted by Bosch, a German engineering and technology firm, the CAN bus protocol was created for the automotive industry, enabling different devices and sensors within vehicles to communicate directly, eliminating the need for an intermediary computer. CAN bus has since been adapted for various two-way device communication applications, including maritime vessels, construction equipment, lighting control systems, and elevator and escalator controls.
  – *EtherCAT (Ethernet for Control Automation Technology)*: The EtherCAT Ethernet-based protocol was initially developed by the German industrial automation company Beckhoff for systems demanding real-time data updates. Supported by the EtherCAT Technology Group (ETG), an industrial consortium with nearly 7,000 member organizations, EtherCAT is among the most extensively employed IoT gateway protocols.
  – *Other DTPs*: Numerous additional DTPs, such as Constrained Application Protocol (CoAP) and Data Distribution Service (DDS) are extensively used in both industrial and non-industrial IoT applications, spanning domestic lighting, security, and smart healthcare devices, among others.

- **Edge Layer:** As IoT networks expand, latency emerges as a significant performance challenge. The sheer volume of devices connecting to a central hub can congest the network. To address these issues, the edge layer of an IoT system architecture comes into play. Edge computing, a key component of the edge layer, tackles these challenges by facilitating data processing and analysis as close to the data source as possible. This approach minimizes latency and optimizes network efficiency [10].
  One common feature shared by all IoT edge devices is their capability to transmit detected data in the form of data packets to nodes for further data processing. Some "smart" edge devices are even programmed to automatically halt target processes or initiate damage control measures upon detecting significant anomalies. IoT edge devices are typically designed to seamlessly interact with devices from various manufacturers. This interoperability is crucial for the widespread adoption of edge devices within IoT systems, particularly at scale [10].

  – *Data Pre-Processing:* It is a rapidly evolving

Pros	Cons
Immediate insights.	High cost of edge processors.
Reduced data transmission costs.	Concerns regarding remote data security.
Localized resolution of problems.	Potential loss of valuable data insights.

TABLE 1. Pros and Cons of Data Pre-Processing at IoT Gateways.

aspect of IoT systems, offering the potential to swiftly detect and resolve issues while mitigating the expense associated with transmitting extensive data volumes. However, the use of advanced processors can elevate network layer setup and maintenance costs. Moreover, pre-processing architecture introduces the risk of filtering out valuable data generated by IoT devices before it undergoes further processing [10].

A concise overview of the advantages and disadvantages of implementing data pre-processing at the network level is tabulated in Table 1.

- **Processing Layer:** A core element within the structure of an IoT system architecture is its processing layer, also known as the middleware layer. This layer efficiently harnesses the power of multiple interconnected computers, often in the form of cloud computing resources. It is designed to provide exceptional capabilities in terms of computing, storage, networking, and security [10].

  The primary responsibility of the processing layer in an IoT system architecture lies in the analysis of input data. Its role encompasses the generation of fresh insights, valuable predictions, and timely warnings. Given that IoT systems routinely manage vast volumes of data originating from numerous edge devices distributed across multiple network edges, the 'middleware' within the processing layer employs a three-stage methodology to preprocess this data before it reaches the application layer such as:

  - *Data Accumulation:* Within the middleware, data is meticulously categorized and directed to the appropriate storage locations based on its type. Unstructured data, such as audio and video streams, as well as images, tend to consume more storage space and are typically housed in data lakes. In contrast, structured data, which includes instrument readings, log values, and measurements (telemetry data), is more space-efficient and finds its place in data warehouses.
  - *Data Abstraction:* This phase involves the consolidation of data from various sources, along with the transformation of data into a format that can be easily interpreted by the software utilized in the application layer.
  - *Data Analysis:* This step harnesses the capabilities of machine learning (ML) or deep learning algorithms, which specialize in uncovering patterns within extensive and seemingly chaotic data sets.

- **Application Layer:** Within the framework of an IoT system architecture, the application layer plays a crucial role in deciphering meaningful patterns within IoT data and presenting them in user-friendly formats, such as graphical representations and tabular summaries. Examples of the application layer in IoT architecture encompass software programs for device management and monitoring, as well as specialized process control software [10].
- **Business Layer:** Patterns deciphered at the application level serve as a foundation for extracting valuable business insights, forecasting future trends, and guiding operational decisions aimed at enhancing efficiency, safety, cost-effectiveness, customer satisfaction, and other critical aspects of business operations. These tasks are effectively carried out within the business layer of an IoT system architecture [10].

  - *Case Study – IoT System Architecture:* Celli Group, an Italian manufacturer specializing in beverage and beer dispensing equipment, recognized a persistent challenge within the industry. Bar operators often struggled to assess the condition of their dispensing equipment and efficiently manage inventory, resulting in inconsistent product quality and missed sales opportunities [10].

    To address this issue, Celli Group adopted an IoT solution, leveraging the power of Microsoft Azure and PTC's Industrial Internet of Things (IIoT) platform, known as ThingWorx. This innovative approach led to the development of IntelliDraught, a retrofit system designed to transform traditional bar dispensers into intelligent dispensing systems. IntelliDraught empowers bar operators to gather and transmit data to a central processing system, offering invaluable insights into the status of their dispensing equipment, beverage quality, and consumer consumption patterns.

    Consequently, Celli played a pivotal role in elevating customer satisfaction among bar operators by a remarkable 27% while also boosting sales by an impressive 16%. Furthermore, Celli effectively harnesses the wealth of data generated by IntelliDraught to unveil fresh insights pertaining to beer consumption, bar inventories, and drinking behavior [10].

- **Security Layer:** Security stands as a paramount necessity within an IoT system architecture. Paradoxically, it also presents itself as one of the foremost hurdles

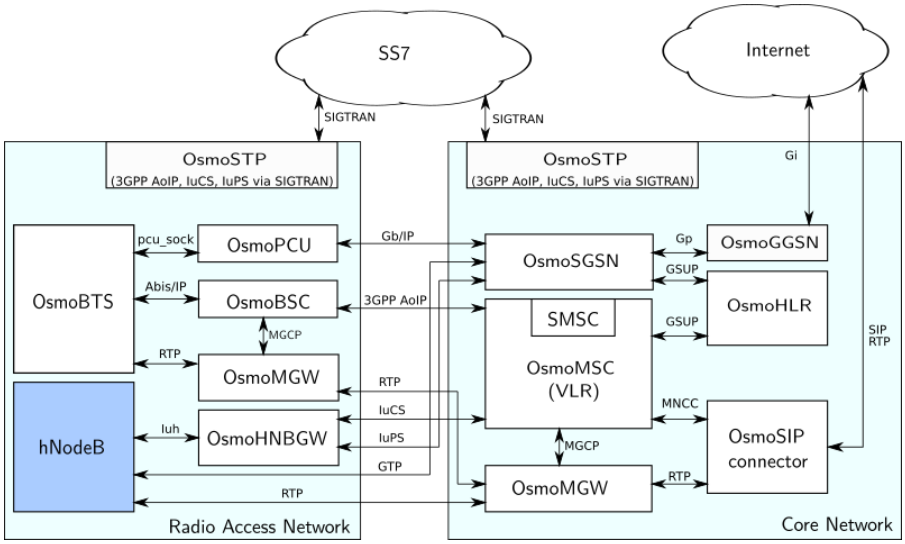

FIGURE 18. **The components you need [12].**

confronting both IoT architecture and the IoT devices they encompass. In a broad sense, the IoT security layer encompasses three primary facets:

- *Device Security:* Encompasses the physical IoT devices and entails safeguarding these endpoints from potential malware and unauthorized intrusions.
- *Cloud Security:* Given that a substantial portion of IoT data is processed within the cloud, ensuring cloud security assumes pivotal importance in thwarting data breaches and leaks.
- *Connectivity Security:* Concentrates on fortifying the security of data traversing networks, predominantly through the implementation of encryption. The TLS protocol is regarded as the gold standard for ensuring the security of IoT connections.

