Resource Management for Edge Computing in Internet of Things (IoT)

Zur Erlangung des akademischen Grades eines
Doktors der Ingenieurwissenschaften

von der Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte
Dissertation

von
Farzad Samie

Tag der mündlichen Prüfung:	26. Januar 2018
Referent:	Prof. Dr. Jörg Henkel
Korreferent:	Prof. Dr. Dimitrios Soudris

Farzad Samie
Bettina-von-Arnim-Weg, 7
76135, Karlsruhe

Hiermit erkläre ich an Eides statt, dass ich die von mir vorgelegte Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen, Internet-Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen – die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

_____

Farzad Samie

To:

Whom may it concern! (Well, as a joke).

&

My family, especially my Mom.

Acknowledgment

First of all, I want to thank my advisor professor Jörg Henkel for giving me the opportunity to join his research group and pursue my Ph.D. under his supervision. He believed in me, motivated me and supported me during the course of Ph.D. study.

I also want to thank my co-advisor, professor Dimitrios Soudris from National Technical University of Athens (NTUA), Greece. I always enjoyed and learned from our conversations.

I would also like to express my high gratitude to my technical supervisor, Dr. Lars Bauer, from whom I learned a lot. He was an incredible support, even sometimes without me knowing it.

Vasileios Tsoutsouras, the Ph.D. student of prof. Soudris has contributed to my papers. I really enjoyed working with him as he has been always positive and helpful. He was a huge help when I visited NTUA in Greece.

In the first few years of my Ph.D. study, I had three teammates Chih-Ming Hsieh, Sammer Srouji, and Manyi Wang. I would like to thank them for the valuable (and sometimes fun) discussions and all the good times.

Living in a foreign country is not always easy. But having awesome friends made it a lot easier for me. I would like to thank my Iranian friends who have been my support since my first day in Germany. I appreciate their valuable advices, geniality and unconditional assistance that helped me overcome many difficulties.

I enjoyed the work environment and atmosphere of our institute. The positive and friendly attitude of my colleagues from CES, CDNC, and CAPP created such a constructive workplace for which I would like to thank them.

I supervised multiple students during my Ph.D. study (some of them are listed in Page vi). Their work contributed to this thesis in one way or the other. I stand on their shoulders. I appreciate all the effort and hard work that they put in.

Finally, I want to dedicate my thesis to my family, especially my Mom whose devotion and sacrifices I can never compensate.

Karlsruhe, im January 2018 *Farzad Samie*

List of Own Publications:

– Farzad Samie, Sebastian Paul, Lars Bauer, and Jörg Henkel. "Highly Efficient and Accurate Seizure Prediction on Constrained IoT Systems", in *IEEE/ACM Design, Automation and Test in Europe Conference (DATE'18)*, 2018.

– Farzad Samie, Vasileios Tsoutsouras, Lars Bauer, Sotirios Xydis, Dimitrios Soudris, Jörg Henkel. "Distributed Trade-based Edge Device Management in Multi-gateway IoT", *ACM Transactions on Cyber-Physical Systems (TCPS), Special Issue on Internet of Things (IoT)*, 2017.

– Farzad Samie, Vasileios Tsoutsouras, Dimosthenis Masouros, Lars Bauer, Dimitrios Soudris, Jörg Henkel. "Fast Operation Mode Selection for Highly Efficient IoT Edge Devices", *ACM Transactions on Cyber-Physical Systems (TCPS), (under review)*, 2017.

– Farzad Samie, Vasileios Tsoutsouras, Lars Bauer, Sotirios Xydis, Dimitrios Soudris, Jörg Henkel. "Oops: Optimizing Operation-mode Selection for IoT Edge Devices", *ACM Transactions on Internet Technology (TOIT), Special Section on Fog, Edge, and Cloud Integration for Smart Environments*, (under review) 2017.

– Jörg Henkel, Santiago Pagani, Hussam Amrouch, Lars Bauer, Farzad Samie. "Ultra-Low Power and Dependability for IoT Devices" (Invited Paper for IoT Technologies) in *Design, Automation and Test in Europe Conference (DATE'17)*, 2017.

– Farzad Samie, Vasileios Tsoutsouras, Lars Bauer, Sotirios Xydis, Dimitrios Soudris, Jörg Henkel. "Computation Offloading Management and Resource Allocation for Low-power IoT Edge Devices" in *IEEE World Forum on Internet of Things (WF-IoT)*, 2016.

– Farzad Samie, Lars Bauer, Jörg Henkel. "IoT Technologies for Embedded Computing: A Survey" in *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2016.

– Farzad Samie, Vasileios Tsoutsouras, Sotirios Xydis, Lars Bauer, Dimitrios Soudris, Jörg Henkel. "Distributed QoS Management for Internet of Things under Resource Constraints" in *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2016.

– Farzad Samie, Lars Bauer, Jörg Henkel. "An Approximate Compressor for Wearable Biomedical Healthcare Monitoring Systems" in *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2015.

– Farzad Samie, Lars Bauer, Chih-Ming Hsieh, Jörg Henkel. "Online Binding of Applications to Multiple Clock Domains in Shared FPGA-based Systems" in *Design, Automation and Test in Europe Conference (DATE'15)*, 2015.

– Chih-Ming Hsieh, Farzad Samie, M. Sammer Srouji, Manyi Wang, Zhonglei Wang, Jörg Henkel. “Hardware/Software Co-design for A Wireless Sensor Network Platform” in *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS'14)*, 2014.

List of Selected Supervised Student Works:

– Shivani Choudhary, “Optimization and Hardware Implementation of IoT Applications on PILPino Platform”, Bachelor thesis, 2017.

– Axel Gallus, “A Simulation Framework for IoT systems”, Diploma thesis, 2017.

– Muhammad Saaduddin Siddiqui, “Study the performance of Bluetooth Low Energy”, Master Thesis, 2017.

– Sebastian Paul, “Computation Offloading in Edge Computing for Internet of Things: A Case Study of Epileptic Seizure Prediction”, Master thesis, 2017.

– Nick Vlasoff, “Design and Implementation of an IoT-based Smart Shoe for Gait Analysis”, Bachelor thesis, 2017.

– Oleg Schell, “ECG Feature Extraction on Low-Power IoT Devices”, Bachelor thesis, 2016.

– Timo Kegelmann, “FPGA Platform for IoT: An Energy Efficiency Evaluation”, Bachelor thesis, 2016.

– Jonathan Knam, “An IoT-based Wearable Solution for Physical Activity Monitoring”, Master thesis, 2016.

– David Barahona Pereira, “Evaluation of Feature Extraction Techniques for an Internet of Things Electroencephalogram”, Bachelor thesis, 2016.

– Moises Araya, “Design and implementation of a IoT-based portable ECG monitoring device”, Bachelor thesis, 2015.

Abstract

The massive number of Internet of Thing (IoT) devices and their continuous data collection will lead to a rapid increase in the scale of collected data. Processing all these collected data with central cloud servers is inefficient, and even sometimes is unfeasible or unnecessary. Hence, the task of processing the data is pushed to the network edges introducing the concept of Edge Computing. Processing the information closer to the source of data (e.g. on the gateways and edge devices) not only reduces the huge workload of central servers and network, also decreases the latency for the real-time applications by avoiding the unreliable and unpredictable network latency to communicate with the cloud servers.

Current art in IoT architectures utilizes gateways to enable application-specific connectivity for IoT devices. In typical configurations, IoT gateways are shared among several IoT edge devices. Given the limited available bandwidth and processing capabilities of an IoT gateway, the service quality (SQ) of connected IoT edge devices must be adjusted over time not only to fulfill the needs of individual IoT device users, but also to tolerate the SQ needs of the other IoT edge devices sharing the same gateway.

This thesis, first, investigates essential technologies for IoT and its existing trends, provides distinguishing properties of IoT for embedded system domain in addition to a comprehensive IoT perspective for embedded systems. Several applications in the healthcare domain are studied and implemented to help to derive a model for their data processing software. The application model helps in identifying multiple operation modes. IoT systems anticipate the edge devices to support different operation modes to be able to adapt to varying runtime situations, like preserving energy at low battery while still maintaining some crucial functionalities, increasing the service quality upon user's request, etc. These modes are either because of different offloading schemes (e.g. raw data transmission, partially process on IoT device, or transmitting the final result), or due to different service qualities. Operation modes differ in terms of resource usage, both on the device (e.g. energy consumption) and on the gateway (e.g. communication bandwidth, processing power, memory, etc.). Selecting the optimal operation mode for edge devices is a challenge given the limited resources on the edge of the network (e.g. bandwidth and processing power of the shared gateway), diverse constraints on the IoT edge devices (e.g. battery lifetime, quality of service, etc.), and runtime dynamics of the IoT edge infrastructure. This thesis proposes and presents fast and efficient operation mode selection techniques.

Moving to multi-gateway IoT systems, where some IoT devices are reachable by more than one gateway, managing the shared resources and selecting the operation mode of IoT devices become much more complex. This thesis also presents a distributed trade-based device management mechanism for multi-gateway IoT systems. This mechanism targets the joint problem

of binding (i.e. determines the gateway for each IoT device) and allocation (i.e. determines the allocated resources to each device). Starting from an initial setup, gateways negotiate and communicate with each other, migrate or exchange IoT devices when it increases the system's overall benefit.

This thesis also presents application-specific optimizations for individual IoT devices. Three applications in the health monitoring domain are implemented and investigated for wearable IoT devices. It also proposes a novel approximate compression technique for IoT applications that process bio-signals for health monitoring. This technique reduces the amount of data transmission from IoT device which reduces the resource usage both on the device and on the shared gateway.

To evaluate the proposed techniques and mechanisms, some applications are profiled on the IoT platforms to measure their required parameters including execution time and resource utilization. These parameters are then used in a framework that models the IoT network and captures the interaction between devices and gateway(s) and measures the communication overhead as well as the achieved battery lifetime and service quality of devices. The operation model selection algorithms are implemented on IoT platforms to measure their overheads in terms of execution time and memory usage.

Kurzfassung

Die große Anzahl an Geräten im Internet der Dinge (IoT) und deren kontinuierliche Datensammlungen führen zu einem rapiden Wachstum der gesammelten Datenmenge. Die Daten komplett mittels zentraler Cloud Server zu verarbeiten ist ineffizient und zum Teil sogar unmöglich oder unnötig. Darum wird die Datenverarbeitung an den Rand des Netzwerks verschoben, was zu den Konzepten des Edge Computings geführt hat. Informationsverarbeitung nahe an der Datenquelle (z.B. auf Gateways und Edge Geräten) reduziert nicht nur die hohe Arbeitslast zentraler Server und Netzwerke, sondern verringer auch die Latenz für Echtzeitanwendungen, da die potentiell unzuverlässige Kommunikation zu Cloud Servern mit ihrer unvorhersehbaren Netzwerklatenz vermieden wird. Aktuelle IoT Architekturen verwenden Gateways, um anwendungsspezifische Verbindungen zu IoT Geräten herzustellen. In typischen Konfigurationen teilen sich mehrere IoT Edge Geräte ein IoT Gateway. Wegen der begrenzten verfügbaren Bandbreite und Rechenkapazität eines IoT Gateways muss die Servicequalität (SQ) der verbundenen IoT Edge Geräte über die Zeit angepasst werden. Nicht nur um die Anforderungen der einzelnen Nutzer der IoT Geräte zu erfüllen, sondern auch um die SQ-Bedürfnisse der anderen IoT Edge Geräte desselben Gateways zu tolerieren.

Diese Arbeit untersucht zuerst essentielle Technologien für IoT und existierende Trends. Dabei werden charakteristische Eigenschaften von IoT für die Embedded Domäne, sowie eine umfassende IoT Perspektive für Eingebettete Systeme vorgestellt. Mehrere Anwendungen aus dem Gesundheitsbereich werden untersucht und implementiert, um ein Model für deren Datenverarbeitungssoftware abzuleiten. Dieses Anwendungsmodell hilft bei der Identifikation verschiedener Betriebsmodi. IoT Systeme erwarten von den Edge Geräten, dass sie mehrere Betriebsmodi unterstützen, um sich während des Betriebs an wechselnde Szenarien anpassen zu können. Z.B. Energiesparmodi bei geringen Batteriereserven trotz gleichzeitiger Aufrechterhaltung der kritischen Funktionalität oder einen Modus, um die Servicequalität auf Wunsch des Nutzers zu erhöhen etc. Diese Modi verwenden entweder verschiedene Auslagerungsschemata (z.B. die übertragung von Rohdaten, von partiell bearbeiteten Daten, oder nur des finalen Ergebnisses) oder verschiedene Servicequalitäten. Betriebsmodi unterscheiden sich in ihren Ressourcenanforderungen sowohl auf dem Gerät (z.B. Energieverbrauch), wie auch auf dem Gateway (z.B. Kommunikationsbandbreite, Rechenleistung, Speicher etc.). Die Auswahl des besten Betriebsmodus für Edge Geräte ist eine Herausforderung in Anbetracht der begrenzten Ressourcen am Rand des Netzwerks (z.B. Bandbreite und Rechenleistung des gemeinsamen Gateways), diverser Randbedingungen der IoT Edge Geräte (z.B. Batterielaufzeit, Servicequalität etc.) und der Laufzeitvariabilität am Rand der IoT Infrastruktur. In dieser Arbeit werden schnelle und effiziente Auswahltechniken für Betriebsmodi entwickelt und präsentiert.

Wenn sich IoT Geräte in der Reichweite mehrerer Gateways befinden, ist die Verwaltung der gemeinsamen Ressourcen und die Auswahl der Betriebsmodi für die IoT Geräte sogar noch komplexer. In dieser Arbeit wird ein verteilter handelsorientierter Geräteverwaltungsmechanismus für IoT Systeme mit mehreren Gateways präsentiert. Dieser Mechanismus zielt auf das kombinierte Problem des Bindens (d.h. ein Gateway für jedes IoT Gerät bestimmen) und der Allokation (d.h. die zugewiesenen Ressourcen für jedes Gerät bestimmen) ab. Beginnend mit einer initialen Konfiguration verhandeln und kommunizieren die Gateways miteinander und migrieren IoT Geräte zwischen den Gateways, wenn es den Nutzen für das Gesamtsystem erhöht.

In dieser Arbeit werden auch anwendungsspezifische Optimierungen für IoT Geräte vorgestellt. Drei Anwendungen für den Gesundheitsbereich wurden realisiert und für tragbare IoT Geräte untersucht. Es wird auch eine neuartige Kompressionsmethode vorgestellt, die speziell für IoT Anwendungen geeignet ist, die Bio-Signale für Gesundheitsüberwachungen verarbeiten. Diese Technik reduziert die zu übertragende Datenmenge des IoT Gerätes, wodurch die Ressourcenauslastung auf dem Gerät und dem gemeinsamen Gateway reduziert wird.

Um die vorgeschlagenen Techniken und Mechanismen zu evaluieren, wurden einige Anwendungen auf IoT Plattformen untersucht, um ihre Parameter, wie die Ausführungszeit und Ressourcennutzung, zu bestimmen. Diese Parameter wurden dann in einem Rahmenwerk verwendet, welches das IoT Netzwerk modelliert, die Interaktion zwischen Geräten und Gateway erfasst und den Kommunikationsoverhead sowie die erreichte Batterielebenszeit und Servicequalität der Geräte misst. Die Algorithmen zur Auswahl der Betriebsmodi wurden zusätzlich auf IoT Plattformen implementiert, um ihre Overheads bzgl. Ausführungszeit und Speicherverbrauch zu messen.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

Recent advances in technologies of sensors, wireless communication, and embedded processors have enabled the design of small-size low-power and low-cost devices that can be networked or connected to the Internet. These are the key components of the emerging paradigm of Internet of things (IoT) [SBH16, AIM10]. IoT devices use the network, or particularly the Internet, as an infrastructure for connecting to each other to communicate, coordinate and cooperate in order to offer advanced control and monitoring services. IoT is covering an ever-increasing range of applications, such as healthcare monitoring, smart home, smart building, smart city, smart industry (also known as *Industry 4.0* in Germany), etc. [SBH16].

Figure 1.1 shows the estimated number of IoT devices by the year 2020 in different sectors. Consumer sector includes the devices which are used by the end user which include tracking and fitness bands, healthcare devices, etc. The cross-industry sector includes the usual and general devices that are being used in different industries including smart home, smart parking, smart city, etc. The last sector is industry specific which includes special devices and infrastructures used in factories to increase the efficiency of assembly lines, quality assurances, etc. The total number of installed IoT devices is predicted to be more than 20 billion by 2020 [gar]. There are other forecasts announced by other companies including International Data Corporation (IDC), Ericsson and Cisco. Even though the predicted numbers are different, but they all anticipate a massive number of connected IoT devices.