F. HOW 2G STEPPED IN TO COVER FOR IoT

At its launch in 1991, 2G marked a significant technological advancement, ushering in a new era of wireless digital communication. Building upon the foundation of 1G technology, which primarily facilitated analog voice transmissions over relatively unsecured channels, 2G introduced a range of pioneering features. These innovations encompassed call and text encryption, SMS and MMS messaging, and the ability to transmit images, all at speeds nearly 25 times faster than the preceding 1G technology [13].

1) The Vital Role of 2G Technology and Narrowband IoT

2G technology continues to be in use worldwide today, despite its relatively high power consumption when compared to modern standards. It is especially favored by European IoT adopters and companies, thanks to its affordability, extensive coverage, and sufficient speed – qualities that are also characteristic of the emerging Narrowband Internet of Things (NB-IoT) solutions. NB-IoT represents a cellular Low Power Wide Area Network (LPWAN) standard that connects devices globally by using a narrow-band approach, efficiently utilizing radio spectrum resources within the LTE mobile network framework. Beyond its effective spectrum and power management, NB-IoT excels in reaching challenging locations, such as utility meter closets, underground garages, and pump rooms [13].

This technology is poised to establish a new wireless foundation for applications that don't necessitate high mobility, spanning applications from smart metering, smart cities, home security systems, building climate control, asset and facility management, to portions of supply chain management. It is estimated that by 2020, nearly 24 billion everyday devices will be IoT-enabled. Undoubtedly, NB-IoT is set to transform the market. However, it still faces certain obstacles related to hardware, data plan costs, and global coverage, which contribute to the adoption challenges worldwide. While the world awaits widespread NB-IoT solutions, Europe is well-placed to mitigate risks and expedite IoT solutions across the continent by revitalizing the two-decade-old 2G technology [13].

2) Why Choose 2G? Europe Leads the Way in IoT Mobility

In the United States, the phase-out of 2G technology is well underway, largely due to congestion in the cellular radio spectrum. However, for the majority of the world, 2G technology remains highly practical. For instance, in the realm of smart buildings, there's no need for transferring large files, streaming videos, or engaging with data-intensive applications like Instagram when the goal is simply to adjust

the lighting in a parking facility or monitor temperature and humidity levels in a distant pump room.

With the ultimate aim of establishing an NB-IoT LTE infrastructure, introducing a completely new standard across a network of millions of cell towers throughout the continent is a formidable challenge. What Europe, the Middle East, and Africa (EMEA) have in place is a robust and reliable 2G network of mobile networks, alongside a multitude of 2G IoT modules engineered to function with greater intelligence and efficiency than ever before [13].

With this for the readers, we have explained in depth the origins of 1G, 2G, Open-source, IoT, and why 2G is better for IoT. Now we focus on the implementation of the 2G network for IoT indoor and outdoor applications. First, we will talk about the 2G employment through Osmocom technology, then we will present the deployment scenario of IoT using Osmocom.

II. OSMOCOM: WHAT IS IT?

Osmocom, short for open-source mobile communications, is a software project available as open source. It is designed to support a range of mobile communication standards such as GSM, DECT, TETRA, and more. In 2008, Harald Welte and Dieter Spaar conducted experiments using a Siemens base transceiver station that had reached the end of its operational life. They initially developed the BSC side of the A-bis protocol, which later evolved into what we now know as OpenBSC. As the project gained traction, it expanded its compatibility to include other base transceiver station (BTS) models. OpenBSC was officially released during the 25th Chaos Communication Congress in December 2008.

In 2010, a GSM stack implementation designed for mobile phones was created and called OsmocomBB. These projects, along with OpenBSC, were unified under the new Osmocom umbrella project. In 2011, Harald Welte and Holger Freyther established the company Sysmocom GmbH to offer commercial support for these projects. Starting in 2018, Osmocom software and Sysmocom hardware have been utilized in Villa Talea de Castro, Mexico, to deliver a cellular network serving approximately 3,500 residents [14].

Osmocom project was aimed at creating a freely available software implementation of the GSM protocol stack and its components. Figure 18 showcases common possible configurations of the Osmocom software. It was designed to operate on Linux and required an E1 interface (specifically the ISDN Primary Rate Interface via mISDN). The software was primarily written in the C programming language and was distributed under the GNU General Public License (version 2 or later).

The initial release of OpenBSC adhered to GSM specifications 21.12 and 08.5x and was tailored to function with a specific Base Transceiver Station, the Siemens BS11 MicroBTS. OpenBSC included the implementation of various Mobile Switching Center (MSC) components, which encompassed the A-bis protocol (the communication protocol between the Base Transceiver Station and the Base Station Controller), Authentication Center (AUC), Home Location Register (HLR), Visitor Location Register (VLR) that used SQL tables for data storage, and a Short Message Service Switching Center. OpenBSC could be accessed through telnet. The software extended support to different BTS devices, including the Siemens BS11 (micro BTS) with an E1 Primary Rate interface and the IP. access nano BTS with a Power over Ethernet (PoE) interface [14].

OpenBSC is now considered outdated, and its functionalities have been divided into distinct projects: OsmoBSC, OsmoMSC, and OsmoHLR. Each one of these projects has its own guide manuals and instructions.

- **SDR:** Steve Markgraf is credited with discovering rtl-sdr, and he also developed osmo-fl2k for radio transmissions. These projects made the use of OsmoSDR outdated.
- **OsmoTETRA:** The OsmoTETRA project is dedicated to the implementation of the TETRA protocol, specifically focusing on the lower layers of this protocol. Research conducted during the project has exposed security vulnerabilities in some government communications.
- **OsmocomBB:** OsmocomBB stands as a free firmware designed for the baseband processor within mobile phones, managing the encoding and radio communication for both voice and data. Notably, OsmocomBB is the sole available free implementation of baseband firmware, setting it apart from unsuccessful projects like TSM30 by THC and MadOS.

OsmocomBB handles the three lowest OSI Layers of the GSM protocol stack on the client side, along with device drivers. The protocol layers, constituting the core, are housed on the baseband processor, typically featuring an ARM processor and a digital signal processor. OsmocomBB offers compatibility with the Calypso chipset developed by Texas Instruments. Karsten Nohl expanded OsmocomBB's capabilities to enable the detection of IMSI catchers [14].

Now, we will present the traditional **Osmocom Network In The Box**[1] which serves as a concise overview of the simplest and essential configuration for an Osmocom 2G and/or 3G network designed for voice and data services. It provides an excellent foundation for beginners to get acquainted with the software and serves as a platform for further exploration through the Osmocom Manuals and additional wiki resources [12].

In the past, Osmocom provided the OsmoNITB, known as the "Network-In-The-Box," as a single integrated program. While this was a convenient approach, in 2017, Osmocom made the decision to restructure OsmoNITB into separate programs that align more with conventional network architecture. It is advisable to use these new individual components instead of OsmoNITB, as the primary focus of ongoing development has shifted to them.

[1]This is an obsolete version must only be used for practice or learning for the first time.

Required for				Program	Description
2G		3G			
CS	PS	CS	PS		
✓	✓	✓	✓	OsmoHLR	Home Location Register, stores subscriber IMSI, phone number and auth tokens.
✓	(1)	✓ (3)	(1)	OsmoMSC	Mobile Switching Center, handles signaling, i.e., attach/detach of subscribers, call establishment, messaging (SMS and USSD).
✓		✓		OsmoMGW	Media Gateway, is instructed by the MSC and/or the BSC to direct RTP streams for active voice calls.
✓	✓	✓	✓	OsmoSTP	Signal Transfer Point, routes SCCP messages between MSC, BSC, HNBGW and for 3G also the SGSN.
✓	(1)			OsmoBSC	2G Base Station Controller, manages logical channels and other lower-level aspects for one or more 2G BTS; it is technically part of the BSS and not the "core network".
		✓	✓	OsmoHNBGW	3G HomeNodeB Gateway, receives the Iuh protocol from a 3G femto cell and forwards to MSC and SGSN by SCCP/M3UA via OsmoSTP.
	✓ (2)		✓ (2)	OsmoGGSN	Gateway GPRS Support Node, "opens" GTP tunnels received from SGSNs to internet uplink.
	✓		✓ (3)	OsmoSGSN	Serving GPRS Support Node, handles signalling, i.e. attach/detach of subscribers and PDP contexts.
✓	(1)			OsmoBTS	for 2G networks, drives the TRX and ties to the BSC via Abis-interface.
	✓			OsmoPCU	for 2G networks, a component closely tied to the BTS, drives the TRX for PS timeslots and ties to the SGSN via Gb-interface.
		✓	✓	hNodeb	3rd party 3G femto cell hardware to connect to OsmoHNBGW via Iuh
				OsmoSIPConnector	Optional: switch OsmoMSC to external MNCC and forward Call Control and RTP to a PBX of your choice.

FIGURE 19. **The components you need part 2 [12].**

Nonetheless, it is still entirely feasible to operate a comprehensive Osmocom core network within a single unit.[2] For instance, a sysmoBTS can manage the entire core network on the same hardware responsible for driving the TRX, effectively creating a complete network within a single device. Simultaneously, the availability of separate components allows for scalability in large-scale deployments, with a well-distributed workload and a centralized subscriber database.

Within this comprehensive network setup, given that your radio hardware is prepared (such as a BTS, a femto cell, or an SDR operated by osmo-trx), the core network comprises distinct programs that deliver voice, SMS, and USSD services (referred to as "circuit-switched" or CS), as well as data services (known as "packet-switched" or PS). For these, what the users need are shown in Fig. 19:

1: In the realm of mobile network components, it's essential to recognize that PS, or Packet Switching, is a supplementary feature to CS, or Circuit Switching. Even though CS elements don't directly manage PS requests, they play a crucial role in network setup and registration. This is a prerequisite for accessing data services. Generally, this situation arises due to mobile phone policies. Theoretically, it's possible for them to connect to networks primarily offering data services without voice functionality.

2: For the GGSN (Gateway GPRS Support Node) to effectively route packets to an internet uplink, it requires the establishment of a tun device and typically the activation of IP masquerading or forwarding.

3: When you're in the process of building from the source code, remember to include the –enable-iu option. This ensures that the essential components are correctly configured and enabled, which is crucial for seamless operation.

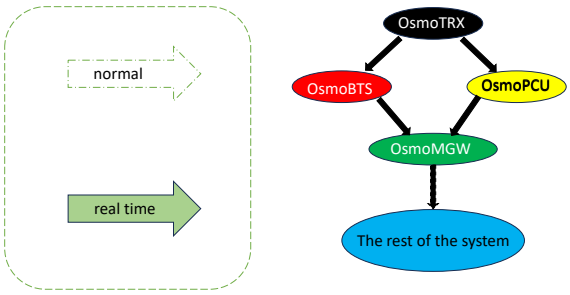

FIGURE 20. **Realtime scheduling hierarchy [12].**

Each Osmocom program must have a distinct configuration file, a VTY telnet console for live interaction, and a CTRL interface for live interaction from 3rd party programs.

The time-critical components employ a real-time scheduling policy, which doesn't necessarily demand a real-time kernel. While a real-time GNU/Linux kernel can provide enhanced guarantees, such as minimizing jitter and consistently meeting process execution deadlines, these are not mandatory in our case [12]. The priority structure is

[2]To migrate from OsmoNITB to the new separate programs, see the OsmoNITB Migration Guide [15].

straightforward: components at the top have higher priority as revealed in Fig. 20.

1) Real-Life Applications of Osmocom

To truly appreciate the impact of Osmocom, let's delve into real-life examples where it has played a pivotal role:

Example 1: In rural India, where traditional telecommunications infrastructure struggles to reach, Osmocom has emerged as a beacon of hope. A collaborative effort between a local NGO and tech enthusiasts leveraged Osmocom's open-source architecture to set up a cost-effective 2G network. This network provides vital connectivity for farmers, enabling them to access weather forecasts, market prices, and educational resources.

Example 2: In the wake of natural disasters in the Philippines, rapid communication is a lifeline. Osmocom's quick deployment capabilities proved invaluable in establishing temporary communication networks. Relief teams utilized Osmocom-powered networks to coordinate efforts, locate survivors, and provide medical assistance, showcasing the resilience of open-source solutions in times of crisis.

Example 3: In remote African villages with limited access to formal education, Osmocom has opened doors to learning. An innovative project, supported by a global nonprofit, deployed Osmocom-based networks to deliver educational content via mobile devices. This initiative has not only improved literacy rates but also empowered communities with knowledge and skills.

III. CONSTRUCTING PRACTICAL 2G NETWORKS WITH OSMOCOM: AN OPEN SOURCE GUIDE

From here on, we will discuss the basic building blocks of the Osmocom and their functionality [3].

```
hlr
 ssd route prefix *#100# internal own-msisdn
 ussd route prefix *#101# internal own-imsi


log stderr
 logging filter all 1
 logging print extended-timestamp 1
 logging print category 1
 logging print category-hex 0
 logging print level 1
 logging print file basename last
 logging level set-all debug
```

FIGURE 21. osmo-hlr.cfg [12].