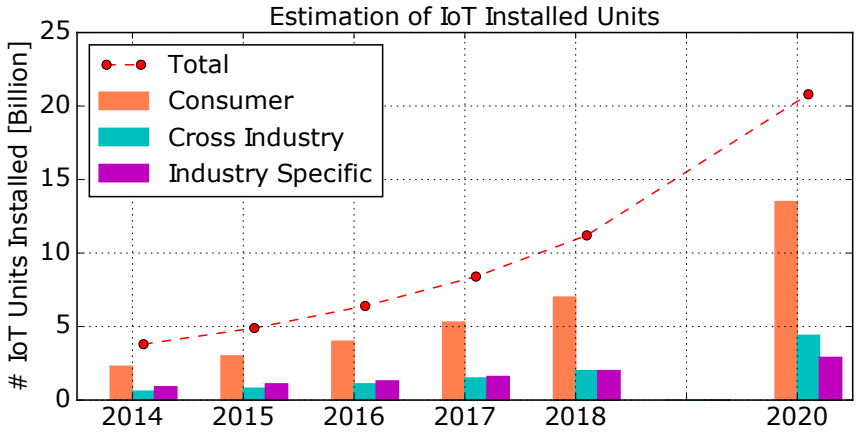


Figure 1.1: Estimated number of IoT devices in different sectors by Gartner [gar]

Figure 1.2 shows the components of the IoT ecosystem, including embedded devices, analytics, networks, etc. [BI]. IoT devices are interacting with the physical world using sensors and/or actuators to monitor and/or control the desired parameter. The gateway interfaces the IoT local networks with the Internet. The gateway bridges the networks, aggregates the collected data and even offers processing service. Cloud servers or cloudlets provide analysis and storage services.

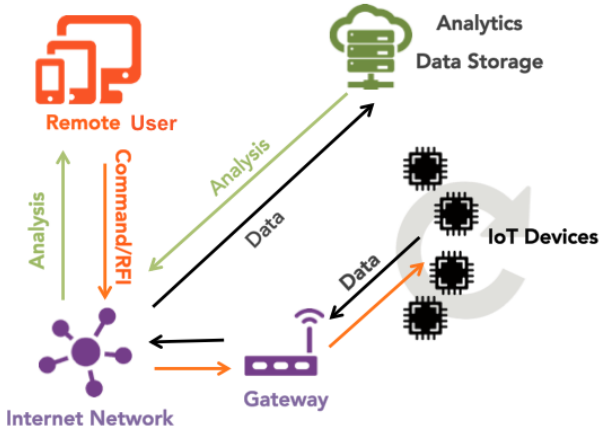


Figure 1.2: Components of the IoT ecosystem introduced by *Business Insider* [BI]

Figure 1.3 shows the hierarchical layers of computation in an IoT system [SBH16]. As we move to the higher levels (i.e. from edge devices to the cloud servers), the processing capability increases. However, the latency would increase due to two factors: 1) network delay and 2) more workload on the servers. Therefore, the predictability of the real-time properties would decrease.

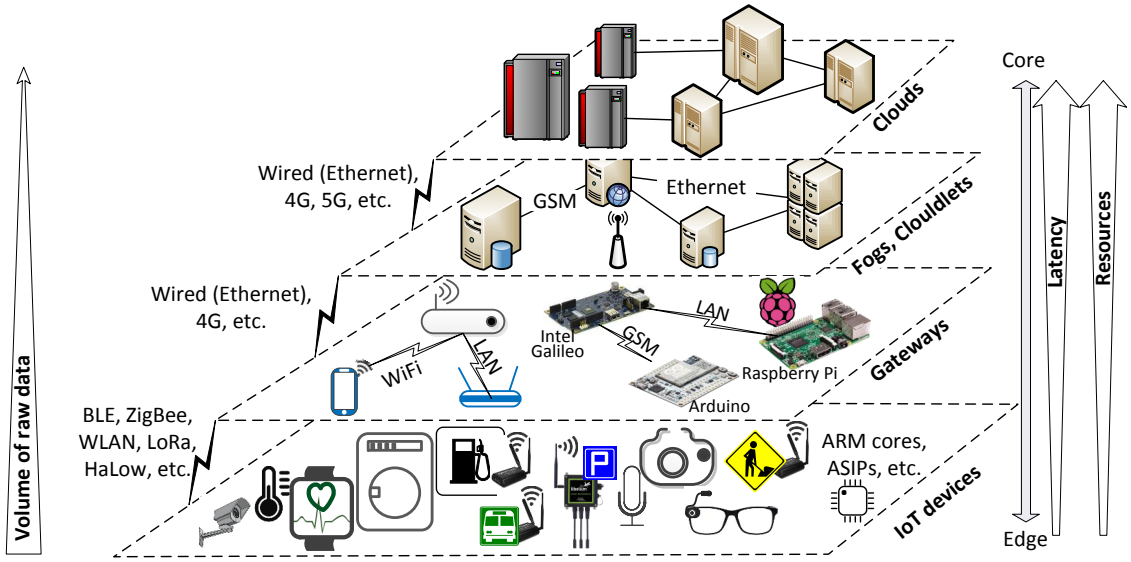


Figure 1.3: Computation layers in IoT systems and their properties

The large number of IoT edge devices, as well as their long-term and continuous data collection, will lead to a hard-to-manage amount of acquired data [CCT+15] which brings new challenges

into the embedded world. One of the challenges in IoT is to process and analyze a huge amount of data from heterogeneous devices. This challenge has two aspects: 1) the large volume of data which is also known as Big Data [LL15] and 2) diverse application requirements of IoT [ZMK+15]. Handling all these collected data with central cloud servers is inefficient, and even sometimes is unfeasible [ZMK+15], because of:

- the overall energy and cost to transmit large amount of data,
- unreliable and high latency of the network caused by the high workload to transmit [WSJ15, SEKC15].

Edge Computing (EC) is a promising solution to address this issue [Rap16, SCZ+16]. In EC the task of data processing is pushed to the edge of the IoT network (comprising gateways and edge devices) close to where the data is collected [SEKC15, Rap16]. According to IDC Futurescape [Rap16], more than 40% of IoT-generated data is predicted to be processed, stored, and acted upon close to the network edge. This approach reduces the application latency (and increase the reliability of the application), and reduces the load on the IoT network by distributing the computation. However, it brings new challenges and problems such as resource management and allocation that are needed to be addressed.

1.1.1 Limited Resources on the Edge of IoT system

Many IoT devices are battery-operated or have limited energy sources due to either their portability requirements or the lower cost of installation, deployment and maintenance [HPA+17]. This limitation imposes the following two constraints on the edge devices: 1) processing capability, 2) communication bandwidth.

– Processing Capability Constraint: Low-power and long battery lifetime requirements do allow IoT devices to utilize high-performance processor or microcontrollers. IoT devices feature microcontrollers or processors with limited memory and performance. Due to this constraint, some of the IoT applications are not able to autonomously process the whole collected data at the edge devices. Hence, a portion of the processing task must be *offloaded* to the more powerful layers (e.g. gateways, Fogs or cloudlets) [BMB+16, BBC+14]. However, to offload the computation, the raw data or partially-processed data must be transmitted to the gateway, and this is where the bandwidth constraint comes into play.

– Bandwidth Constraint: Limited energy resources on IoT edge nodes mandate low-power wireless technologies. One of the main limitations of low power wireless technologies is their low data rate and throughput. Despite the recent and rapid enhancements in these wireless technologies and protocols, the throughput is still low. Table 1.1 summarizes the throughput of the most popular low-power IoT wireless technologies including Bluetooth low energy (BLE), ANT+, ZigBee, low power WiFi (also known as HaLow), Low Power Wide Area (LPWAN) technologies which include LoRa, SigFox, etc. [MM+16]. It is worth mentioning that

the throughput may decrease further in case of interference with other surrounding wireless radios [SBH16].

Table 1.1: Throughput of IoT wireless technologies [Smi11, Mulb]

Wireless	BLE	ANT+	ZigBee	HaLow	LoRa	SigFox
Throughput [Kbps]	270	20	120	150	50	<10

IoT envisions a model in which IoT edge nodes are connected to the Internet through gateways, as illustrated in Figures 1.2 and 1.3. A gateway (i) enables seamless integration of low-power wireless networks of IoT devices with other networks (e.g. cellular network, LAN, etc. [ZWC+10]) and (ii) provides local data processing service at the edge of the network [GRW+15] in the EC paradigm. Although the gateway might exploit a high-bandwidth connection to the Internet (i.e. cellular or WiFi), its interface with IoT edge nodes is still a low-power wireless connection such as LoRa, BLE or ZigBee, which have a low bandwidth (see Figure 1.3 and Table 1.1). The bandwidth demand of IoT edge nodes can be reduced by novel compression techniques [BMB+16, SBH15] or service quality adaptation [STX+16b]. However, the most effective approach would be processing the data partially on the IoT devices and offload the rest of computation to the gateway or a more resourceful device.

Moreover, IoT gateways are located in the vicinity of IoT devices, and therefore, require a small form-factor (e.g. they cannot afford to deploy high-performance processors due to cooling requirements). For instance, Intel IoT gateways (DK50 and DK100 series) are based on Intel Quark SoC X1000 and X1020D with only 400 MHz operating frequency [int]. Texas Instrument's IoT gateway TM4C129 is based on an ARM Cortex-M4F microcontroller unit running at 120 MHz [Fol15b, Fol15a]. To reduce the cost and deployment effort, some IoT gateways are battery-operated [BBM+15, Sch17]. Therefore, the processing power of gateways that is available to edge processing is constrained [Sch17].

The scarce gateway resources (i.e. processing power, memory, communication bandwidth) are shared between multiple IoT devices. The limited resources on the edge of the network (e.g. bandwidth, memory and processing power of the shared gateway), diverse constraints on the IoT edge devices (e.g. battery lifetime, quality of service, etc.), and the runtime dynamics of the IoT edge infrastructure introduce the resource management as a challenge in IoT. Due to the scalability and reliability issues especially in edge computing paradigm, the IoT local networks must be self-organizing and self-supported [MSDPC12, GZY+13] and not dependent on the cloud [ZMK+15]. Therefore, the resource management of IoT edge devices also needs to be handled at the edge, on the gateway. Resource management needs to be fast and efficient at runtime to be both responsive and low-overhead.

1.2 Thesis Contribution

The aim of this thesis is to study and investigate the challenges and opportunities of IoT from an embedded perspective and improve the efficiency of edge computing and address the efficient resource management for edge computing.

In particular, the contributions of this thesis are as follows:

- Leveraging the pipeline structure of the application, IoT devices can benefit from multiple computation offloading schemes well as multiple service quality levels. This introduces the operation modes for the IoT device. This thesis uses the operation mode of devices as a control parameter to respond to dynamic and varying runtime situations. The operation modes differ in terms of resource usage both on the IoT devices and on the shared gateway. Fast of efficient mode selection techniques are proposed in this thesis to allocate the constrained resources at the edge of the network more efficiently while respecting the requirements of the IoT devices, their applications and the resources that are shared between IoT devices.
- This work presents a distributed trade-based mechanism to manage the IoT devices in a multi-gateway system. Having multiple gateways gives some IoT devices more than one option for connecting to a gateway. The binding problem (determining the gateway for each IoT device) is addressed jointly with allocation problem (determining the allocated resource to each IoT device). The proposed mechanism starts with an initial setup and step-by-step improves the overall service quality by migrating or exchanging IoT devices between gateways.
- This thesis presents application-specific optimizations considering individual IoT devices. It studies the IoT applications whose input data is in form of the signal such as health monitoring applications that require bio-signals as their input. Three applications are implemented on IoT platforms and investigated to illustrate the data processing steps in such applications. These optimizations reduce the resource usage of IoT devices which consequently diminish the resource contention. It derives a general model for the structure of IoT applications and their sequence of pipeline stages. Then, it presents a novel approximate compression technique for bio-signals on wearable IoT devices which reduces the amount of data transmission.
- This thesis provides an overview of IoT technologies required from an embedded design perspective and specific properties associated with IoT in embedded systems' landscape. It investigates essential technologies for development of IoT systems, existing trends, and its distinguishing properties. By discussing the key characteristics, main application domains, and major research issues in IoT, this thesis provides a comprehensive IoT perspective for embedded system design. It categorizes IoT applications and devices based on different criteria and parameters and highlights the associated properties with each category. The suitability of different technologies for IoT is investigated, too.

1.3 Thesis Outline

The remainder of this thesis is structured as follows:

- Chapter 2 provides background and detailed overview of the related work and state-of-the-art. First, the hardware and software architecture of IoT embedded devices are discussed. Then, an overview of edge computing and computation offloading is presented. Eventually, the background and state-of-the-art in mode selection and resource management of IoT systems are discussed.
- Chapter 3 provides essential knowledge on IoT applications and their properties, characteristics, and requirements. It also presents IoT technologies required from an embedded design perspective and specific properties associated with IoT in embedded systems' landscape. Essential technologies for the development of IoT systems, existing trends, and its distinguishing properties are investigated. By discussing the key characteristics, main application domains, and major research issues in IoT, this chapter provides a comprehensive IoT perspective for embedded system design and edge devices.
- Chapter 4 presents the main contributions of this thesis. It describes the proposed techniques for resource management and efficient operation mode selection for the IoT device at the edge of the network. First, a model has been presented in Section 4.1 according to the data processing structure of IoT applications in health monitoring domain. It also discusses the computation offloading levels, service quality levels in IoT applications and introduces the operation modes for IoT applications, accordingly. Then, fast and low-overhead techniques are presented in Sections 4.2 and 4.3 to determine the operation mode of devices that share multiple resources of the gateway at runtime. The experimental results are presented at the end of each section. Section 4.4 presents a distributed trade-based mechanism for management of devices and resources in a multi-gateway IoT system.
- Chapter 5 presents some application-specific optimizations. It studies several IoT applications in the healthcare domain whose input data is in form of signal. Sections 5.1 to 5.3 describe EEG processing, ECG processing, and physical activity monitoring applications, respectively. The structure of their data processing software and their pipeline stages are studied. Then Section 5.4 presents the novel approximate compressor for bio-signal data.
- Chapter 6 concludes the thesis with a summary of the contributions and an outlook for the future extensions to this work. Section A presents the practical experiment to study some properties of Bluetooth Low Energy which are used in other experiments as input parameters. It also describes the designed and developed IoT prototypes in the scope of this thesis.

2 Background and Related Work

This chapter provides an overview and background for the following technical chapters and presents related work. This thesis envisions a dynamic IoT system that benefits from computation offloading, resource allocation, and operation adaptation to enable the data processing at the edge of the network. An overview of IoT architecture from embedded point of view is presented in Section 2.1. Both hardware and software architecture are analyzed, and for each, several opportunities for efficiency improvement are discussed. Section 2.2 discusses the concept of edge computing and computation offloading and their related work in the literature. It presents the achievements as well as the shortcomings and the current challenges, especially when they meet the emerging paradigm of IoT. Section 2.3 presents the related work on managing the shared resources using efficient allocation techniques. This chapter contains the background that is needed for the proposed resource management techniques in the following technical chapters. The rest of background information that is required for the discussions on IoT technologies and enablers can be found in the next chapter.

2.1 IoT Architecture

2.1.1 Hardware & Software Architecture of IoT Embedded Device

The general operation stages of an IoT application include 1) data acquisition, 2) data processing, 3) data storage, and 4) data transmission. The first and last stages exist on every application, while the processing and storage may or may not exist in some applications (see Figure 2.1).

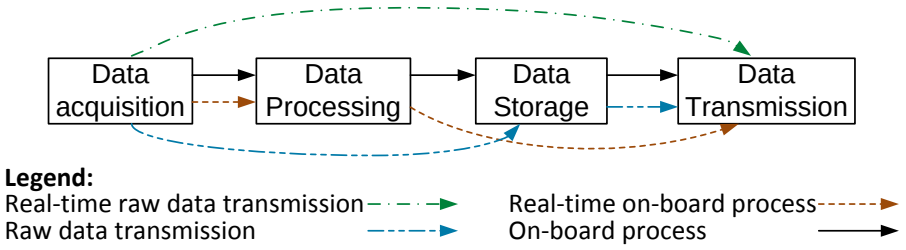


Figure 2.1: General stages of IoT applications

Figure 2.2 shows a general architecture of the main components of an IoT SoC platform [Kel14]. An IoT embedded device has many –if not most– of these components, e.g. at least one RF component for the connectivity.

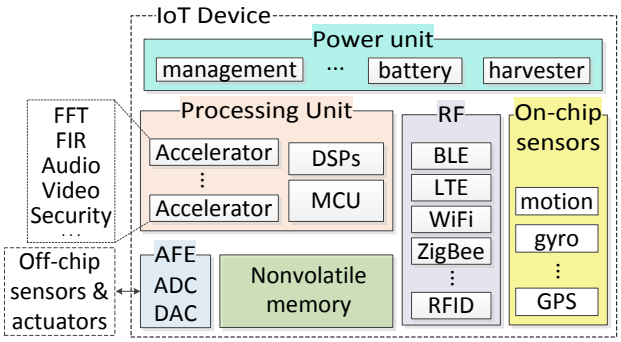


Figure 2.2: General architecture of an IoT embedded device

Efficiency at Different Stages

- **Data acquisition**:
  At hardware level: Low power integrated and on-chip sensors with Micro-Electro-Mechanical Systems (MEMS) technology can reduce the energy consumption of data acquisition. On-chip accelerometers, gyroscopes, microphones and biosensors are some examples.
  At software level: Energy efficient sensing schemes should effectively exploit spatial and temporal characteristics of the input data, in order to collect as less samples as possible while the required signal quality is retained [GBMP13]. The reduced amount of input data (depends on number of samples and resolution) affects the energy for transmission and storage, too. Compressed sensing (CS) [LDXW13] is a novel techniques in which the signal can be reconstructed from much fewer samples than Nyquist theory, at the cost of accuracy loss. As long as the input data has the sparseness property, a smaller number of samples can capture the required information, and CS can reduce the volume of collected data without significant loss of information.
  Many IoT applications have the data sparsity property and can exploit the CS paradigm. In health monitoring applications and wireless body sensor network, CS has been investigated and studied extensively.
- **Data processing**:
  At hardware level: Energy efficiency in a processing unit can be achieved by 1) ultra-low power processors [MZL+15] and 2) efficiently customized co-processors [KC11]. In [WLS+15], a heterogeneous dual-core processor is proposed and fabricated based on the big.LITTLE architecture. An ultra low power near-threshold processor alongside with a high performance processor in addition to a task scheduling framework brings energy efficiency for IoT applications.
- **Data storage**:
  At hardware level: Energy reduction in memory has received significant industrial and academic attention in embedded system design community [UAY+15], but there are some characteristics specific to IoT applications that can be exploited for further improvements in energy efficiency of memory in IoT embedded devices. For instance, many IoT applications inherently tolerate errors in data which opens up new possibilities for hybrid memory architectures composed of an error-free portion (for more reliable data and operations) and an error-prone one (for less important data). A hybrid memory based on this property has been

designed for a low power biomedical signal processors in [BMB+14]. For wearable IoT devices that need to have flexible and curvilinear forms, flexible non-volatile memory (NVM) has been proposed [GH15].

- **Data transmission**:
  At hardware level: Data transmission can be improved by integrating radio transceivers into SoCs, providing low power multi-radio chips, etc. Main existing technologies, trends and challenges in connectivity and communication are discussed in Section 3.3.
  At software level: In order to reduce the amount of data to be stored or transmitted, new data compression techniques, specially for the streams of data, are needed [PBM13].

Several new architectures have been designed and proposed in *Chair for Embedded Systems (CES)* to address the needs of emerging applications and technologies:

- Invasive manycore systems [HHB+12] offer a scalable hardware and software platform with a heterogeneous, tile-based manycore structure. This architecture along with its multi-agent management software can assist the analysis of Big Data in a manycore environment on the cloud servers.
- RISPP [BSKH07], another new architecture, is an extensible embedded processors concept that provides runtime adaptation to enhance the flexibility and efficiency of mid-end embedded (and IoT) devices.
- Dependability is one of the major requirement of new embedded and IoT devices. Ref. [HBB+11] presents an overview of design and architecture for dependable embedded systems. The design process of embedded systems starts from gate level and includes various levels of abstraction including operating system, application software, and systems architecture. This paper covers these levels of abstraction and considers error as a specific design constraint and suggest to develop techniques for error resiliency in embedded systems. It also introduces a new classification on faults, errors, and failures.

2.2 Edge Computing & Computation Offloading

Many recent and ongoing research efforts have addressed the foreseen challenges in IoT, including efficient resource allocation, service quality management, and computation offloading [CRMS09]. In mobile edge computing and wireless sensor networks, devices with multiple operation modes have been modeled in forms of either computation offloading schemes [MZ16, HWN12, Che15, YCY+13] or adaptive data input rate [XZST07, BMB+15].

Computation offloading has received notable attention in the domain of mobile computing in the recent past [Kha15]. Most of the proposed schemes aim at saving energy [XLL07, LWX01] or improving the performance of applications [Kha15]. However, they only consider the mobile device, as if there would be no resource contention between mobile devices. In IoT local networks, the limited resources (bandwidth, gateway's processing power) are shared between

multiple devices. Hence, the selected mode of one device would affect other IoT devices, as it changes the available resources for others.

In [SML+15], authors propose to minimize the energy consumption of sensor nodes by offloading the application tasks to other nodes. They optimally partition the workload of node, and then use a node selection strategy to find cooperative node to offload the workload to. However, in this approach there is a one-to-one relation between the nodes that offload their tasks and the nodes that cooperatively perform the offloaded tasks. Therefore, the computation and communication resources are shared between multiple nodes and consequently, there is no contention.

Ref. [SSB15] studies the joint optimization of the communication and computational resources for offloading problem. The goal is to minimize the overall energy consumption of mobile devices by offloading computation from mobile devices to resourceful cloud servers, while meeting the latency constraints. The proposed solution supports only two operation modes: either entirely offloading the computation or entirely performing it locally. The formulated problem is executed in the powerful cloud, and therefore is not optimized to be fast and efficient for embedded IoT gateways.

Mao et al. [MZ16] proposed a low-complexity online algorithm for dynamic computation offloading in energy harvesting devices. Considering the system state, this algorithm decides whether to offload a task or perform it locally. In addition, it determines the frequency of the processor and the transmission power of the radio. As it assumes unlimited resources at the offloading target side (i.e. gateway or server), this solution is not suitable for typical IoT systems with limited resources. Besides, they only consider one device solely instead of multiple devices in a local network.

In [AP13], the distributed mobile health monitoring system *mHealthMon* is presented. The cost of computation and communication is taken into account to minimize the overall energy consumption by means of program partitioning, network resource allocation, and network selection. However, the problems of resource allocation and program partitioning are solved separately: first the network resources are determined and allocated to maximize the network utilization. Then according to the allocated resources, the program is partitioned into blocks for offloading. However in typical IoT systems, maximizing resource utilization is not an objective. Instead, improving the efficiency of devices (e.g. energy consumption reduction, battery lifetime extension, etc.) is the main objective. The control parameters to achieve this goal in IoT systems are the given operation modes of edge devices. Therefore, the problem of resource allocation and mode selection shall be solved conjointly to promise more efficient systems.

A decentralized game theoretic approach for computation offloading in mobile computing systems is presented in [Che15]. A single gateway is considered, whose wireless channel is the scarce and shared resource while the processing capability of the server is unlimited. In addition, this approach does not support multiple levels of offloading for a device. In other words, the decision is between fully offloading the computation or fully processing on-board.

2.3 Mode Selection and Resource Allocation

Several research works attempted to address the resource allocation problems, particularly the bandwidth allocation in shared wireless networks, by determining the optimal operating mode for each device [Che15, SSB15].

MiLAN [MH03, HMCP04] is a middleware to manage and allocate network resources for the applications, that *fuses* data from multiple sensors and needs to select optimal set of sensors. However, the SQ of each sensor is fixed. Moreover, MiLAN only considers the bandwidth limitation of the network while the processing capability is not modeled. Besides, it does not model the on-board processing and therefore cannot support combinations of offloading schemes.

In [MPW07], a utility-based approach is studied for bandwidth allocation in wireless networks. The proposed approach is market-oriented but in a centralized fashion that is designed for single gateway systems. Hence, it does not support computation offloading.

A protocol for bandwidth allocation in mobile ad-hoc networks is presented in [CH05]. It uses a method to estimate the approximate bandwidth, finds the residual available bandwidth and then reacts to the network traffic. This solution is not applicable to the systems where applications operate at different input data rates with multiple offloading policies.

In [OPP+15] an approach based on game theory is proposed to allocate the bandwidth dynamically in a shared network channel to manage the quality of experience. One of the restriction of this approach is that it needs a continuous function of bandwidth allocated to the device (i.e. does not support discrete operating modes).

An optimal dynamic resource allocation technique is presented in [VQA14] for mobile cloud computing. While the mobile application can be offloaded to the cloud servers, the computation resources and the communication bandwidth are shared and constrained. Considering the quality of service requirements, power consumption, and available resources, a multi-objective optimization problem is formulated to select the network resources and the QoS profiles for applications. The optimum offloading coefficient (i.e. the portion of workload to be offloaded) is determined by a method that uses dynamic programming at its core algorithm.

3 IoT Enabling Technologies from an Embedded Design Perspective

This chapter aims at exploring the IoT and its technological enablers from embedded design point of view. Besides investigating essential technologies for IoT and existing trends, this chapter provides distinguishing properties of IoT for embedded domain in addition to a comprehensive IoT perspective for embedded systems. It presents the essential findings and insights that are published in [SBH16].

The chapter is organized as follows: Section 3.1 presents a short motivation and introduction on the IoT challenges in embedded design. Section 3.2 provides discussion on IoT application domain, properties of IoT devices and their applications. Section 3.3 discusses one of the most important aspects of IoT devices, the connectivity. Wireless low-power technologies and their properties are summarized, and the suitability of those technologies for each IoT domain is discussed. Section 3.4 is dedicated to the computation layers in an IoT system including embedded devices, gateways, cloudlets and cloud servers. Finally, Section 3.5 studies the opportunities for using approximate computing in different components of an IoT device to meet its tight constraints.

3.1 Motivation

Recent and ongoing advances in the technologies such as wireless communication, ultra-low power processors, embedded sensors and actuators, Radio Frequency IDentification (RFID), mobile phones, and cloud/fog computing has enabled the emergence of IoT [GZY+13]. Although not all those technologies are needed for each and every IoT application, they all facilitate the proliferation of IoT by providing an essential prerequisite [MSDPC12, LL15]. While RFID enables low-cost object identification, and while ultra-low power system-on-chips (SoC) enable portable battery-operated embedded devices, cloud computing and fog computing can be used to offload computations and services to the local or global servers, providing additional resources for handling large-scale data or performing more complex operations [BMZA12, GBMP13].

IoT takes advantage of several existing technologies and application domains and brings them under one umbrella. These technologies include wireless sensor networks (WSN), machine-to-machine (M2M), RFID, Cyber Physical Systems (CPS), Mobile Computing (MC) [WTJ+11,

DXHL14, S+14]. There have been many research efforts on IoT from the perspective of networking, object identification, data access (security and privacy) [GZY+13, LL15], however, it has gained less attention from the perspective of embedded computing.

The diversity of IoT applications, their requirements and technologies makes it difficult to present a general comprehensive statement for the requirements of IoT in hardware and software. Therefore, the IoT embedded designer faces questions whose answers are challenging as the solutions can be contradictory, e.g.:

- Which wireless communication technology 1) covers the required range, 2) provides the required data rate, 3) is still (ultra) low-power and meets energy constraints?
- What trade-offs to make between 1) Quality of Service (QoS) and energy consumption, 2) on-board processing and computation offloading, etc.?
- How to handle the uncertainty and unpredictability of IoT systems (mainly caused by communication)?

3.2 Properties of Devices and Applications

3.2.1 Application Areas

IoT can impact various application domains either by enabling new services, or by improving the efficiency of existing ones [PZCG14]. Among the possible applications, this section provides a review of their main categories (that cover a wide range of different requirements, technologies, development challenges) and futuristic applications. Indeed, IoT applications are not limited to these categories, and a huge number of applications can be envisioned. However, their requirements, properties, and design challenges have similarities with those presented in these categories. The main challenges and requirements are discussed in the following sections.

– Healthcare:
IoT has shown a great potential for enabling and improving healthcare services [HPS+15]. IoT-based healthcare systems enable long-term monitoring of personal health status in real-time anytime, anywhere. They acquire vital biosignals including electrocardiogram (ECG) –electrical signal of heart–, electroencephalogram (EEG) –electrical signal of the brain–, and electromyogram (EMG) –electrical signal of muscles–, body motion, blood pressure, Glucose level etc. The data can be processed real-time or could be transmitted to a remote device (e.g. cloud server) for further processing and diagnosis [GRW+15, ABC+15]. Ultra-low power design and real-time constraints are among the challenges for these applications.

– Assisted Living:
Assisted living aims at offering solutions for helping (i) elderly, (ii) chronically ill, and (iii) disabled people [AIM10]. For instance, a wearable IoT device can leverage online city maps together with a smart cane to detect and avoid obstacles, access buildings, navigating indoor and outdoor, etc. [Dom12]. Another example is fall detection systems that use wearable devices

or installed devices at home to provide immediate assistance to patients by issuing a message to the emergency center or family members if needed [PPB+12].

– Smart Building and Home:
IoT provides connectivity for embedded devices which can enable applications for reducing the costs, increasing personal comfort, and improving safety and security in buildings and homes [S+14]. Smart thermostats, smart lights, smart doors, etc. are among the IoT devices that improve the efficiency in the buildings and homes.

– Smart City:
In a smart city, distributed IoT devices equipped with different sensors are used to improve the transportation and traffic management, monitoring the air quality (e.g. pollution, temperature, humidity, etc.), smart parking, smart lighting, and smart watering gardens [cit, TBA+14].

– Smart Industry:
IoT-enabled solutions for automation, control and monitoring may improve industry by lowering operational and maintenance cost, and increasing quality of service [LL15], for industrial domains such as supply chain management, transportation and logistics, and automotive [PZCG14, KS15]. For instance, food industry can exploit systems based on RFID and NFC for tracking, monitoring, and tracing food quality. Another example is remote monitoring of machinery (e.g. in plant, wind turbine, etc.) for predictive maintenance [LL15, DXHL14].

3.2.2 Applications vs. Devices

The combination of IoT applications and their underlying hardware or device introduces some design challenges which need to be address either at software application or at the hardware level. The relation between the number of devices and number of provided service and applications can be classified into four categories as also shown in Figure 3.1:

- One-to-One: One IoT device is used for a single service. For instance, an IoT-based healthcare monitoring device that captures real time biosignals [SBH15, BCM+15, MNMKSK+15]. Another example, is the IoT-based smart cane to help blind people navigating [Dom12].
- One-to-many: One single IoT device provides multiple services. One example is a wearable device like a smart watch that has several sensors and can keep track of user's physical activity, heart rate, location, etc. [MSDPC12]. Another example is a smart conference room which uses a single device for multiple applications including detecting the start/end of a meeting, analyzing the environmental condition of the room (e.g. temperature and luminance), and processing the acoustic signals to record the proceedings of the meeting, etc. [SBHH15]. For this category, a decision that needs to be made by the designer is the management of shared resources. The solutions range from conservatively choosing the underlying hardware (processor, memory, wireless ratio, etc.) which support the worst case accumulated usage, to dynamically managing and scheduling the hardware usage.

- Many-to-one: In this class, spatially distributed devices provide a single service. For instance, distributed smart cameras are exploited for video surveillance in [CCT+15]. This category usually has two properties that need to be considered by designers to optimize the system: 1) high communication between devices and 2) large amount of redundancy.
- Many-to-many: In some IoT applications multiple devices are shared between multiple applications and services. Smart Citizen [sma] consists of multiple IoT embedded devices that are geographically distributed to gather information for the applications that report temperature, humidity, noise level, and air pollution. Shared WSNs belong to this category, too [SBHH15, LEMC12].

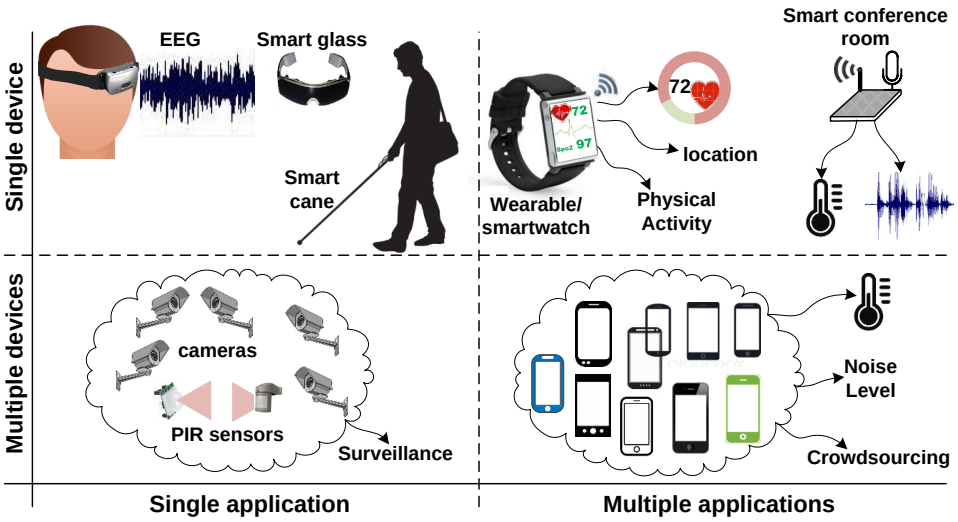


Figure 3.1: IoT systems may exploit single/multiple devices to implement single/multiple applications

Figure 3.1 shows these categories with some examples. Although shared IoT devices reduce the cost of hardware and maintenance, they introduce new challenges like binding, allocation, online and runtime resource management, guaranteeing the timing constraints of applications that share a single device, etc. [S+14, WLS+15, Kim15].

3.3 Connectivity

Connectivity (wired or wireless) is what distinguishes embedded IoT systems from conventional embedded systems. In a broader sense and vision, IoT is a global infrastructure of *heterogeneous*, *networked* embedded devices and objects [GKN+11]. Communication ability, and in particular the Internet connectivity, lets devices and smart objects (also known as machines) communicate and interact with (i) other machines and devices, or (ii) humans [MSDPC12, WTJ+11, DXHL14].