1) OsmoHLR

OsmoHLR, known as the Home Location Register, acts as the central hub for storing vital subscriber data including IMSI, phone numbers, and authentication tokens. This is where you manage who's granted access to your network and assign phone numbers. OsmoHLR also takes care of USSD services, like those accessed via codes such as "*100#" as shown in the Fig. 21.

Osmo-hlr simplifies the process by automatically creating an initial subscriber database. Consult the manual for instructions on adding one or more subscribers. If you're uncertain about your IMSI, you can initiate a connection attempt and monitor the OsmoHLR log for any rejected IMSIs.

Interestingly, while a configuration file is necessary, it might remain empty. OsmoHLR will function via GSUP on the local host (127.0.0.1), which is adequate for a setup where the MSC and SGSN are on the same machine as the HLR. As an added feature, this example also provides the option to configure two USSD services and enable logging as needed [12]. Once the HLR is running, you will want to add subscribers with authentication keys to the HLR database [4].

```
network
 network country code 901
 mobile network code 70
msc
 mgw remote-ip 192.168.0.9
 # For nano3G:
 iu rab-assign-addr-enc x213
log stderr
 logging filter all 1
 logging print extended-timestamp 1
 logging print category 1
 logging print category-hex 0
 logging print level 1
 logging print file basename last
 logging level set-all info
```

FIGURE 22. osmo-msc.cfg [12].

2) OsmoMSC

OsmoMSC takes on the pivotal role of managing various network operations. It's responsible for handling signaling, including tasks like subscriber attachment and detachment, call setup, and messaging functions like SMS and USSD. Within the OsmoMSC system, the VLR component is dependent on connecting to OsmoHLR's GSUP server to determine authorized subscribers. Typically, this connection is established with OsmoHLR located on the localhost, requiring no explicit configuration [12].

To ensure accessibility by OsmoBSC and OsmoHNBGW, OsmoMSC requires an SCCP point code and must establish a connection with OsmoSTP to effectively integrate with SCCP routing. By default, it employs a point code, currently set at 0.23.1 in the 8.8.3 point code format [12]. OsmoMSC

[3] You can also check their tar file configurations from here [16].

[4] For this you can refer to Osmocom Manuals, section "Managing Subscribers" [17].

automatically looks for OsmoSTP on the localhost's M3UA port, numbered 2905. For the management of RTP streams, OsmoMSC relies on an OsmoMGW instance. Configuration primarily revolves around specifying your NCC, MNC, and configuring how to reach and use the MGW as shown in the steps in the Fig. 22.

```
mgcp
  bind ip 192.168.0.9
line vty
  bind 127.0.0.1

log stderr
  logging filter all 1
  logging print extended-timestamp 1
  logging print category 1
  logging print category-hex 0
  logging print level 1
  logging print file basename last
  logging level set-all info
```

FIGURE 23. **osmo-mgw-for-msc.cfg [12].**

3) OsmoMGW

OsmoMGW, also known as the Media Gateway, operates under the guidance of the MSC and/or the BSC, directing RTP streams for active voice calls. These directives, in the form of MGCP messages, are transmitted by OsmoMSC/OsmoBSC. The core function of the Media Gateway is to efficiently forward RTP streams between various network components, such as BTS, femto cells, and remote endpoints, which may include other MGW instances. In the future, it will also provide transcoding services between different codecs. In your network setup, you require an OsmoMGW to handle MGCP requests from OsmoMSC and a separate OsmoMGW to manage MGCP requests from OsmoBSC. Alternatively, both of these tasks can be managed by a single OsmoMGW instance. In this case, it's crucial to ensure that they don't attempt to bind to identical ports on the same IP address, not only for MGCP but also for VTY and CTRL interfaces [12].

Consider a scenario where you have a 2G network with an external BTS, like a sysmoBTS. In such a setup, your OsmoBSC's MGW instance needs to be accessible via a public interface. On the other hand, the MSC's MGW can be on a local loopback interface, as it only needs to be reachable by the BSC's MGW and the MSC. If you introduce a 3G femto cell into your network, the MSC's MGW instance also requires access via a public interface. This may necessitate two public interface addresses or the allocation of different MGCP ports to one of the MGWs. If your network structure permits, you can choose to use a single OsmoMGW for both the BSC and MSC, as the MGW now automatically manages endpoint configuration for each. To enhance the chances of a successful initial setup, the provided examples specify distinct MGCP ports and VTY interfaces, enabling the simultaneous operation of two MGWs on the same public IP address [12].

4) OsmoMGW for OsmoMSC

In a scenario where your setup is entirely contained within a single device, such as a sysmoBTS or osmo-trx with an integrated core network, the listening IP address can be set to the default, localhost (127.0.0.1). In such cases, you may leave out the 'bind ip' configuration in your file [12].

However, if you are operating with a separate BTS and/or RNC, for example, a 3G femtocell or nanoBTS, it's crucial to configure an IP address that is reachable by both the hNodeB and BTS components as shown in Fig. 23.

```
mgcp
  bind ip 192.168.0.9
  # default port is 2427 (is used for MSC's MGW)
  bind port 12427
line vty
  # default VTY interface is on 127.0.0.1 (used for MSC's MGW)
  bind 127.0.0.2

log stderr
  logging filter all 1
  logging print extended-timestamp 1
  logging print category 1
  logging print category-hex 0
  logging print level 1
  logging print file basename last
  logging level set-all info
```

FIGURE 24. **osmo-mgw-for-bsc.cfg [12].**

5) OsmoMGW for OsmoBSC

OsmoBSC, in its operation, also necessitates the presence of an OsmoMGW instance. If your setup allows direct accessibility for both OsmoBSC and OsmoMSC, you can opt for a shared OsmoMGW instance. In this configuration, endpoints are allocated dynamically.

However, for better semantic clarity, it's advisable to run a separate OsmoMGW instance specifically for OsmoBSC. When these components operate on the same machine, each MGW needs to employ distinct UDP ports. For instance, as depicted in Fig. 24. Please be aware that the 'mgw remote-port' specified in the osmo-bsc.cfg file corresponds to the 'bind port' configuration set here. If the MGWs are running on separate interfaces, the default ports will suffice in both situations [12].

6) OsmoMGW for OsmoHNBGW

Starting in 2022, OsmoHNBGW now includes support for a local hop MGW instance concerning 3G's IuUP/RTP functionalities as shown in Fig. 25. Please take note that the osmo-hnbgw.cfg details provided in Fig. 25 designate

```
mgcp
  bind ip 192.168.0.9
  bind port 22427
line vty
  bind 127.0.0.3

log stderr
  logging filter all 1
  logging print extended-timestamp 1
  logging print category 1
  logging print category-hex 0
  logging print level 1
  logging print file basename last
  logging level set-all info
```

FIGURE 25. **osmo-mgw-for-hnbgw.cfg [12].**

the 'mgw remote-port' as the 'bind port' set here. In the event that the MGWs operate on separate interfaces, the default ports will suffice in both scenarios.

```
cs7 instance 0
  xua rkm routing-key-allocation dynamic-permitted
  listen m3ua 2905
    accept-asp-connections dynamic-permitted

log stderr
  logging filter all 1
  logging print extended-timestamp 1
  logging print category 1
  logging print category-hex 0
  logging print level 1
  logging print file basename last
  logging level set-all info
```

FIGURE 26. **osmo-stp.cfg [12].**

7) OsmoSTP

OsmoSTP serves as the Signal Transfer Point, essentially functioning as a network switch that efficiently directs messages between SS7 system components. Typically, there's little need to delve into its logging or configuration [12].