3.3.1 Wireless Communication Technologies

Different wireless communication technologies can be used for (i) connecting the IoT device as local networks, and (ii) connecting these local networks (or individual IoT devices) to the Internet. IoT wireless communication has three main aspects as shown in Figure 3.2: data rate, communication range and power consumption. IoT devices and applications benefit from a wireless radio with high data rate, long range and low power consumption. But these three metric cannot be improved at the same time. Each wireless technology can achieve only two out of three. Improving one aspect costs degradation in another one [Mula]. In other words, each technology falls into one of the areas labeled with 1, 2 and 3 in Figure 3.2 . In most of IoT applications, low power consumption is a critical objective. Therefore the communication range is compromised to gain the required data rate or vice versa.

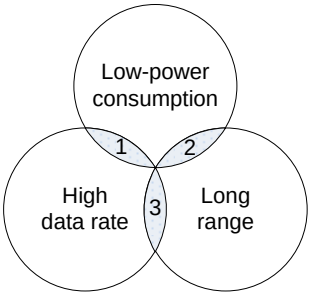


Figure 3.2: Three main aspects of wireless technologies. Each technology can be optimized for only two out of three [Mula].

The main IoT wireless technologies in the market are as follows:

- **NFC** [COO13]: It is a short-range wireless communication technology that enables the data transmission between devices in a close proximity to each other ($\sim$20 cm). It has a tag that can contain small amount of data. This tag can be read-only (similar to RFID tags for identification purposes) or can be re-writable and be altered later by the device.
- **Bluetooth**: This technology witnesses an increasingly ubiquitous presence including in smartphones, tablets, laptops, headsets, etc. Therefore it is a suitable technology for the IoT devices in the consumer sector where the end user can control or monitor the surrounding IoT devices without special equipment such as dedicated gateways but just using his/her smartphone.

    1. **Classic Bluetooth**: It offers a high enough throughput and bandwidth which makes it suitable for data stream applications (e.g. audio). However, it has several limitations including limited number of nodes in the network (up to seven slaves) or topology. The power consumption is also relatively high which does not fit to the ultra-low power applications.
    2. **Bluetooth Low Energy (BLE)**: It is also known as Bluetooth smart and is designed and enhanced for short-rage, low bandwidth, and low latency IoT applications. The

advantages of BLE over classic Bluetooth include lower power consumption, lower setup time, and supporting star topology with unlimited number of nodes. BLE is quickly becoming the dominate transceiver in IoT.

3. **Bluetooth 5 (BT v5)**: To better fit to the IoT requirements, Bluetooth 5 has been enhanced and features longer range (up to four times) and higher data rate (up to two times) compared to BLE while increasing the capacity of broadcasts. It provides four discrete data rate speeds (i.e. 125 Kbps, 500 Kbps, 1 Mbps and 2 Mbps) which offers a wider range to support diverse application requirements. The overall throughput is 5x higher than BLE. Bluetooth 5 offers long range communication with two of its modes (i.e. data rate of 125 Kbps and 500 Kbps) in which it uses Forward Error Correction (FEC). It is anticipated to become the *de-facto* wireless technology in IoT.

- **ZigBee**: A small-size, low-cost, low-power wireless specification that can support different network topologies (e.g. mesh, star, tree). It offers a wide transmission range, depending on the output power. Although ZigBee has established in some industrial applications and WSN nodes, it faces some market barriers, especially with the emergence of attractive alternatives like BLE that provides higher bandwidth at a lower energy consumption.
- **WiFi**:

1. **Conventional WiFi (IEEE 802.11 b/g/n)**: The main advantages are high bandwidth and availability in urban districts. Its high energy consumption makes it unsuitable for ultra-low-power IoT devices.
2. **Low-power WiFi (802.11 ah) or HaLow**: Compared to conventional WiFi, it is intended to extend the range of transmission with less data rate, and to reduce the energy consumption for IoT applications. In addition, it will suffer less interference with existing wireless networks as it uses a different frequency band (i.e. 0.9 GHz).

- **Cellular network**: Widespread mobile networks like 3G and LTE provide reliable high-speed connectivity to the Internet. However, they have a high power consumption profile and they are not suitable for M2M or local network communication.
- **Low Power Wide Area Network (LPWAN)** [LL16]: These technologies are suited for low power applications with very long rage transmission. They support up to 10 Km distance between end-nodes and gateway. However, it comes at the cost of very low data rate (<1 Kbps). Main technologies of LPWAN include SigFox, LoRaWAN and Weightless, which operate in sub-GHz bands. One of the challenges for LPWAN is the lack of a globally available band for LPWAN in sub-GHz.

Table 3.1 summarizes some characteristics of wireless technologies which are needed to be considered in the design process of IoT devices. This table shows the typical value of these

Table 3.1: Communication technologies for IoT applications, and their properties

		NFC	Bluetooth	BLE	BT v5	ZigBee	HaLow 802.11 b/g/n	LP WiFi 802.11ah	LPWAN	Cellular network	
										3G	LTE
Range	indoor	<0.2 m	1–100 m	∼100 m	<300 m	<20 m	<70 m	<700 m	<10 Km	>5 Km	>5 Km
	outdoor					<1500 m	<230 m	<1000 m			
Bit rate [Mpbs]		0.424	1–3	1	2	0.25	>1	0.15–40	<0.05	0.17	75–300
Throughput [Mbps]		0.22	1.5	0.30	1.5	0.15	2-50	>0.1	<0.05	NA	
freq. [GHz]		0.014	2.4–2.5	2.4–2.5	2.4	2.4	2.4/5	0.9	sub-GHz	0.8–1.9	2.1
Network topology		p2p	scatternet	star, scatternet	NA	star, tree, mesh	star	star	star	NA	

Table 3.2: Suitability of communication technologies for IoT application domains

		Application domains					Local Network (M2M)
		Healthcare	Smart Cities	Smart Building	Automotive	Industry	
wireless technology	NFC	medium	high	low	very low	very high	medium
	BLE	very high	low	low	very low	low	high
	ZigBee, BT v5	medium	high	very high	low	high	high
	WiFi b/g/n	low	high	medium	medium	low	high
	HaLow	high	very high	high	high	high	very high
	LPWAN	low	very high	high	high	very high	high
	Cellular networks (3G, LTE, etc.)	low	high	high	high	medium	very low

parameters. Indeed, most of them depend on the design constraints of the IoT devices. For example, by increasing/decreasing the transmission power (and the size of antenna), the transmission range can be further increased/decreased. Table 3.2 shows the suitability of each wireless technology for different application domains.

Each of these communication technologies has its advantages and disadvantages. For instance, Bluetooth, WiFi, and ZigBee may face interference due to the coexistence of other devices working at the same frequency band (i.e. 2.4 GHz), especially with the increasing rate of IoT devices. The interference can lead to severe drop in data rates, which consequently may increase the energy consumption of IoT devices, reduce the QoS, and result in missing the deadline in real-time applications (i.e. affecting other optimization goals of the system). On the other hand, NFC needs sender and receivers to be close to each other.

Hybrid communication schemes seem to be the best fitted solutions for IoT applications. The emergence of integrated transceivers which include multiple communication technologies on a single chip[1] has opened doors for more efficient wireless communication. For instance, BLE is not intended for continuous data streaming applications, but it is highly efficient for sending small, discrete data chunks (e.g. states, temperature, heart rate). Bluetooth classic and WiFi, on the other hand, offer higher throughput and efficiency for streaming application with higher data rate demands (e.g. ECG monitoring, audio).

3.3.2 Timing of Communication

The timing of data transmission schemes in IoT applications can be classified into three different categories.

- **Continuous**: The IoT devices send or receive data continuously (e.g. real-time health monitoring).
- **Sporadic**: The IoT device collects and stores the data and then transmits it whenever the connection is available [DMSS15]. Long-term health monitoring applications for fitness and wellness which keep track of biomedical signals in a daily basis, and analyze them later on the cloud server are examples of this category.
- **On-demand**:

1. **User driven**: The IoT device can be requested by the operator to send the collected data [SCL+05].
2. **Event driven**: The communication is done once a specific event happens. For instance, the IoT device is capturing biosignals from person's body, and processing it by some basic feature extractions. If a suspicious anomaly is observed, the device starts sending the collected raw biosignal data to the cloud server for further and deeper analysis [MNMKSK+15].

For designing an efficient IoT system, the timing of data transmission matters, especially for managing low power modes (deep sleep, standby, active). For instance, if the data transmission

[1] For example, TI CC1350 modules provide BLE and sub-1 GHz radios on a chip.

is sporadic, a predictor based on learning techniques can be used to dynamically manage the low power modes of the wireless transceiver.

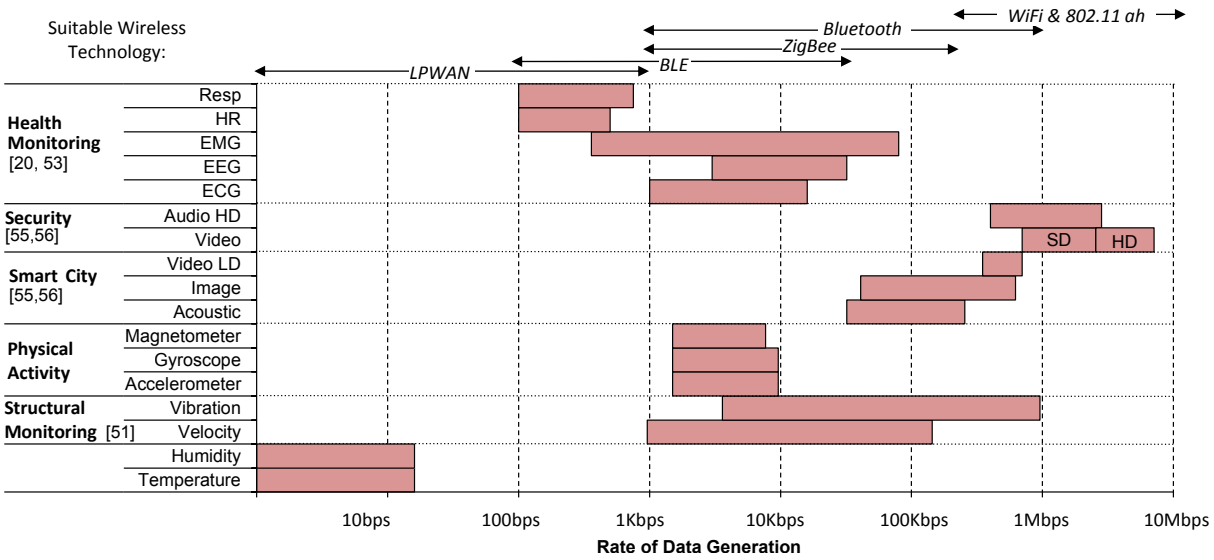


Figure 3.3: Data rate generation by different sensors in typical IoT applications

3.3.3 Bandwidth & Data Rate of IoT Sensors

This subsection provides a comprehensive overview of the main sensor applications in IoT. Then it derives the typical data generation rate of different sensors according to the reported application scenarios. The required wireless transmission bandwidth and the appropriate technology is annotated, too. Then it presents an estimation for the required processing power to perform the typical operations on the captured data, and reports the appropriate IoT hardware core to support it.

- **Ambient/Object Temperature, Humidity**: Different applications including smart homes, smart cities, smart industry, etc. use these sensors.
- **Accelerometers and Gyroscopes**: These sensors are widely used in industrial machinery, medical and fitness devices, as well as wearable devices for monitoring physical activity, fall detection, navigation, or for structural health monitoring of machinery or buildings [Kim05].
- **Magnetometer**: It usually comes along side with accelerometers and gyroscopes and is used for finding the direction for navigation.
- **Light**: This sensor is usually used for saving energy consumption by adjusting the lighting (e.g. smartphones, smart homes, smart cities).
- **Chemical Sensors**: These sensors are mainly used for measuring the air quality (e.g. CO, NO2, SH2, and CO2) in smart city application, environment monitoring, or smart factories [sma].
- **Location**: These sensors are mainly based on GPS technology (seldom work indoors). Smart industry and transportation use this sensor for tracking and localizing the objects.

- **Imaging**: Besides normal imaging sensors, infrared sensors are emerging in IoT market to capture the temperature differences of the objects. The feature is useful for security applications to detect intruders or getting thermal image of home or objects to detect insulation or possible leaks.
- **Acoustic**: Analog or digital microphones are used for security and surveillance, and noise pollution for smart cities [sma].
- **Ultra Violet (UV)**: It measures the strength of UV radiation for healthcare, wellness, and smart city applications. It also indicates the strength of sun's exposure for smart farming and smart agriculture.
- **Ultrasonic**: It is useful to detect the obstacles and get the distance. Assisted-living and smart industry can benefit from this sensor to help visually impaired people or preventing the moving robots to collide, respectively.
- **Health Monitoring Sensors**: As mentioned in Section 3.2.1, healthcare applications of IoT [HPS+15, CDDM+12] include, but are not limited to:
  - Heart Rate: To detect the heart rate (and consequently heart rate variability).
  - ECG: To measure the electrical activity of the heart which conveys essential information about the status of heart and the function of its muscular contractions [BCM+15].
  - EMG: To measure the electrical signal causes by muscular activity [SCL+05] for gesture recognition, detection of neuromuscular diseases, etc. [BCM+15].
  - EEG: To capture the electrical voltages which represent the brain activity. The captured data has many applications in diagnosis (e.g. epileptic seizure detection), well-being (e.g. stress monitoring and sleep monitoring) and brain-machine-interface.
  - Blood Pressure
  - Respiration Rate
  - SpO2: The arterial oxygen saturation or the amount of oxygen dissolved in blood.
  - Skin Conductivity: To measure the conductivity of skin to detect psychological or physiological arousal, or the moisture level of the skin [CLM+10].
- **RF radio modules**: Although RF modules are primarily used for communication, they can be used as localization sensors. The strength of received signal can be used for indoor localization and navigation [PBCA15].

3.3.4 Analysis & Insight

Figure 3.3 illustrates the typical data rate of different IoT sensors in different applications. Those numbers are obtained based on the typical resolution of captured data and typical sampling rates in different case studies and applications [vid13, SSGS15].

Figure 3.4 estimates the required processing capability to perform the basic operations and processing tasks on the captured signals of different sensors in different IoT applications. For instance, to process the EEG data on the IoT device, it needs to be subdivided into different bands (e.g. Delta, Theta, etc.) by filters. Then, different bands are transformed to frequency domain using Fast Fourier Transform (FFT). The average number of CPU cycles to perform

those operations per sample are measured and then the required frequency to process the data at the derived data rates is calculated.

Figure 3.5 shows the minimum required frequency for a microcontroller to transmit the unprocessed data to the Internet (through the gateway). Those numbers are obtained under the assumption that the microcontroller needs to encrypt the data and add error detection codes to it before transmission. Advanced Encryption Standard (AES) and cyclic redundancy check (CRC) are used for this purpose, implement them and measured the average execution time and number of cycles to perform these operations on one byte of captured data. Then, using the derived typical data rate, the required CPU cycles to transmit the data are estimated.

The number of CPU cycles and execution time for the typical operations (e.g. FFT, FIR, AES, CRC) are obtained from the conducted experiments on the Atmel Atmega328 microcontroller which is used in IoT platforms including Arduino boards. Table 3.3 summarizes the measurements for those operations.

Table 3.3: Typical data processing operations in IoT applications and their execution time (per Byte)

Operation		**Exe. time [μs]**	**# of cycles**	**Code Size [B]**
FFT (256 p)		18	586	3850
FIR		1492	23872	1750
AES	Enc	18.25	292	3666
	Dec	22.75	364	
CRC		3.75	60	500

3.4 Different Computing Layers

An enormous amount of data, including streams of data, audio, or video, will be generated from IoT devices which is also known as Big Data [ZMK+15]. The problems associated with emerging Big Data (e.g. massive storage and huge processing power demand, high latency, etc.) necessitates the migration of computation and processing to different underlying computing layers available in the IoT chain [GRW+15, Kim15]. Figure 3.6 shows different processing and computing layers starting from IoT embedded device up to the cloud servers.

The collected data can be processed on either IoT device, or gateway, etc. Hence, for IoT applications that involve data processing, a major challenge is to decide where the computation should be done (i.e. computation offloading) [Kim15, PJZ+14]. The decision depends on many parameters and factors including the system objectives (e.g. real-time requirement, energy efficiency, etc.), and system specifications (e.g. energy consumption for data processing and data transmission on IoT device, communication bandwidth, transmission delay, etc.). As shown in Figure 3.6 different computing layers and platforms include:

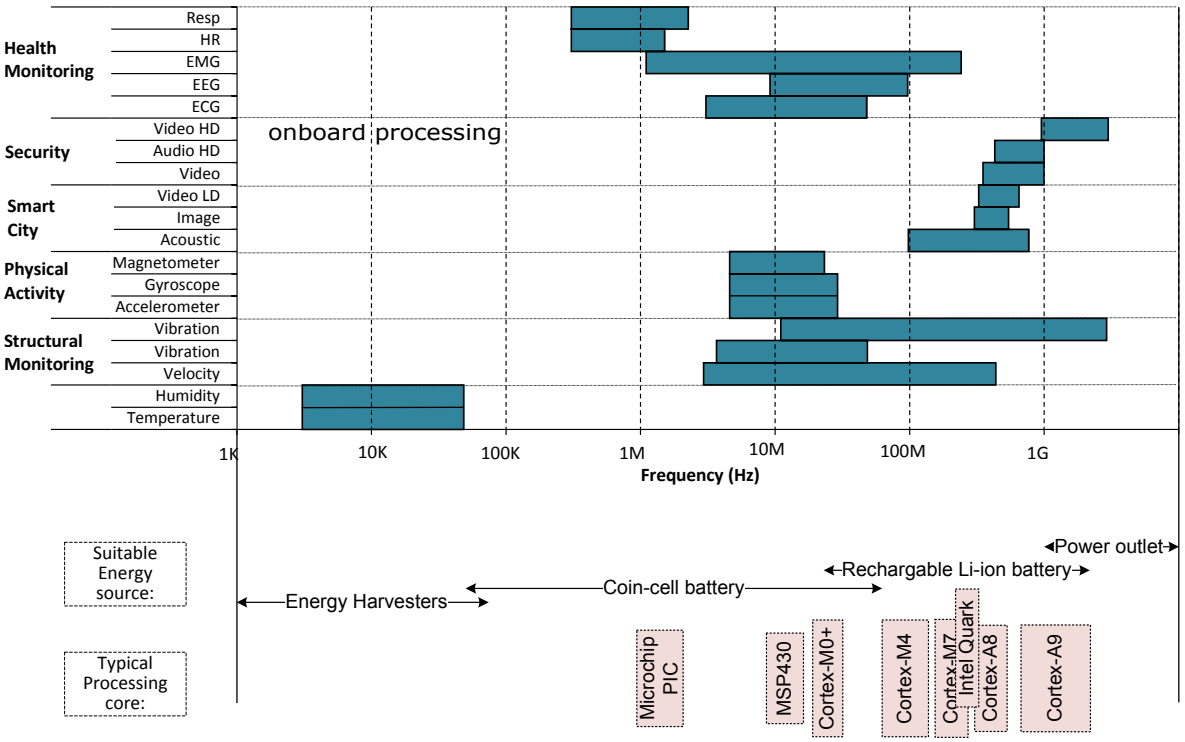


Figure 3.4: The number of cycles (i.e. required frequency) to fully process the IoT sensors, and their expected power source

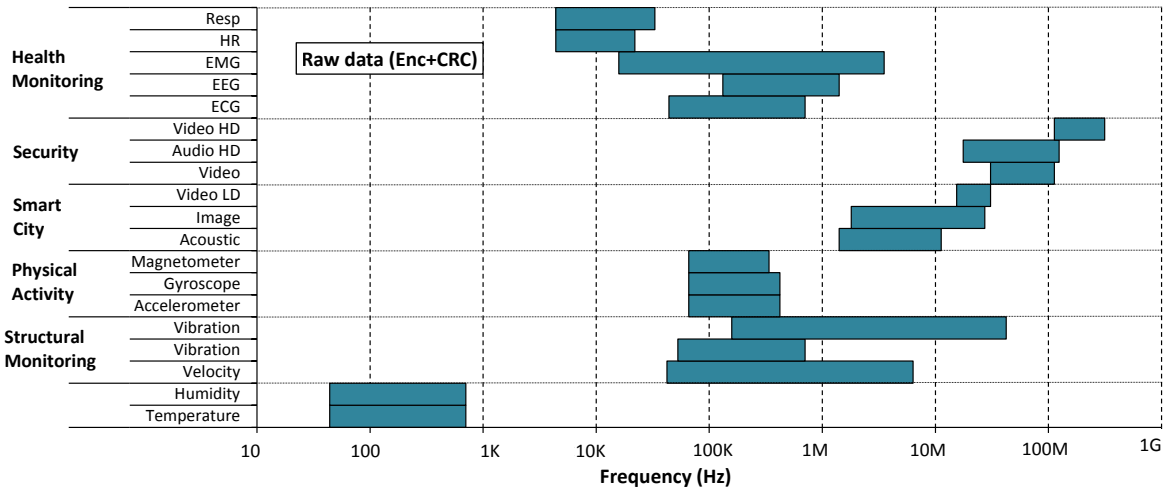


Figure 3.5: The number of cycles (i.e. required frequency) to encrypt and transmit the captured data from IoT sensors

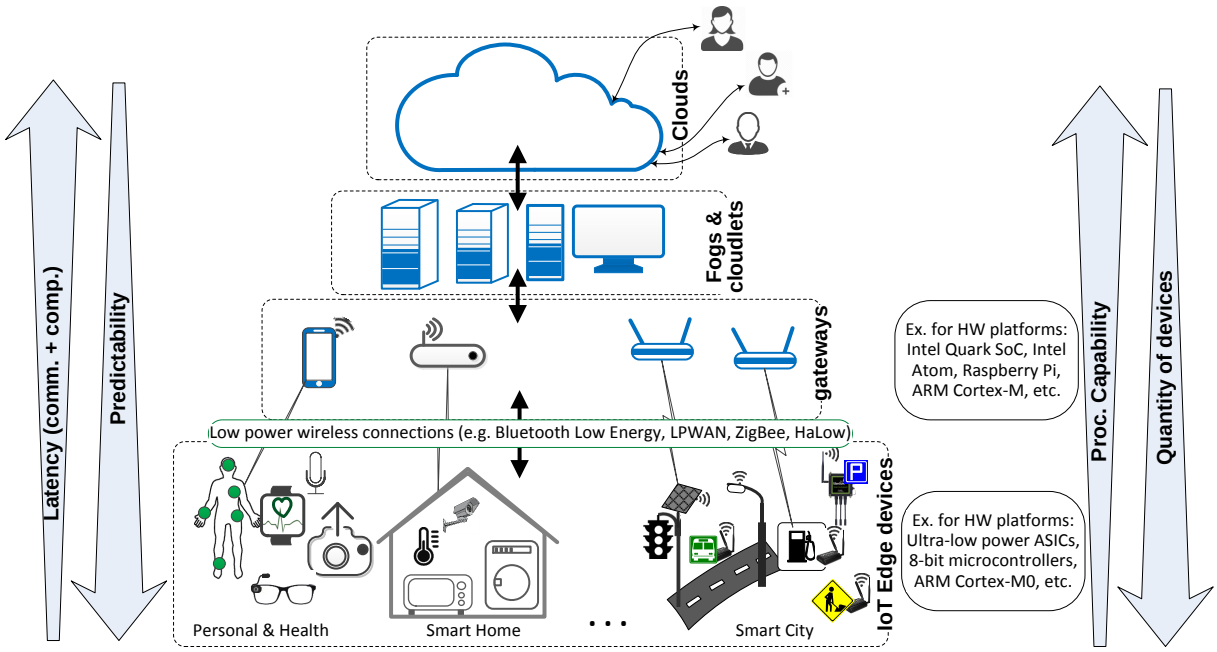


Figure 3.6: Different computation layers. The available resources (e.g. memory, processing power) and networking latency increase from bottom to top.

- **Device centric**: The microcontroller in an IoT device can be exploited to perform the computation. The main challenges are the scarce resources on IoT devices (e.g. on-chip memory, processing capabilities) and to meet the low-power requirements. A key decision concerns whether to perform the computation on the IoT embedded device or to offload it to other layers of computation. This decision making becomes more challenging when the answer should be determined at runtime (i.e. depends on the operation, input data, energy source's status, and other runtime parameters) [Kim15].
- **Gateway centric** (e.g. smartphone): IoT gateway devices which are used to settle the heterogeneity between different networks and Internet usually have more computational power [PJZ+14, DBN14] (e.g. ARM Cortex-A and Cortex-M MCUs are proposed as gateway processor in ARM, Freescale and Texas Instrument solutions). Gateway devices like smartphones can be used to perform data processing as proposed in [ZKC+15]. This scheme has been used for medical and healthcare monitoring application in [SBH15], where the captured ECG signal is transmitted to the smartphone to be processed. The key challenge here is to guarantee the availability and deadline constraints, which is difficult due to the unpredictability of the wireless communication and latency, especially when the number of IoT devices increases.
- **Fog centric**: The concept of fog computing is the extension of the cloud computing paradigm, which can help the IoT domain to restrain the Big Data problem [BMZA12, ZMK+15]. Fogs provide more computational power compared to IoT embedded and gateway devices and have less latency compared to the cloud servers (due to their location and distribution) [GRW+15, AFGM+15]. An ECG feature extraction for a healthcare monitoring application is presented in [GRW+15] and shows the benefits of the fog computing concept.
- **Cloud centric**: [GBMP13, KS15] Cloud computing provides a solution for handling Big data and processing them. It requires massive data storage volume, huge processing resources to deliver a high quality of service and to help decision making [LL15]. However, as the number

of IoT devices increases, and consequently, the amount of stream data increases, the cloud computing solution faces problems and challenges including the scalability, high energy cost, latency, bandwidth, and availability [HPS+15, ZMK+15].

- **Hybrid approach**: Since each of the aforementioned choices have their advantages and disadvantages, solutions that exploit a multi-layer and hybrid computing approach are more efficient. An example: classification methods, including Support Vector (SV) machines, are widely used for anomaly detection or gesture recognition in personal healthcare monitoring applications [BCM+15]. A set of SVs are stored in memory which will be used at runtime for classification. When the number of SVs is large (i.e. needs large memory on IoT device), an efficient solution is to store some SVs on the gateway (e.g. smartphone), instead of increasing the memory size at design time. In this solution, if a feature is far from the SVs that are stored on the IoT device, it will be transmitted to the smartphone, and compared with other SVs. Such a hybrid approach enables the applications to increase their efficiency and meet their requirements by leveraging the advantages of each computing level (e.g. larger memory, more computational resources, lower latency, etc.). However, the main challenge to address in a hybrid approach is to find the efficient balance between local processing and computation offloading, and to find the perfect timing for offloading (in real-time applications).

3.5 Approximate vs. Exact Computing

The emerging paradigm of approximate computing leverages inherent resilience of applications and relaxes the requirement of exact equivalence between the specification and implementation to gain more efficiency [CCRR13, HO13]. Most of IoT applications are interacting with the physical world with noisy input data [S+14]. Therefore, they are inherently dealing with approximation. For instance, the first stage of approximation happens in the Analog-to-Digital conversion which introduces a quantization error. Although these applications tolerate some errors, the final output or QoS should be in a certain range.

Many applications can exploit the error tolerance property to trade the output quality for computational effort (e.g. energy consumption, performance, etc.). Some open challenges exist in the domain of approximate computing for IoT. The tolerable error can be accepted and exploited at different hardware components and different software parts:

- **Data acquisition**: The quality of input data, during the data acquisition, is determined by resolution and sampling rate (or frame rate). For instance, the ECG signal in [GRW+15] is capture at 360 samples per second with 11-bit resolution. When the IoT application tolerates error, reducing the quality of input data is one way to take advantage of it to reduce the energy consumption or delay. In [RT09], a low-power analog system is presented to adjust the sampling rate in a body sensor network to reduce the acquired data.
- **Data processing**: The approximation can be done also in underlying hardware by designing inexact hardware units for specific arithmetic operations (e.g. adders, multipliers, DCT,

FFT, etc.) [HO13]. It can also be done at the software level by methods like stage skipping [CCRR13].

- **Data storage**: The tolerable error can be exploited at the memory unit to reduce the size of required memory, reduce the number of access to memory, or reduce its energy consumption. One example of approximation in memory is presented in [BMB+14].

Indeed, hybrid schemes, where multiple stages exploit approximation, are also possible. The main challenge here is to decide 1) at which stage, and 2) how much approximation should be applied in order to minimize the computational effort while meeting the QoS requirements.

4 Efficient Resource Management Techniques for IoT Edge Computing

This chapter first presents a general model for the IoT applications in the healthcare domain which process input signal data in Section 4.1. Then it introduces the operation modes of IoT applications as a control parameter to adjust to the varying situations of IoT environment. The operation modes are the results of different computation offloading level and/or service quality levels that can be chosen at runtime.

Then, it considers IoT systems where multiple IoT devices are connected to a shared gateway, and hence the limited resources of the gateway are shared between devices. It presents efficient and low-overhead techniques to select the operation modes of IoT devices at runtime such that the constraints of individual devices are met, the sharable resources of the gateway are allocated to the devices to achieve the optimization goal of the system. Each section is dedicated to a different problem and technique, followed by evaluation of the proposed solution.

Particularly, Section 4.2 addresses the problem of service quality (SQ) management for IoT devices under bandwidth, battery, and processing constraints. This section first formulates the problem of resource-aware SQ tailored to the IoT paradigm and then proposes an efficient problem decomposition that enables the adoption of a recurrent dynamic programming approach with reduced execution time overhead. The proposed approach is evaluated with a case study and through extensive experimentation over different IoT systems, regarding the number and type of the employed IoT devices. Then in Section 4.3, an efficient and fast technique to select the operation mode of IoT devices is proposed. The proposed technique is especially advantageous when multiple types of resources are shared between devices.

Section 4.4 is dedicated to the IoT device management in a multi-gateway system. A trade-based distributed mechanism is presented to find the sub-optimal solutions. It starts with an initial setup and step-by-step improves the overall SQ of the system by migrating the IoT devices to other gateways and exchange the IoT devices between gateways when it is beneficial. Eventually, Section 4.5 summaries the main contributions and concludes the chapter.

4.1 Model for IoT Applications in Healthcare

Figure 4.1 illustrates the general model for IoT applications whose input data is in form of signal and their task is to classify the event based on input data. It not only fits to the above-mentioned IoT applications, but describes other applications including gesture recognition using

EMG processing [MBF17], structural health monitoring, etc. [LYZ+17]. The main properties of these applications are the following, which are also shared by many other IoT applications:

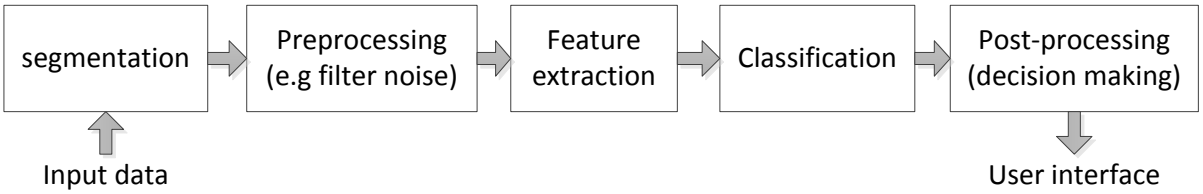


Figure 4.1: Genral model for IoT applications which classify input signal

- they operate on segments of data, which are buffered first and then acted upon,
- while processing one segment, the data for the next segment is captured and buffered,
- they are composed of a multi-stage pipeline structure to process input data, where discrete operations are performed per pipeline stage,
- the input data rate in most of IoT applications is relatively low. For instance, the sampling rate for the above-mentioned applications is only a few hundred samples per second. The epileptic seizure detection accepts EEG signals with 200-400 Hz. The ECG signal in heart monitoring applications is captured with sampling rate in the range of 100 Hz to 400 Hz. The length of segments is usually in the range of few seconds (e.g. 5 seconds in EEG application, 3 seconds in ECG application and 4 seconds in physical activity monitoring).
- due to generally low input data rate, the processing of one segment of data is finished before the next segment is completely buffered. This has two consequences: 1) the processor spends most of the times in the low-power sleep mode before the next data segment is ready [HPA+17], and 2) at each point in time only one stage of pipeline is busy.

4.1.1 Computation Offloading

The IoT application may either 1) fully process the captured data on the IoT device and send only the results to the Internet, or 2) partially process the data on the IoT device and offload the rest of the computation to the gateway or 3) offload the whole computation to the gateway by transmitting the raw data. This would lead to different possible offloading levels.

Regardless of pipeline structure of the IoT application, it has at least two computation offloading levels: (i) Level 1 submits the raw data without on-board processing, (ii) Level 2 indicates 'no computation offloading', thus fully processes the data and only transmits the results. Moreover, the pipeline stages (in Figure 4.1) are potentially suitable to be considered for offloading the partially-processed data. For instance, consider an IoT-based fitness & heart monitoring device that can capture and process ECG signals. This device may have 4 offloading levels. Level 1 transmits the raw data. Level 2 could indicate 'no offloading' and thus only transmit a small amount of results (e.g. the features extracted from the signal). Level 3 could be used to offload

a certain percentage of the input sample rate. For instance, the device could store a certain amount of input samples in a buffer, then process the buffer and offload all incoming samples during this processing to the gateway. Level 4 could perform some pre-processing on the data, e.g. apply a filter on it, and then send the filter output to the gateway.

4.1.2 Service Quality

Some IoT applications offer their service at different qualities, ranging from 'low quality' to 'high quality'. Different service qualities can be achieved either by employing more complex processing pipelines or by exploiting higher quality input data (e.g. sampling rate [TSE+15]). For instance, consider the IoT-based EEG processing device that can capture EEG signals at any of multiple discrete sampling rates (e.g. 200 Hz, 300 Hz, and 400 Hz). The quality of the captured signal allows for processing of higher resolution and thus determines the service quality and affects the user's satisfaction [STX+16b]. Another example is when the device can use different classification algorithms which differ in accuracy vs. computational requirements. A more advanced classification algorithm provides higher service quality to the user.