OsmoSTP operates as a server to route SCCP messages. OsmoMSC, OsmoBSC, OsmoHNBGW, and OsmoSGSN establish contact with OsmoSTP, disclosing their respective point codes. Subsequently, they can direct OsmoSTP to route SCCP messages to one another based on these point codes. The fundamental setup enabling dynamic routing is described in Fig. 26.

8) OsmoBSC

OsmoBSC serves as the 2G Base Station Controller, responsible for managing the finer details such as logical

```
network
  network country code 901
  mobile network code 70
  bts 0
    type sysmobts
    band GSM-1800
    location_area_code 23
    # This is the unit id that has to match
        the BTS configuration
    ip.access unit_id 1800 0
    codec-support fr hr amr
    gprs mode gprs
    gprs nsvc 0 remote ip 192.168.0.9
    gprs nsvc 0 remote udp port 23000
    gprs nsvc 0 local udp port 23000
    gprs nsvc 0 nsvci 1800
    gprs nsei 1800
    gprs cell bvci 1800
    trx 0
      rf_locked 0
      arfcn 868
      nominal power 23
      timeslot 0
        phys_chan_config CCCH+SDCCH4
      timeslot 1
        phys_chan_config SDCCH8
      timeslot 2
        phys_chan_config TCH/F_TCH/H_PDCH
      timeslot 3
        phys_chan_config TCH/F_TCH/H_PDCH
      timeslot 4
        phys_chan_config TCH/F_TCH/H_PDCH
      timeslot 5
        phys_chan_config TCH/F_TCH/H_PDCH
      timeslot 6
        phys_chan_config TCH/F_TCH/H_PDCH
      timeslot 7
        phys_chan_config PDCH
  e1_input
    e1_line 0 driver ipa
  msc 0
    mgw remote-ip 192.168.0.9
    mgw remote-port 12427
    allow-emergency deny
    codec-list hr3

  log stderr
    logging filter all 1
    logging print extended-timestamp 1
    logging print category 1
    logging print category-hex 0
    logging print level 1
    logging print file basename last
    logging level set-all info
```

FIGURE 27. **osmo-bsc.cfg for voice and data service [12].**

channels and lower-level functions for one or more 2G BTS. Essentially, the BSC communicates the mobile phones' intentions to the MSC. The MSC, in turn, instructs the BSC to establish channels, page mobile phones, and handle the maintenance tasks associated with the lower-level BTS operations.

To operate efficiently, OsmoBSC needs to register with OsmoSTP and establish contact with the MSC through its designated point code. Unless configured otherwise, it typically utilizes the default point code of OsmoMSC for this purpose. Moreover, OsmoBSC must connect with an OsmoMGW on its MGCP port to manage Real-time Transport Protocol (RTP) streams between BTS and the MSC's Media Gateway (MGW), as detailed earlier under "OsmoMGW".

Furthermore, the BSC necessitates comprehensive configuration information for all connected BTS. Typically, the BTS side takes care of configuring the physical attributes, unit ID, and the remote address of the BSC. Meanwhile, the BSC manages the rest of the configuration via the Operation and Maintenance Link (OML). The provided example in Fig. 27 outlines the configuration for a sysmoBTS.

Additionally, certain network properties must be established. The 'gprs mode' parameter determines whether packet-switched access is enabled. 'gprs mode none' deactivates data services, indicating that the OsmoBTS should not engage with osmo-pcu to establish data services. Notably, selecting 'gprs mode gprs' without a functioning Packet Control Unit (PCU) can lead to mobile phones frequently changing between BTS cells in their quest to establish GPRS services [12].

To allow data services, you can opt for 'gprs mode gprs' or 'gprs mode egprs' and configure the Physical Data Channel (PDCH) timeslots. Traditionally, a fixed number of TCH timeslots for voice and PDCH timeslots for data services are configured. OsmoBTS also supports dynamic timeslots, offering flexibility in channel combinations, as explained in the Abis manual, specifically in the chapter titled "Dynamic Channel Combinations." The configuration represented in Fig. 27 provides a voice-and-data service setup following Osmocom's dynamic timeslots style.

```
hnbgw
  iuh
    local-ip 192.168.0.9
  mgcp
    mgw remote-ip 192.168.0.9
    mgw remote-port 22427
  pfcp
   remote-addr 192.168.0.9

log stderr
  logging filter all 1
  logging print extended-timestamp 1
  logging print category 1
  logging print category-hex 0
  logging print level 1
  logging print file basename last
  logging level set-all info
```

FIGURE 28. **osmo-hnbgw.cfg [12].**

9) OsmoHNBGW

OsmoHNBGW is the 3G HomeNodeB Gateway, found in the osmo-iuh.git repository: it receives the Iuh protocol from a 3G femto cell, separates it into IuCS and IuPS and forwards to the MSC and SGSN [5].

OsmoHNBGW needs to connect to OsmoSTP for routing and needs to know the MSC and SGSN point codes [12]. If omitted, it assumes OsmoSTP on 127.0.0.1 and uses the point codes that are default in OsmoMSC and OsmoSGSN. It must also be reachable by the hNodeB, hence, its Iuh must typically run on a public IP, not a loopback address like 127.0.0.1 as shown in Fig. 28.

10) OsmoUPF

OsmoUPF serves as a local intermediary for the GTP user plane in 3G packet-switched networks. It's responsible

[5]In the case of the nano3G, the MSC necessitates encoding addresses in the X.213 style when performing RAB assignments. You can find the relevant configuration in the 'iu rab-assign-addr-enc x213' section of osmo-msc.cfg. To establish a connection between your femto cell and the HNBGW, you can refer to the guide titled "Configuring the ipaccess nano3G." The 'hnbap-allow-tmsi' option serves as a workaround for the nano3G, enabling it to pass a TMSI (Temporary Mobile Subscriber Identity) as the UE-Register identity. Normally, this identity should be an IMSI (International Mobile Subscriber Identity). It's important to note that utilizing a UPF as a GTP hop is an optional choice in this setup.

```
pfcp
  local-addr 192.168.0.9
nft
  # netfilter requires no specifc configuration
gtp
  # if encaps/decaps of GTP to "the internet" is required,
      uncomment the following line:
  #dev create apn0
log stderr
  logging filter all 1
  logging print extended-timestamp 1
  logging print category 1
  logging print category-hex 0
  logging print level 1
  logging print file basename last
  logging level set-all info
```

FIGURE 29. **osmo-upf.cfg [12].**

for receiving PFCP instructions from OsmoHNBGW and establishing connections between GTP tunnels on the hNodeB (Access) side and the GGSN (Core) side. Running OsmoUPF requires a Linux operating system. OsmoUPF leverages specific Linux kernel functionalities to manage the GTP user plane within the kernel space. This includes using the GTP module for encapsulating and decapsulating data to and from the internet and utilizing netfilter (nftables) for routing GTP tunnels between two interfaces. Please note that the netfilter features employed by OsmoUPF demand a Linux kernel version of at least 5.17 [12].