The service quality is quantified as the *utility* which is user's satisfaction [TZGX15, MPW07]. It is a soft requirements in the systems and can be given by the system's designer or the end user. Higher service quality or utility comes at the cost of higher resource usage both on the device and the gateway (e.g. energy consumption, bandwidth, etc.). The service quality may be changed at runtime upon the user-specific requirements or in response to the limited resources. For instance if the battery is running low, the device can switch to a lower service quality to preserve the energy.

4.1.3 Operation Modes

IoT devices can support multiple operation modes to be able to adapt to varying conditions of the system at runtime (e.g. battery status). These modes are either because of different offloading schemes (e.g. raw data transmission, partially process on IoT device, or transmitting the final result), or due to different service qualities [STX+16b]. For instance, in case of low battery, a device may switch to a mode with lower quality of input data (e.g. lower sampling rate) to preserve energy. Or when the cost of transmission increases (due to interference), the device may process the data on-board and transmit only the final result. Operation modes differ in terms of resource usage, both on the device (e.g. energy consumption) and on the gateway (e.g. communication bandwidth, processing power, memory, etc.).

The difference between these modes is due to: (i) a change in the provided quality of service, i.e. different processing pipeline or input sampling rate, (ii) a different computation offloading scheme, or (iii) a conjunction of the above parameters (i.e. quality and offloading scheme).

Changing the operation mode of devices at runtime is a control parameter to manage the application requirements, available resources on the device, shared resources on the gateway and

the design objectives of the IoT system. If the new operation mode requires runtime adaptation of the execution flow, first the processing of the current segment of data must finish, and meanwhile the next segment is buffered according to the updated operation mode. If the new operation mode requires changing the offloading scheme, the partially-processed data will be transmitted after the associated pipeline stage (e.g. filtering). If the current segment has completed that particular stage, the new offloading scheme is applied to the next segment.

Due to the scalability and reliability issues specially in edge computing paradigm, the IoT local networks must be self-organizing and self-supported [MSDPC12, GZY+13] and not dependent on the cloud [ZMK+15]. Therefore, the management of IoT systems also needs to be handled at the edge and thus the mode selection for IoT edge devices needs to be decided on the gateway. However, mode selection is challenging due to dynamic changes in the operating conditions of IoT systems and highly constrained processing capabilities of gateway. It needs to be online and it needs to be fast to be both responsive and low-overhead.

4.1.4 Summary of Application Model

This section has presented a general application model for IoT applications in health monitoring domain. This application model can describe a wide range of IoT applications and consists of a sequence of similar stages. It has shown that those stages offer the opportunity to offload the partially-processed data to other devices (e.g. the gateway). Therefore, IoT devices can support multiple offloading levels including 'no offloading' (i.e. process the data fully on the IoT device and only transmit the final results), offloading the partially-processed data, and offloading the whole computation by transmitting the raw data. This section also introduced multiple operation modes for the IoT devices which have different service quality levels and/or different offloading levels. These operation modes are a control parameter that can be leveraged in order to respond to the varying runtime situations such as application requirements, available resources, etc.

In the following sections, fast and low-overhead techniques are presented to select the operation mode of IoT devices at runtime to manage the limited shared resources at the edge while respecting the application requirements and devices' constraints.

4.2 Distributed SQ Management for Internet of Things under Resource Constraints

This section presents a technique to select the operation mode of IoT devices and manage their service quality under different constraints at the edge of network including battery lifetime of devices and shared communication and computation resources of the gateway. The proposed technique is presented in [STX+16b].

4.2.1 Problem Formulation

Consider a local network with a set of N IoT devices, where each device is uniquely identified by an integer value $d \in \{1,\ldots,N\}$.

$$I_d = \Big(X_d, R_d, B_d, e_d, U_d(\cdot), S_d(\cdot), T_d(\cdot), C_d(\cdot)\Big) \tag{4.1}$$

Each IoT device I_d is specified by a tuple, where

- X_d denotes the set of possible *input data rates* of device I_d. They depend on the sensor sampling frequency and data resolution. An IoT device offers its service at M_d different *SQ levels*, with each level having a different input data rate and thus providing a different service quality. For instance, consider an IoT-based heart monitoring device that can capture ECG signals at multiple discrete sampling rates (e.g. 100 Hz, 200 Hz, 300 Hz, and 1000 Hz).
$$X_d = \big\{x_{d_i} \mid i \in [1, M_d]\big\} \tag{4.2}$$

- R_d denotes the set of possible *transmission data rates* of device I_d. They depend on the input data rate x_{d_i} and the computation offloading strategy of the IoT device. Offloading determines how much input data is not processed on the device (on-board processing), but instead transmitted to the gateway (offloaded). An IoT device offers Q_d different *offloading levels*. The transmission data rate $r_{d_{ij}}$ depends on the SQ level i (input data rate) and the offloading level j. It corresponds to the share of input data rate that is offloaded plus the (intermediate) results from the on-board processed data.
$$R_d = \big\{r_{d_{ij}} \mid i \in [1, M_d],\ j \in [1, Q_d]\big\} \tag{4.3}$$

  For instance, consider that the above-mentioned IoT-based heart monitoring device had 3 offloading levels. Level 1 could indicate 'no offloading' and thus only transmits a small amount of results (e.g. the features that were extracted from the signal), independent of the input data rate. Level 2 could be used to offload a certain percentage of the input sample rate. For instance, the device could store a certain amount of input samples in a buffer, then process the buffer and offload all incoming samples during this processing to the gateway. Level 3 could perform some pre-processing on the data, e.g. applying a filter on it, and then send the filter output to the gateway. As the filter would not reduce the data rate, the transmission rate would equal the input data rate, but some of the processing is already done. The particular transmission data rates depend on the device and how it is used, which has to be determined by the user and thus is considered as given in this problem formulation.

- B_d denotes the minimum required battery lifetime (i.e. until the next recharge or battery replacement).

- e_d is the remaining energy in the battery of device I_d.

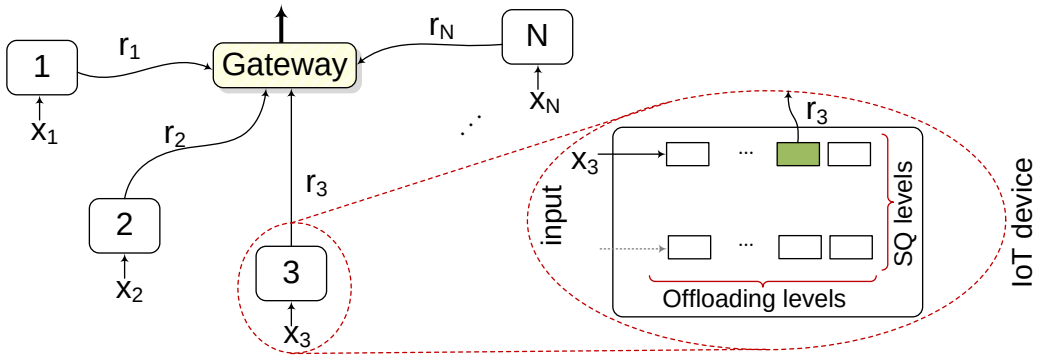


Figure 4.2: Problem model: IoT devices with different SQ and offloading levels resulting in different transmission data rates. The gateway receives and processes the data.

- $U_d(x_{d_i})$ is the utility function that quantifies the utility or service quality (SQ) provided to the user when the device is capturing input data at rate x_{d_i}.
- $S_d(x_{d_i})$ is the power consumption of the device for sensing and capturing data at rate x_{d_i}.
- $T_d(r_{d_{ij}})$ is the power consumption for transmitting data at rate $r_{d_{ij}}$.
- $C_d(i,j)$ is the power consumption for processing at input data rate x_{d_i} under offloading level j.

The battery lifetime of IoT devices depends on 1) the remaining energy and 2) the total power consumption rate:

$$b_{d_{ij}} = \frac{e_d}{S_d(x_{d_i}) + C_d(i,j) + T_d(r_{d_{ij}})} \tag{4.4}$$

where $b_{d_{ij}}$ denotes the expected battery lifetime when the device captures input data at rate x_{d_i}, processes it, and then transmits at rate $r_{d_{ij}}$.

The gateway connects devices to the Internet. It receives data from IoT devices, processes it and transmits the final result to the Internet. The gateway is specified by triple:

$$G = \Big(p(\cdot), R, P\Big)$$

where:

- $p(r_{d_{ij}})$ shows the required processing capability of the gateway to perform the necessary operations on the received data at rate $r_{d_{ij}}$ and offloading level j.
- R is the total available bandwidth of the gateway to receive data from IoT devices.
- P shows the total processing capability of the gateway.

The effect of environment and surrounding devices (e.g. interference) on the transmission can be modeled in R and $T_d(\cdot)$.

The system is summarized in Figure 4.2. The problem that is targeted in this section can be solved by deciding the SQ level i and the offloading level j for each IoT device I_d at runtime,

such that the bandwidth, computation, and lifetime constraints are fulfilled (Eq. (4.5) to (4.7)) and the overall benefit (Eq. (4.8)) is maximized.

$$\text{Bandwidth constraint:} \quad \sum_{\forall d} r_{d_{ij}} \leq R \tag{4.5}$$

$$\text{Computation constraint:} \quad \sum_{\forall d} p(r_{d_{ij}}) \leq P \tag{4.6}$$

$$\text{Lifetime constraint:} \quad \forall\, d:\ b_{d_{ij}} \geq B_d \tag{4.7}$$

$$\text{Optimization goal:} \quad \text{maximize} \sum_{\forall d} U_d(x_{d_i}) \tag{4.8}$$

4.2.2 Proposed Solution

Decomposing the Problem

The targeted problem (see Section 4.2.1) has two sets of constraints: one for IoT devices and one for the gateway. The selected configurations for devices I_d (i.e. x_{d_i} and $r_{d_{ij}}$) should meet the lifetime constraint (Eq. (4.7)). Given the selected configuration for IoT devices, the gateway's constraints to be met are bandwidth and computation (Eq. (4.5) and (4.6)).

The device's constraint depends solely on device parameters. To reduce the search space, the optimization problem can be decomposed into 1) device's problem and 2) network (gateway) problem. In the device's problem, each IoT device excludes those configurations that violate its lifetime constraint to reduce the search space. Then, the network (gateway) problem is solved by considering the reduced search space.

Device Problem

– CoD Matrix Consider a matrix for each IoT device that shows the possible Configurations of that Device (CoD matrix). Each element $\kappa_d[i,j]$ at the intersection of the SQ level i (corresponding to the input data rate x_{d_i}) and the offloading level j contains a pair of the battery lifetime (see Eq. (4.4)) and the transmission data rate, i.e. $(b_{d_{ij}}, r_{d_{ij}}), i \in [1, M_d],\ j \in [1, Q_d]$, as shown in Figure 4.3.

$$\begin{array}{c} \\ \\ M \text{ SQ levels} \end{array} \overbrace{\begin{array}{c} \\ \\ \end{array}}^{Q \text{ Offloading levels}} \begin{array}{c} \begin{array}{ccc} 1 & \dots & \mathrm{Q} \end{array} \\ \begin{Bmatrix} x_1 \\ \vdots \\ x_M \end{Bmatrix} \begin{pmatrix} (b_{11}, r_{11}) & \cdots & (b_{1Q}, r_{1Q}) \\ \vdots & \ddots & \vdots \\ (b_{M1}, r_{M1}) & \cdots & (b_{MQ}, r_{MQ}) \end{pmatrix} \end{array}$$

Figure 4.3: CoD matrix for an IoT device I_d; omitting subscripts d for brevity

Since the available energy of IoT devices changes over time (due to consumption or re-charge), the expected battery lifetime of each configuration (i.e. the first element of each tuple in the matrix) changes over time, which means this matrix needs to be updated periodically.

Example 4.2.1: Consider an IoT device I_d with battery lifetime constraint of $B_d = 40$ that has 5 different SQ levels $X_d = \{100, 200, 300, 400, 1000\}$ and 4 different offloading levels per SQ level. In this example, lets say that each offloading level transmits 20% more data than the previous offloading level, i.e. $r_{i(j+1)} = r_{ij} + 0.2 * x_i$.

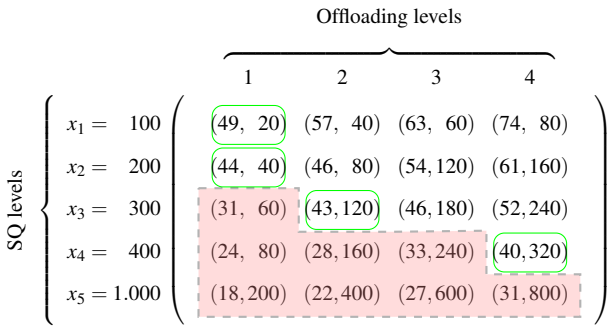


Figure 4.4: CoD matrix for Example 4.2.1

Figure 4.4 shows the CoD matrix associated with Example 4.2.1. The configurations whose expected battery lifetime is less than the constraint (i.e. $b_{d_{ij}} < B_d$) are not feasible and should be excluded from the search space. The infeasible configurations (those with expected battery lifetime less than 40) are shown in shaded area. All other configurations are feasible as they meet the battery constraint.

– Properties of CoD Matrix The CoD matrix of most applications may have the following property. However, it should be emphasized that neither the formulated problem nor the presented solution is restricted to this property.

Property 4.2.1: For each column of the CoD matrix, the elements are in an increasing order in terms of transmission data rate, but in an decreasing order in terms of estimated battery lifetime.

The nature of this property can be understood intuitively. For a given offloading level (i.e. column), an increased input data rate leads to increased transmission data rate and increased on-board processing requirements. This has negative effects on the power consumption for sensing data S_d, transmitting data T_d, and processing data C_d, all of which have a negative effect on the battery lifetime (see Eq. (4.4)).

– Reducing Problem Size Although all the elements outside the shaded area meet the battery lifetime constraint and are feasible configurations for the problem, some of them are intuitively inefficient. For instance, all the elements in the first row of the above matrix (Figure 4.4) provide the same SQ and utility to the user. They are just different in the the amount of computation offloading. Selecting the element with the smallest transmission data rate (marked with a green circle in Figure 4.4) would result to the optimal solution as is proved in the following.

Theorem 1. *If two elements from the same row i are feasible, $\kappa(i, j_1)$ and $\kappa(i, j_2)$ where $j_1 < j_2$, then $\kappa(i, j_2)$ never outperforms $\kappa(j, j_1)$ in the optimal solution.*

Proof. The theorem is intuitively obvious. Since $\kappa(i, j_1)$ and $\kappa(i, j_2)$ offer the same SQ level i and utility $U(x_i)$ their direct contribution to the objective goal (see Eq. (4.8)) does not differ. However, imposing less computation to the gateway (i.e. less transmitted data) may leave more room for other IoT devices, so that the gateway can possibly choose another feasible configuration with higher utility and correspondingly more computation and offloading. □

Inspired by theorem 1, for each IoT device, one can only consider the leftmost feasible element of each row (i.e. the one with the least data transmission rate):

$$\forall\ 1 \leq i \leq M_d \quad r_{d_i} = \min_{b_{d_{ij}} \geq B_d} (r_{d_{ij}}) \tag{4.9}$$

After building/updating the CoD matrix, each device finds the efficient feasible configurations (EFC), from Eq. (4.9), and forms a set of pairs containing 1) utility of each EFC, and 2) the transmission data rate r_{d_i} of each EFC. In the previous example shown in Figure 4.4, let us assume the utility of SQ levels is as $U(100) = 50$, $U(200) = 90$, $U(300) = 110$, $U(400) = 150$, and $U(1000) = 310$. Then the EFC set of this device is shown in Figure 4.5.

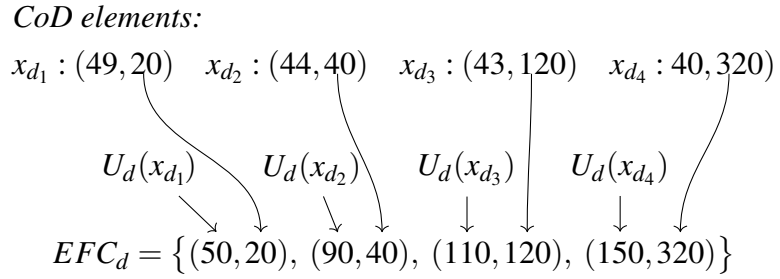


Figure 4.5: An example of EFC set

Each IoT device periodically checks its remaining energy (i.e. e_d), updates the CoD matrix, and updates the EFC set. In case that the EFC set changes, the device sends the new set to the gateway, where it is used to solve the *Network problem*. Note that the EFC set only contains feasible solutions, i.e. the number of entries in the EFC set of a particular device may change over time.