In order to use these kernel features, the osmo-upf program should be granted cap_net_admin permissions, which can be accomplished with the following command. If you've installed it from packages, this permission should typically be granted automatically:

sudo setcap cap_net_admin+pe /usr/bin/osmo-upf

The UPF paired with OsmoHNBGW will always do tunnel mapping, and never does encapsulation/decapsulation, which means that it does not require a GTP device as mentioned in Fig. 29.

```
#!/bin/sh
# usage: ./second_dhclient.sh eth0
dev="${1:-eth0}"
nr="$(ip a | grep "^[0-9]*: $dev" | wc -l)"
name="$(echo "$dev" | sed 's/[^0-9a-fA-F]//g' | head -c 1)"
mac="ac:ac:1a:b0:a0:$name$nr"
set -e -x
sudo ip link add link $dev address $mac $dev.$nr type macvlan
sudo dhclient $dev.$nr
ip addr show dev $dev.$nr
```

FIGURE 30. **Example of DHCP server [12].**

11) OsmoGGSN

OsmoGGSN, also known as the Gateway GPRS Support Node, plays a crucial role in 3G and 2G networks by facilitating the establishment of GTP tunnels from SGSNs to the internet uplink. In order to provide packet-switched

```
log stderr
  logging level all debug
  logging filter all 1
  logging print category 1
ggsn ggsn0
  gtp bind-ip 192.168.0.42
  apn internet
    tun-device apn0
    type-support v4
    ip dns 0 192.168.0.1
    ip dns 1 9.9.9.9
    ip prefix dynamic 192.168.42.0/24
    no shutdown
  default-apn internet
  no shutdown ggsn

log stderr
  logging filter all 1
  logging print extended-timestamp 1
  logging print category 1
  logging print category-hex 0
  logging print level 1
  logging print file basename last
  logging level set-all info
```

FIGURE 31. osmo-ggsn.cfg [12].

services, OsmoGGSN is required to offer GTP services to OsmoSGSN.

It's worth noting that OsmoGGSN and OsmoSGSN must employ identical GTP port numbers, as dictated by the GTP protocol. However, they should not operate on the same IP address. Additionally, for 2G networks, the SGSN must be accessible by the PCU, necessitating a public interface if the BTS is located in a separate box. Similarly, for 3G networks, the GGSN must be reachable by the hNodeB, requiring it to be situated on a public interface. To accommodate both scenarios, two public interfaces are essential. In example 30, the IP address 192.168.0.42 is used, assuming it's available on the local Ethernet interface [12].

Please consult your distribution's documentation for instructions on configuring a second IP address. As an alternative, you can configure a second address using your DHCP server as illustrated in Fig 30. For this illustration to work, the DCHP server would need to assign to you the address 192.168.0.42.

OsmoGGSN maintains a gsn_restart counter, which serves the purpose of reliably indicating to the SGSN when it has restarted. This counter is stored in the 'state-dir,' typically located in /tmp.

Furthermore, OsmoGGSN requires access to a tun device that possesses an address range available for subscribers' PDP contexts. This configuration can be set up in advance to avoid the need for root privileges during OsmoGGSN operation. If you choose to run it with 'sudo,' OsmoGGSN has the capability to create its own tun device. In the following example, the 'apn0' device has been established in advance using the following command:

sudo ip tuntap add dev apn0 mode tun user $USER group $USER
sudo ip addr add 192.168.42.0/24 dev apn0
sudo ip link set apn0 up

IPv4 operation is activated by default, although it's recommended to explicitly specify it for future compatibility. OsmoGGSN also specifies DNS servers and an IPv4 address range designated for assigning to subscribers' PDP contexts.

It's important to note that the APN mentioned in this configuration file, labeled as "internet" in this instance, must be set up on your phone. For this you have to refer to "APN for Data Service" in [12]. While the default-apn command can be used to handle any unknown APN names by defaulting to a predefined APN, it's still essential to configure some APN on your phone for it to attempt a connection to the data service.

A significant aspect of GGSN configuration is the network setup of your system as shown in Fig. 31, which should allow the routing of packets between the subscribers and your internet uplink. Detailed information on this can be found in the OsmoGGSN User Manual [6], specifically in the section labeled "Running OsmoGGSN / Routing."

```
sgsn
  gtp local-ip 192.168.0.9
  ggsn 0 remote-ip 192.168.0.42
  ggsn 0 gtp-version 1
  auth-policy remote
  gsup remote-ip 127.0.0.1
ns
  encapsulation udp local-ip 192.168.0.9
  encapsulation udp local-port 23000
  encapsulation framerelay-gre enabled 0

log stderr
  logging filter all 1
  logging print extended-timestamp 1
  logging print category 1
  logging print category-hex 0
  logging print level 1
  logging print file basename last
  logging level set-all info
```

FIGURE 32. osmo-sgsn.cfg [12].

12) OsmoSGSN

OsmoSGSN serves as the Serving GPRS Support Node, managing signaling tasks such as subscriber attachments and detachments, as well as PDP context handling for data services. To establish GTP tunnels for subscribers, OsmoSGSN requires connectivity to the GGSN. It's important to emphasize that it must have a distinct GTP IP address from OsmoGGSN, as previously mentioned. In the case of 2G networks, OsmoSGSN needs to be accessible by the PCU and necessitates a public IP for the Gb interface, especially when it is not directly co-located on the BTS hardware. For

[6]The configuration provided in Fig. 31 requires the apn0 tun device to be configured and up, as well as IP-forwarding and masquerading to be enabled, for this refer to the mentioned manual

2G operations, the SGSN and GGSN can both use local IP addresses for GTP, as long as these addresses differ (e.g., 127.0.0.1 and 127.0.0.2) [12].

In 3G networks, OsmoSGSN should be reachable by the HNBGW for IuPS. If your network solely involves 3G, the SGSN doesn't need to listen on a public IP address. Regarding 3G IuPS, the SGSN must register with OsmoSTP using a point code known to the HNBGW. If not explicitly configured, the default settings are employed, as detailed in the Point Codes section. Lastly, OsmoSGSN requires access to OsmoHLR for subscriber data. To use the HLR for subscriber authorization, set 'auth-policy remote' as shown in Fig. 32.

To employ an 'auth-policy remote,' it's essential to have the authentication tokens of the SIM cards stored in your OsmoHLR database. Alternatively, you can opt for 'auth-policy accept-all,' but keep in mind that this method is solely effective for 2G networks. For 3G networks, successful authentication is mandatory, and 'auth-policy remote' stands as the exclusive choice for a 3G SGSN.