Network (Gateway) Problem

– Integer Linear Programming (ILP) Formulation For each IoT device, there is a set of efficient feasible configurations (EFC), showing the offered utility and transmission rate to the gateway. Given all EFC sets of all IoT devices, the gateway needs to solve the SQ management

problem. The gateway extends each EFC set by including the processing requirement of the associated transmitted data (i.e. $p(r_d)$). The resulting EFC'_d set is shown in Eq. (4.10).

$$\begin{aligned} EFC'_d &= \left\{ \left(U_{df},\ r_{df},\ p_{df}\right)_{1 \le f \le \left|EFC_d\right|} \right\} \\ U_{df} &= U_d(x_{d_f}) \text{ // Utility of device } I_d \text{ for } f\text{-th EFC entry} \\ r_{df} &= r_{d_f} \text{ // Corresponding transmission rate (see Eq. (4.9))} \\ p_{df} &= p(r_{d_f}) \text{ // Corresp. gateway processing requirement} \end{aligned} \tag{4.10}$$

The gateway problem is to select one and only one item from each set such that the overall utility is maximized and the gateway's constraints are met. This can be formulated as an Integer Linear Programming (ILP):

$$\max \quad \sum_d \sum_f (U_{df} \times w_{df}) \tag{4.11}$$

$$\text{subject to} \quad \forall d: \quad \sum_f w_{df} = 1 \tag{4.12}$$

$$\sum_d \sum_f r_{df} \times w_{df} \le R \tag{4.13}$$

$$\sum_d \sum_f p_{df} \times w_{df} \le P \tag{4.14}$$

where

$$w_{df} = \begin{cases} 1 & \text{if } f\text{-th EFC element from } d\text{-th device is chosen} \\ 0 & \text{otherwise} \end{cases}$$

This optimization problem corresponds to the Multidimensional Multiple-Choice Knapsack Problem (MMKP) and is NP-hard, thus computationally intractable [MJS97]. Since the constraints in Eq. (4.13) and (4.14) are inequalities, the proposed technique in [Pis95] for merging multiple constraints does not apply to this problem. In the following, a solution to this problem is presented based on a dynamic programming (DP) approach.

Definition 4.2.1: The term "instance of problem" is used to refer to the configurations of the gateway problem (i.e. the input data for Eq. (4.11) to (4.14)). Any change in the EFC sets makes it another instance of the problem.

Dynamic Programming Solution

– Intuition Although one can design and introduce heuristic approaches, a dynamic programming solution seems more appropriate for this problems because:

- It is needed to solve different instances of the problem where subsequent instances do not differ substantially. This gives the dynamic programming approach the opportunity

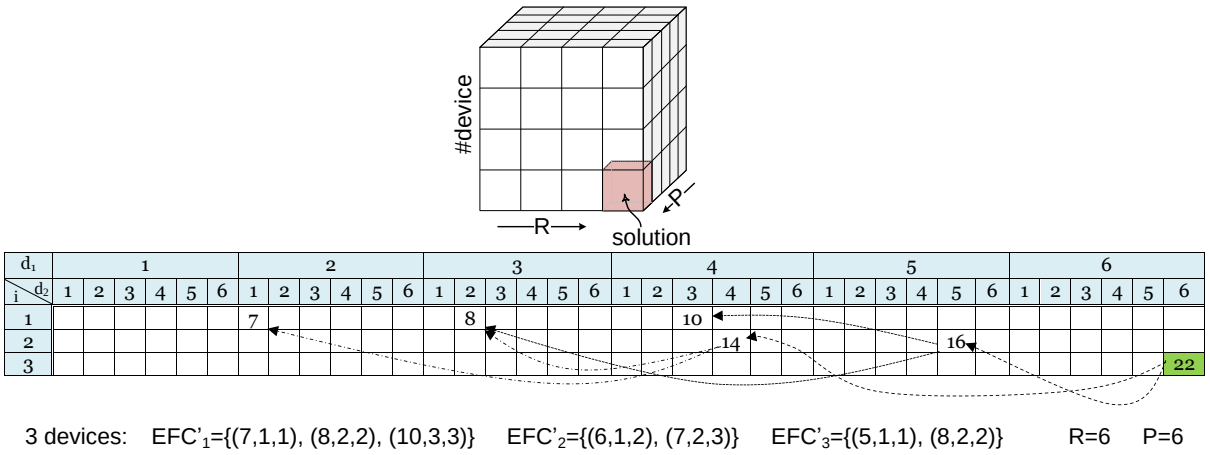


Figure 4.6: The dynamic programming table (top), and an example solved using proposed approach (bottom).

to possibly reuse some computations that are performed in the previous instance of the problem.

- It provides a quick and optimal solution to the problem.

Since this problem has two dimensions (i.e. two constraints including data rate and processing power), the table to form the basis for the dynamic programming approach has 3 dimensions: one for the number of devices, and two others for the constraints.

– Formulation & Example Let $Z(d,R,P)$ denote the maximum overall utility that can be obtained from the first d devices while the constraints on the data rate and processing capability of gateway are R and P, respectively.

For the d^{th} device, it is needed to choose one of its configurations from its EFC set whose size is $\left|EFC_d\right|$. Considering the f^{th} element of the EFC set, its utility, data rate, and processing requirements are U_{df}, r_{df} and p_{df}, respectively. For the f^{th} configuration, the algorithm first finds the overall utility of a solution with $d-1$ devices whose overall required bandwidth and processing resources are $R-r_{df}$ and $P-p_{df}$, respectively. Then, it will be added to the provided utility by f^{th} configuration (i.e. U_{df}). All the possible configurations of device d are investigated and the one that maximizes the overall utility is selected. Therefore, this algorithm leads to the global optimum solution. It should be noted that the order of devices does not matter. Equation (4.15) shows the recurrence relation which is proposed to calculate $Z(d,R,P)$ (as explained above):

$$Z(d,R,P) \;=\; \max_{1\le f\le|EFC_d|}\Big\{Z\big(d-1,R-r_{df},P-p_{df}\big)+U_{df}\Big\} \tag{4.15}$$

Example 4.2.2: Figure 4.6 shows an example with three devices whose extended EFC sets are as follows, respectively:

$$EFC'_1=\{(7,1,1),(8,2,2),(10,3,3)\}$$
$$EFC'_2=\{(6,1,2),(7,2,3)\}$$
$$EFC'_3=\{(5,1,1),(8,2,2)\}$$

Algorithm 1: The proposed approach based on top-down DP

Inputs : M EFC sets, and constraints P and R
Function $Z(d,R,P)$
  if $R \leq 0$ *or* $P \leq 0$ **then**
    return $-\infty$;
  else if $d == 0$ **then**
    return 0;
  end
  for $f \leftarrow 1$ **to** $|EFC_d|$ **do**
    if $r_{df} \leq R$ and $p_{df}) \leq P$ **then**
      if $Tb[d-1][R-r_{df}][P-p_{df}] == -\infty$ **then**
        $Tb[d-1][R-r_{df}][P-p_{df}] \leftarrow Z(d-1,R-r_{df},P-p_{df}))$;
      end
      if $Tb[d-1][R-r_{df}][P-p_{df}]+U_{df} > Tb[d][R][P]$ **then**
        $Tb[d][R][P] \leftarrow Tb[d-1][P-r_{df}][P-p_{df}]+U_{df}$;
      end
    end
  end
call $Z(N,R,P)$

The constraints of the gateway are $R=6$ and $P=6$. The optimal solution is $Z^* = Z(3,6,6)$. Based on Eq. (4.15):

$$
\begin{aligned}
Z(3,6,6) &= max\{5+Z(2,6–1,6–1), 8+Z(2,6–2,6–2)\} \\
&= max\{5+16,\ 8+14\} = 22 \\
Z(2,5,5) &= max\{6+Z(1,4,3), 7+Z(1,3,2)\} = 16 \\
Z(2,4,4) &= max\{6+Z(1,3,2), 7+Z(1,2,1))\} = 14
\end{aligned}
$$

In this example, the cell $Z(1,3,2)$ is referred twice, which simply shows the benefit of using dynamic programming to avoid recomputing the same sub-problem repeatedly. A top-down dynamic programming is preferred for the proposed approach as some of the sub-problems never get examined at all (see Figure 4.6). For instance, among all the elements corresponding to device N (3 in the above example), only computing $Z(N,R,P)$ (or $Z(3,6,6)$ in the example) is required. Hence, to avoid computing unnecessary cells, a top-down DP is implemented that is based on the recursive relation and on *memoizing* the computed sub-problems in a linked list structure, which reduces the memory usage as well as computation runtime.

Algorithm 1 illustrates the utilized top-down dynamic programming approach to find the optimum solution to the SQ management problem based on the recursive relation in Eq. (4.15).

Reusing Sub-solutions

Definition 4.2.2: Mask the change: Any change in the EFC set of device d that does not change the cells of the table corresponding to device $d+1$ (i.e. $[d+1,*,*]$) is masked, and hence does not propagate to other cells of the table. A masked change does not affect the solution.

As mentioned earlier, the EFC set of each node may change over time as its available energy changes due to consumption or battery re-charge. Some of those changes can be completely masked. However, the unmasked changes still benefit from pre-computed cells of the solution table.

– Classifying the Possible Changes Possible changes in an EFC set can be categorized into three different class:

1. **ADD:** A new item is added: It happens only when the battery is re-charging and the available energy (i.e. e_d) increases. Then, a SQ level that was not present in the EFC set is added to it. For example in Figure 4.7(a), the highest SQ level (last row) is included as one of its configurations meets the battery lifetime constraint.
2. **REM:** An existing item is removed: It happens when the battery depletes and the available energy decreases. Then, a SQ level that can no longer fulfill the battery lifetime constraint (under the new circumstances) is completely excluded and its corresponding items are removed from the EFC set as shown in the example of Figure 4.7(b).
3. **CHANGE:** An existing item changes its second entry (i.e. r). It means that the item still offers the same SQ level (the same utility value U), but with a different data transmission rate. In other words, the item is replaced with another item from the same row in the CoD matrix but from a different column as shown in Figure 4.7(c) and Figure 4.7(d). It happens under two different circumstances:
   - **DEC:** Due to battery recharge, a new element in the CoD matrix is added to the feasible region of a row (see Figure 4.7(c)). Therefore, the corresponding item in the EFC set is replaced with a new one which has the same utility (they are both in the same row of CoD) but with less data to transmit (i.e. more on-board processing): $(U,r^*) \rightarrow (U,\hat{r}) : r^* > \hat{r}$.
   - **INC:** Due to the energy consumption, the previous EFC item is not feasible anymore and another element in the CoD matrix from the same row is selected to be in the EFC set (see Figure 4.7(d)). The first entry (i.e. utility) is the same, but the data rate is increased (i.e. more computation offloading): $(U,r^*) \rightarrow (U,\hat{r}) : r^* < \hat{r}$.

Any change in the EFC set falls into one of the above categories. It should be noted that multiple changes can happen at the same round of updating the CoD matrix.

– When and How to Reuse? This subsection investigates different classes of changes, and tries to propose efficient solutions to reuse the sub-solutions that were computed before the changes.

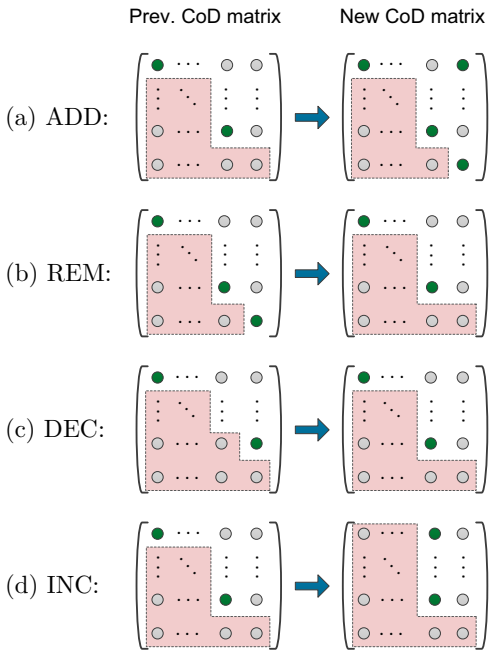


Figure 4.7: Different possible changes in EFC set of an IoT device

Among all the possible changes, some do not change the optimal solution. Hence, the previous solutions remain the same with no need to solve the problem again. Some cases can reuse the previous solution partially, and others need to rerun the algorithm.

- ADD: In case that the change is an 'ADD', the previously computed cells of table are completely reused, but some new cells need to be computed. For instance, assume that in Example 4.2.2, a new item is added to the EFC set of device 2, i.e. $EFC_2' = EFC_2' \cup \{(8,3,4)\}$. It affects computing of $Z(2,5,5)$ and $Z(2,4,4)$ as following:

$$\begin{aligned} Z(2,5,5) &= max\{6+Z(1,4,3), 7+Z(1,3,2), 8+Z(1,2,1)\} \\ &= 16 \\ Z(2,4,4) &= max\{6+Z(1,3,2), 7+Z(1,2,1), 8+\boxed{Z(1,1,0)}\} \\ &= 14 \end{aligned}$$

  As it is shown, only one new table cell is needed to be computed (i.e. $Z(1,1,0)$) and all the other required cells are reused from previously computed cells.

- REM:

  1. If the removed item was not the item that was selected as the solution in the previous setup, then the optimal solution does not change and there is no need to re-run the algorithm. This case can be checked and masked at the IoT device without the need to inform the gateway.

  2. Based on Property 4.2.1, the removed item had the highest utility and data rate. In this case, the solution to the new setup is already in the computed table, as the new instance of the problem is a subset of the previous instance.

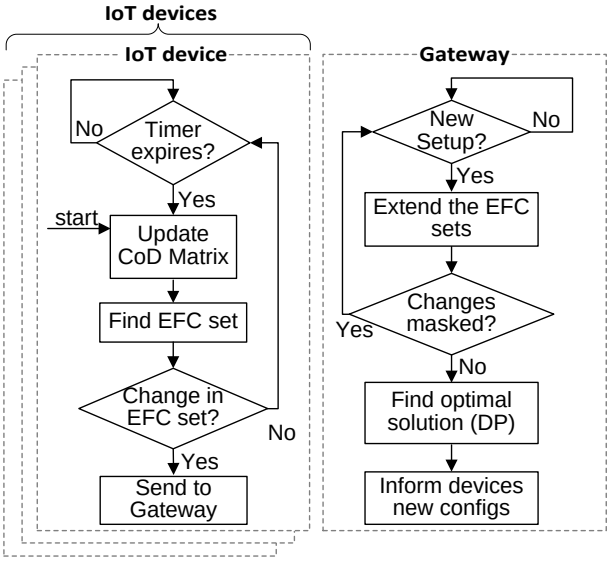


Figure 4.8: Simplified flow of SQ management

- INC: The updated item in the EFC set is either the selected item in previous instance or a non-selected item.

  1. In the former case, if the amount of increase in bandwidth (i.e. $\Delta r = r^* - \hat{r}$) and processing (i.e. $\Delta p = p(r^*) - p(\hat{r})$) are less than available resources that are left (i.e. $R - \sum r_d^*$ and $P - \sum p(r_d^*)$), then the solution does not change.
  2. In the latter case, the optimal solution does not change, hence there is not need to re-run the algorithm to find it.

- DEC: If this change happens for device d_1 ($1 \leq d_1 \leq N$), only the precomputed cells of devices $1 \leq d_2 < d_1$ can be reused, and the other referred cells need to be updated. Let us assume that in Example 4.2.2, the EFC set of device 2 witnesses a 'DEC' change as the second item becomes $(7,2,2)$. All the cells containing $Z(1,*,*)$ remain unchanged, but the others are updated as following:

$$\begin{aligned} Z(2,5,5) &= max\{6+Z(1,4,3), 7+Z(1,3,3)\} = 17 \\ Z(2,4,4) &= max\{6+Z(1,3,2), 7+Z(1,2,2)\} = 15 \\ Z^* = Z(3,6,6) &= max\{5+17, 8+15\} = 23 \end{aligned}$$

This shows that in the proposed solution not only the sub-problems of each instance can be reused to avoid re-computation (by means of dynamic programming), but different instances of the problem (i.e. after changes in the setup) can still benefit from previously computed sub-solutions.

Figure 4.8 shows a simplified flow of the solution including the device flow and gateway solution flow.

4.2.3 Use case: IoT in Healthcare Monitoring

This subsection uses on ECG arrhythmia detection, serving as the application around which a realistic use case is formulated for evaluating the efficiency of the proposed IoT resource management solution. The following subsections present the use case including the details of a monitoring IoT device and a network of devices with realistic models for gateway and IoT devices.

ECG Analysis & Arrhythmia Detection

ECG provides essential information about the status of the heart which is critical for prevention, early diagnosis, and treatment of cardiovascular diseases [LZP+12] as well as wellness applications [BMB+15]. In this work, ECG signals are used to detect heart arrhythmia, i.e. irregularly fast or slow heart beats which may lead to strokes or heart failure [SCL+05]. The ECG analysis flow of Figure 4.9 is implemented on the IoT device, which receives raw ECG data and detects arrhythmia (if any).

1. **Filtering:** The initial step includes signal acquisition and filtering. A band-pass FIR filter is used to remove baseline wander (< 1 Hz) and power line noise (50 Hz).
2. **Segmentation and heart beat detection:** For segmentation of the ECG signal, it is first needed to detect the R peak (see the annotated ECG signal in Figure 4.9). The periodicity of the heart beat is not constant and varies due to different reasons such as physical activity, stress level, etc. Hence, a window of samples are examined to locate the peak. The window size depends on the sampling rate. The detected R peak is used as the start of a new segment.
3. **Feature extraction:** Features extraction of the heart beat is performed through Discrete Wavelet Transformation (DWT) which is very popular for ECG signal processing due to the fact that it is lightweight and capable of providing time and frequency information simultaneously [BAAR12, MAM13]. This is essential when analyzing signals whose frequency response varies in time, such as the ECG signal, and thus time localization of the frequency spectral components is required.
4. **Diagnosis:** For the final stage, a Support Vectors Machine (SVM) classifier is used to capture non-linear relationships of the feature space representing the target classification problem [HPS+15]. The execution of the classifier concludes whether the processed heart beat exhibits any signs of arrhythmia. Figure 4.9 depicts an example ECG signal exhibiting normal and arrhythmic heart-beats.