13) OsmoBTS

OsmoBTS manages 2G radio hardware and is compatible with various hardware platforms, such as sysmoBTS and USRP. Alternatively, you can opt for BTS solutions from vendors like ip.access or Siemens that can seamlessly interact with OsmoBSC without the need for OsmoBTS's direct involvement.

```
phy 0
  instance 0
bts 0
  band 1800
  ipa unit-id 1800 0
  oml remote-ip 192.168.0.9
  trx 0
    phy 0 instance 0
```

FIGURE 33. Configuration for a sysmoBTS [12].

```
pcu
  flow-control-interval 10
  cs 2
  alloc-algorithm dynamic
  alpha 0
  gamma 0
```

FIGURE 34. Configuration for a OsmoPCU [12].

The choice of which BTS implementation to launch depends on the specific hardware in use as shown in Tables 2 and 3.

SDR based BTS like USRP, B210, umTRX [18]	Run osmo-trx and osmo-bts-trx on the machine hosting the SDR device
sysmoBTS device [19]	Run osmo-bts-sysmo and osmo-pcu on the sysmoBTS box itself
Other Osmocom based BTS [20]	Run the matching osmo-bts-* variant

TABLE 2. Matching BTS implementations Part 1

Third party BTS, like ip.access nanoBTS	Typically run in their own box, and you do not launch Osmocom software on them, but configure them to connect to your BSC
3G femto cell, like ip.access nano3G	This is not even a BTS but an hNodeB ("BTS" is a 2G term) – you configure a 3G femto cell to connect to OsmoHNBGW

TABLE 3. Matching BTS implementations Part 2

The BTS requires information about how to connect to OsmoBSC's Abis interface, and its unit id should align with one of the configured BTS unit ids within OsmoBSC.

Here's an illustrative configuration for a sysmoBTS in Fig. 33.

14) OsmoPCU

OsmoPCU manages the packet-switched component of 2G radio hardware. It's responsible for controlling the timeslots used for data transmission, a role that was traditionally handled by the BTS. OsmoPCU is commonly configured through the gprs settings within OsmoBSC. This configuration information is then relayed to the PCU using the OML protocol and OsmoBTS (through the PCU socket). An example configuration for OsmoPCU is shown in Fig. 34.

From here on forward, we will discuss about some operating examples of different osmocom programs. Every Osmocom program is bundled with a systemd service file. To ensure smooth operation, it's advisable to store configuration files in the /etc/osmocom/ directory and use systemd to initiate the individual components.

If you install Osmocom from Debian or OPKG feeds, you'll discover the systemd service files in the /lib/systemd/system/ directory. Re/starting and stopping then works like this:

systemctl restart osmo-hlr
systemctl stop osmo-hlr

For illustration, the manual command invocations for the components would look like this on a typical CNI standalone host:

- osmo-hlr -l hlr.db -c osmo-hlr.cfg
- osmo-msc -c osmo-msc.cfg
- osmo-mgw -c osmo-mgw-for-msc.cfg
- osmo-mgw -c osmo-mgw-for-bsc.cfg
- osmo-ggsn -c osmo-ggsn.cfg
- osmo-sgsn -c osmo-sgsn.cfg
- osmo-stp -c osmo-stp.cfg
- osmo-bsc -c osmo-bsc.cfg
- osmo-hnbgw -c osmo-hnbgw.cfg
- osmo-sip-connector -c osmo-sip-connector.cfg

There is this convenience launcher, which comprises of a useful **osmo-all** script to re/start or stop all components at once, edit to pick yours:

- #!/bin/sh
- cmd="${1:-status}"
- set -ex
- systemctl $cmd osmo-hlr osmo-msc osmo-mgw osmo-ggsn osmo-sgsn osmo-stp osmo-bsc osmo-hnbgw osmo-sip-connector

which provides

- ./osmo-all restart
- ./osmo-all status
- ./osmo-all stop

How to log in an Osmocom program?

Well its simple, because these programs share a unified logging system, which can be adjusted through the configuration files and the telnet VTY.

Starting from:

System Logging: Depending on the system's logging configuration, logs may by default be visible in /var/log/daemon.log, or by using journalctl:

- journalctl -f -u osmo-hlr

When journalctl is used, it may be necessary to enable it first, e.g. by setting "Storage=volatile" in /etc/systemd/journald.conf followed by a 'systemctl restart systemd-journald'; you may also need to 'systemctl unmask systemd-journald.service systemd-jounald.socket'. Logging will only start appearing for components that were restarted after these changes.

telnet VTY logging: A reliable method to access the logs is to establish a connection to the program's telnet VTY and activate logging within the VTY session. This approach ensures that you don't alter the application's default logging settings but rather generate a distinct logging destination specific to your telnet VTY session as shown in Fig. 35.

```
$ telnet localhost 4254
OsmoMSC> logging enable
OsmoMSC> logging level ?
   all    Global setting for all subsystems
   rll    A-bis Radio Link Layer (RLL)
   cc     Layer3 Call Control (CC)
   mm     Layer3 Mobility Management (MM)
   [...]
OsmoMSC> logging level all ?
everything debug      info      notice      error      fatal
OsmoMSC> logging level all debug
OsmoMSC> logging filter all 1
```

FIGURE 35. **Separate logging target [12].**

Logging output will promptly appear on your telnet console. It's important to remember that the VTY prompt remains active, allowing you to enter the 'logging filter all 0' command anytime to deactivate logging and interact with the system without the interference of ongoing log messages.

In Fig. 36 is a handy 'expect' script for attaching to osmo-* components by name and initializing logging while retaining access to a VTY prompt:

stderr logging: A common configuration you can add to any of the above configuration files to show all logging on stderr is shown in Fig. 37.

The filter all 1 switches on logging, read "do not discard all logging". The amount of logging seen is determined by **logging level ...** commands, here all categories are set to level debug, to show absolutely all logging. You will probably want to refine that.

IV. IMPORTANT SETUP REVIEW

The initial hurdle in establishing a mobile network is to clearly define your desired network configuration and identify the necessary components to accomplish it. For 2G support, there are various radio BTS equipment that work with Osmocom to choose from and various BTS models actually run Osmocom's OsmoBTS and OsmoPCU software on the BTS itself:

- sysmoBTS models 1002, 1002OD, 1020, 1200, 1100 and 2050 (osmo-bts-sysmo)
- SDR based BTSs, e.g. using the Ettus B200 or Fairwaves' UmTRX (osmo-trx plus osmo-bts-trx)
- Octasic OctBTS (osmo-bts-octphy)
- Nutaq Litecell 1.5 / sysmoBTS 2100 (osmo-bts-lc15)

However, there are also other "closed" BTS that are nevertheless interoperable with Osmocom's BSC implementation:

- ip.access NanoBTS
- Siemens BS11
- Ericsson RBS

Will you be connecting to a third-party MSC, such as the core "voice" component of a mobile operator's network? In that case, you should consider employing the standalone OpenBSC approach. Starting from 2017, Osmocom provides

```
#!/usr/bin/expect -f
set vty [lindex $argv 0]
set host localhost
switch $vty {
   hlr    { set port 4258 }
   bsc    { set port 4242 }
   mgw { set port 4243 }
   mgw2 {
              set host 127.0.0.2
              set port 4243
            }
 sg { set port 4245 }
 msc { set port 4254 }
 sip { set port 4256 }
 gg { set port 4260 }
 osmo-hlr { set port 4258 }
 osmo-bsc { set port 4242 }
 osmo-mgw { set port 4243 }
 osmo-mgw-for-bsc { set port 4243 }
 osmo-mgw-for-msc {
              set host 127.0.0.2
              set port 4243
          }
 osmo-sgsn { set port 4245 }
 osmo-msc { set port 4254 }
 osmo-sip-connector { set port 4256 }
 osmo-ggsn { set port 4260 }
 default { set port 4242 }
}
spawn telnet localhost $port
expect ">"
send "enable\r"
expect "#"
send "logging enable\r"
expect "#"
send "logging print category 1\r"
expect "#"
send "logging print category-hex 0\r"
expect "#"
send "logging print level 1\r"
expect "#"
send "logging print file basename last\r"
expect "#"
send "logging print extended-timestamp 1\r"
expect "#"
send "logging level set-all notice\r"
expect "#"
switch $vty {
  msc {
     send "logging level mm info\r"
     expect "#"
     send "logging level cc info\r"
     expect "#"
  }
}
send "logging filter all 1\r"
expect "#"
interact
```

FIGURE 36. vty script [12].

a genuine SCCP/M3UA-based 3GPP A-over-IP interface, which can be found in the latest osmo-bsc.git version. You can locate osmo-bsc in Binary_Packages or the sysmocom OpenEmbedded feeds (beginning with "201705"). From 2018 onwards, Osmocom has reintroduced support for an SCCPlite-based A-interface. It's essential to avoid using osmo-bsc-sccplite from the outdated openbsc.git repository,

```
log stderr
  logging filter all 1
  logging color 1
  logging print category 1
  logging timestamp 1
  logging print extended-timestamp 1
  logging level all debug
```

FIGURE 37. All logging on stderr [12].

as this code has been deprecated and left unmaintained for years [21].

V. CONCLUSION

The evolution of telecommunications from its inception to the cutting-edge 5G era has been a journey of astonishing technological progress. Today, as we stand in the year 2023, the horizon of telecommunications holds limitless possibilities. The rollout of 5G networks promises unprecedented speed, minimal latency, and the capacity to connect billions of devices simultaneously. These developments are reshaping industries, from healthcare and transportation to agriculture and smart cities, through the integration of edge computing, network slicing, and the IoT. Yet, the realm of IoT, with its unique demands and challenges, continues to present hurdles, particularly rooted in the constraints of internal device infrastructure. While 5G offers the potential for extensive IoT connectivity, it faces inherent limitations in this regard.

In response, this paper embarked on a journey to explore alternative pathways, ultimately converging on a pioneering concept: the fusion of 2G wireless technology with open source accessibility, specifically the Osmocom project. We began this expedition by retracing the origins of wireless communication, delving into the realm of 1G technology, which played a foundational role in the trajectory of wireless advancements. Continuing our journey, we ventured into the landscape of 2G and delved into the multifaceted world of IoT, examining how these technologies intersect to present innovative possibilities. This exploration culminated in the core theme of our paper: harnessing open source Osmocom technology to revolutionize the deployment of IoT applications.

The symbiotic relationship between open source solutions, Osmocom, and 2G technology has unveiled a new vision for the future of telecommunications and IoT. By embracing the power of open source innovation, these combined forces have the potential to overcome the intrinsic limitations of traditional IoT implementation. This paper has illuminated the profound potential of this novel approach, laying the foundation for further research, development,

and practical implementation in this space. As we journey forward into this promising future, where open source, 2G technology, IoT, and Osmocom converge, a world of boundless opportunities unfolds. It is a realm where connectivity knows no boundaries, and innovation thrives. This paper stands as a clarion call to all stakeholders in the telecommunications landscape to explore this uncharted territory. Together, we venture into a landscape of transformation, where Osmocom and open source are the keys to unlocking a world of possibilities.

REFERENCES

[1] S. Iqbal and J. M. Hamamreh, "A comprehensive tutorial on how to practically build and deploy 5G networks using open-source software and general-purpose, off-the-shelf hardware," RS Open Journal on Innovative Communication Technologies, vol. 2, no. 6, dec 16 2021.

[2] "1G wireless network technologies." Wikipedia, 2023, https://en.wikipedia.org/wiki/1G.

[3] Alexander, "First generation mobile phones." Netizzan, 2023, https://netizzan.com/mobile-network-generations/.

[4] Alexander, "Everything about second generation networks." Netizzan, 2023, https://netizzan.com/everything-about-second-generation-network/.

[5] A. S. Gillis, "Internet of things (IoT)." TechTarget, 2023, https://www.techtarget.com/iotagenda/definition/Internet-of-Things-IoT.

[6] O. Source, "What is open source?" Red Hat, 2023, https://opensource.com/resources/what-open-source.

[7] Wikipedia, "Open source." Wikimedia Foundation, Inc., 2023, https://en.wikipedia.org/wiki/Open_source.

[8] S. Ranger, "History of internet of things (IoT)." ZDNET, 2020, https://www.zdnet.com/article/what-is-the-internet-of-things-everything-you-need-to-know-about-the-iot-right-now/.

[9] "Iot market size." Fortune Business Insights, 2019,https://www.fortunebusinessinsights.com/industry-reports/internet-of-things-iot-market-100307.

[10] A. Simmons, "Internet of things (IoT) architecture: Layers explained." Dgtl Infra, 2022, https://dgtlinfra.com/internet-of-things-iot-architecture/#:~:text=IoT%20architecture%20is%20the%20structure,(e.g.%2C%20software)%20layers.

[11] J. Howell, "An introduction to iot architecture." 101 Blockchains, 2023, https://101blockchains.com/iot-architecture/.

[12] Laforge, "Cellular network infrastructure." Osmocom.org, 2023, https://osmocom.org/projects/cellular-infrastructure/wiki/Osmocom_Network_In_The_Box.

[13] T. Lavie, "Twenty-year-old 2G technology steps in to help large scale iot deployments in europe." Telit, 2018, https://www.telit.com/blog/2g-technology-for-iot-deployments-emea/.

[14] "Osmocom." Wikipedia, 2023, https://en.wikipedia.org/wiki/Osmocom.

[15] Laforge, "OsmoNITB migration guide." Osmocom.org, 2020, https://osmocom.org/projects/cellular-infrastructure/wiki/OsmoNITB_Migration_Guide.

[16] neels, "Osmocom network in the box, tar file." Osmocom.org, 2022, https://osmocom.org/attachments/5279.

[17] laforge, "Osmocom manuals." Osmocom.org, 2022, https://osmocom.org/projects/cellular-infrastructure/wiki/Osmocom_Manuals.

[18] ——, "Osmotrx." Osmocom.org, 2023,https://osmocom.org/projects/osmotrx/wiki/OsmoTRX#RF-Hardware-support.

[19] "Cellular base stations." sysmocom, 2023,https://www.sysmocom.de/products/bts/.

[20] laforge, "Osmobts." sysmocom, 2022,https://osmocom.org/projects/osmobts/wiki/Wiki#Backends-Hardware-support.

[21] ——, "Configuration guide." osmocom.org, 2022,https://osmocom.org/projects/cellular-infrastructure/wiki/Configuration_Guide.