The examined IoT scenario is located at clinical ward which welcomes a large number of patients that should all be monitored simultaneously. Wearable ECG analysis devices communicate with the gateway through low-power low proximity wireless communication which in the deployed scenario is Bluetooth Low Energy.

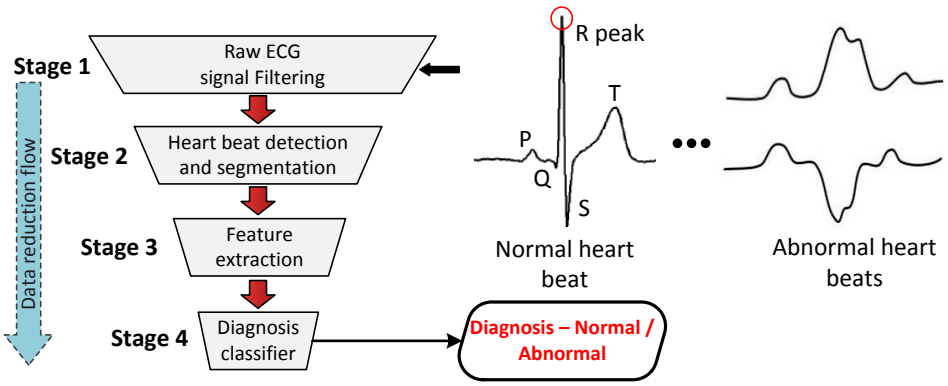


Figure 4.9: ECG analysis flow

SQ and offloading levels

The presented design is pipelined in the sense that it produces valid results at the end of every processing stage. For example, if the device is instructed to execute up to the feature extraction process then the output of the flow is the Discrete Wavelet Decomposition of the input signal. This result can be used in subsequent processing if transmitted to other devices.

This property of the flow enables the system to support offloading of processing to the gateway. All of the aforementioned pipeline stages can be executed on the gateway. Therefore, processing can be performed up to an arbitrary pipeline stage on the IoT node, then transmit its output to the gateway, and resume the execution of pipeline there. The gateway decides the level of offloading for each IoT node of the system by solving the SQ management problem.

As far as different SQ levels are concerned, they correspond to different sampling frequencies of the ECG signals. Signals sampled at a higher frequency offer more detailed description of the monitored ECG. The increased signal resolution enhances further analysis and diagnosis by medical experts thus leading to increased SQ and utility for the patient. Combinations of SQ levels and offloading levels (i.e. after which pipeline stage computation is offloaded to the gateway) result in different data rates for input (x_d in Eq. (4.2)) and output (r_d in Eq. (4.3)) of the device. Table 4.1 summarizes these values for combinations of SQ levels and offloading levels stages for the ECG monitoring prototype.

Table 4.1: Input data rates and transmission data rates for different SQ levels and offloading levels

			Transmission rate r_d [B/s] for offloading after a certain pipeline stage			
SQ level	Sampling freq. [Hz]	Input data rate x_d [B/s]	Stage 1	Stage 2	Stage 3	Stage 4
1	180	720	720	360	104	1
2	360	1440	1440	720	192	1
3	720	2880	2880	1440	372	1
4	1440	5760	5760	2880	564	1
5	2000	8000	8000	4000	1024	1

Due to the increased sampling frequency of higher SQ levels, an increase in input and output data rate of each stage is observed. For example, if the window of data analysis W is 256 points wide at sampling frequency of 360 Hz, then the corresponding window rises to 512 data points at double the sampling frequency. Inevitably, this affects all other pipeline stages given that they operate on greater amount of data. The only exception is the result of the analysis flow (Stage 4), which is always one value that corresponds to the diagnosis label of the processed heart beat.

Stages 1 to 3 of the flow have been designed to operate on a variable-sized input data window while a classifier model (stage 4) was trained for each SQ level. Therefore, there is an instance of the pipeline for each SQ Level, which operates on different amount of data. To comprehend how this fact affects the resources needed for the execution of each combination of SQ level and pipeline stage, the execution of the flow is profiled on the target IoT device. Figure 4.10 summarizes the percentage of execution of each processing stage over one minute for increasing SQ levels.

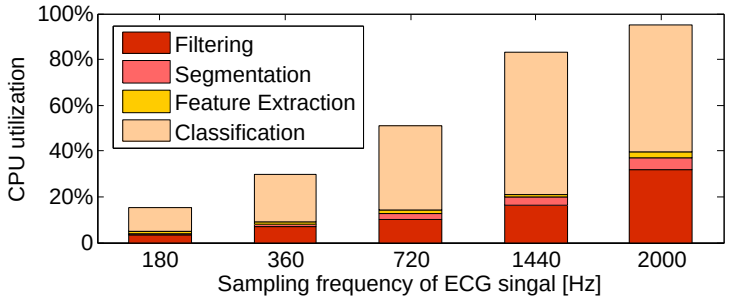


Figure 4.10: CPU utilization of ECG analysis stages

A expected, a higher SQ level comes at the price of increased computational requirements. Figure 4.10 also shows the computational effort that is offloaded to the gateway for the different offloading levels, as the breakdown for the computational complexity of all pipeline stages is shown. In all cases, the most computationally intensive stage is the diagnosis part due to its complex structure in an effort to provide accurate predictions. On the contrary, beat segmentation and DWT do not occupy the CPU for prolonged period. The rest of the time the CPU remains idle, which is the major reason of power consumption variations over different SQ levels.

4.2.4 Evaluation and Results

Experimental Setup

To construct the ECG analysis flow, actual patient ECG data records are used from *MIT-BIH Arrhythmia Database* [MM01]. The annotated signals by medical experts are used for training the machine learning tools. Original signals were sampled at 360 Hz. Down- and up-sampling has been used to generate ECG signals of differing sampling rates.

Experimental analysis has been performed by simulating IoT network topologies of up to 10 nodes. To efficiently model the characteristics of the case study for different SQ and offloading levels, the real execution of the ECG analysis flow is profiled on an Intel Quark SoC, already proposed and used for wearable IoT devices [AB15, AMJ15]. The outcome of this profiling campaign summarizes the computational requirements, expressed in CPU utilization, of each combination of SQ- and offloading-level.

To conduct the experiments, a combination of experimentally derived data enhanced with nominal data from data-sheets of commercial devices is used for the model parameters values. Regarding the available energy of the IoT device (e_d), a battery consumption model of each IoT node is composed based on the instrumented CPU utilization. More specifically, energy consumption of ECG acquisition $S_d(x_{d_i})$ was calculated based on [Han13]. Bluetooth Low Energy (BLE) is used for communication between IoT devices and gateway. Power consumption value of data transmission (i.e. $T_d(r_d)$) is $0.153\,\mu W$ based on [Smi11] and transmission latency is $4\,\mu s/bit$ [SHNN12]. Since BLE exploits an adaptive frequency hopping mechanisms, the probability of interference is very low. To complete the battery model of the IoT nodes, a rechargeable Lithium-Ion coin cell battery is chosen with a nominal capacity of up to $420\,mAh$ [DMHL12]. A realistic discharge model is used for the battery using [ZLSX13] for various values of discharge currents to evaluate the available energy for Eq. (4.4).

An ARM Cortex-M3 device is considered as the gateway [MPV11]. The energy consumption values were acquired by profiling the execution of the ECG analysis flow for all combinations of SQ levels and processing stages to measure the values of the required parameters, e.g. $C_d(\cdot)$, $p(r_{d_{ij}})$, etc.

The final key component of the system model is to determine the utility functions of each device. The SQ value of each combination of SQ level and ECG processing stage was set proportional to the ratio of the sampling frequency of the ECG signal divided by the maximum available ECG sampling frequency (2 kHz). It is possible to create more complex profiles of IoT devices by allowing the user to specify a factor of how important high SQ levels are for this device.

Overhead Analysis

The proposed DP approach is implemented on the ARM Cortex-M3 microcontroller that is used as the gateway platform. The measurements include the number of CPU cycles that the proposed solution needs to calculate the optimal result and compare it against the brute-force (BF) method in Figure 4.11. As the number of devices increases, the algorithm execution time for BF increases exponentially whereas the execution time of the proposed algorithm increases moderately (note the logarithmic scale of the Z-axis). For instance, having ten IoT devices with each one having five feasible configurations, the proposed algorithm finds the optimal solution in 0.4 seconds when the gateway is running at 100 MHz (i.e. $\frac{39{,}480{,}530\,[cycles]}{10^8\,[Hz]} \simeq 0.4\,s$), while the BF methods takes around 56 seconds (i.e. $\frac{5{,}587{,}270{,}875\,[cycles]}{10^8\,[Hz]} \simeq 56\,s$). The generally short execution time of the proposed solution and its good scalability show its suitability for online

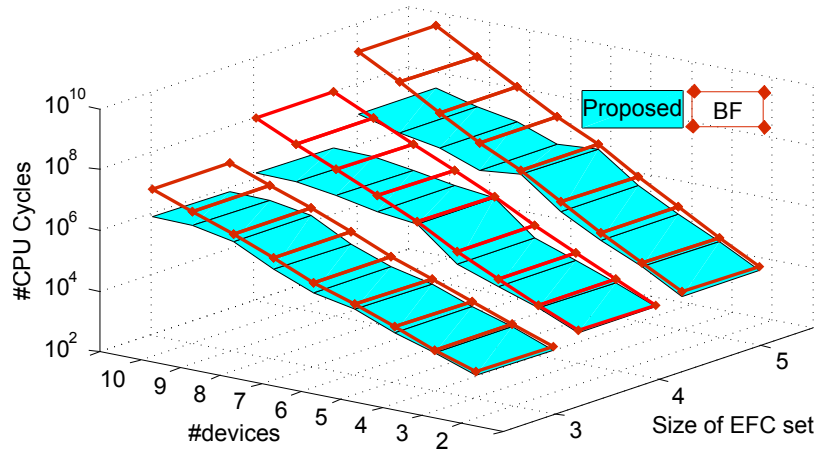


Figure 4.11: The execution time of the proposed method compared to the BF method for different number of devices and different sizes of EFC sets.

and dynamic scenarios of IoT networks where the number of active devices and their feasible configurations change over time repeatedly.

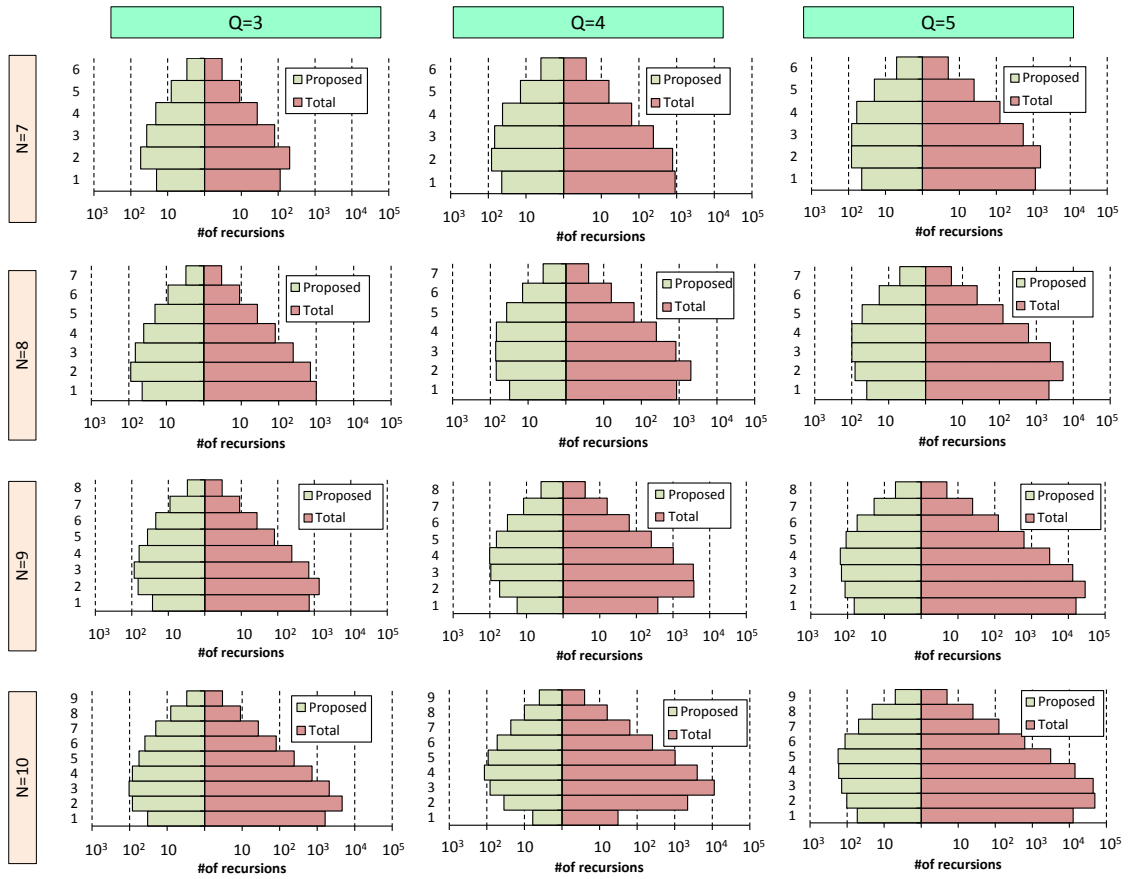


Figure 4.12: Total number of recursive calls of function (see Section 4.2.2), and number of recursions in the proposed algorithm at different levels of the tree

As explained in Section 4.2.2, the proposed solution is based on a recursive function and its sub-problems may occur repeatedly. The execution time is reduced by *memoizing* the answers of sub-problems to avoid recomputing them over and over. Figure 4.12 shows a detailed analysis

of the proposed algorithm with $N = 7, \cdots, 10$ devices and $Q = 3, \cdots, 5$ SQ levels. It shows the average number of recursive function calls if the sub-problems are not stored (named 'Total'), and the average number of recursive function calls in this approach. Note that the values on the X-axis are presented in logarithmic scale. The Y-axis shows the recursion level where the function call happened.

To make it clear, see Figure 4.13 which shows the function calls as a tree. The root is at the N^{th} level, where N is the number of IoT devices (see Section 4.2.1). For instance, in Figure 4.13 the number of function calls at level 9 is equal to 4. Now consider the green node which is labeled with (1). This node is called four times, however, in the proposed solution it is called only once and the result is stored in the table for later references. The number of function calls at the N^{th} level is always 1 and is not shown in Figure 4.12.

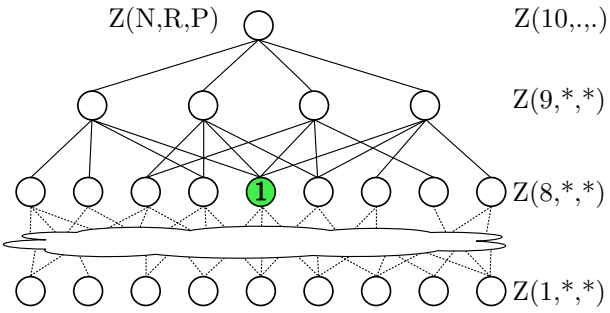


Figure 4.13: function calls tree

Figure 4.14 shows the time interval between two successive re-executions of the algorithm (i.e. time between two problem instances), which is triggered after a change in the set of feasible configurations. As the number of devices increases in the case study, the average time between re-executions decreases (i.e. it is needed more frequently).

Comparison to unsupervised devices

The proposed solution is also compared to the system that operates with no SQ management by the gateway (called 'unsupervised'). The experiments are conducted with two scenarios for the unsupervised system.

In the first scenario, devices operate at the highest SQ level and with no offloading to the gateway. The battery lifetime constraint is assumed to be 20 hours (1200 minutes) and assume that designers can choose a battery out of three ranges with small (260–320 mAh), medium (320–380 mAh) and high (380–420 mAh) capacity. For a varying number of IoT devices in a range of 2 to 10, the achieved battery lifetime of the unsupervised system and the proposed solution are compared. Figure 4.15 shows the average achieved battery lifetime. The unsupervised system fails to meet the constraint even when the battery capacity is high, while the proposed technique always respects it. In some cases, the battery lifetime of the system may exceed the battery constraint (i.e. device operates a bit longer). The reason is that as the number of devices increases, the proposed solution offloads more data and the constraints of the gateway force the devices to decrease their SQ level such that all devices can benefit from offloading. For instance, in Figure 4.4 instead of $x_4 = 400$ and $r_{4,4} = 320$, the algorithm has to select $x_3 = 300$ and $r_{3,2} = 120$ in order to meet the gateway's constraints. This leads to lower energy consumption and longer battery lifetime.

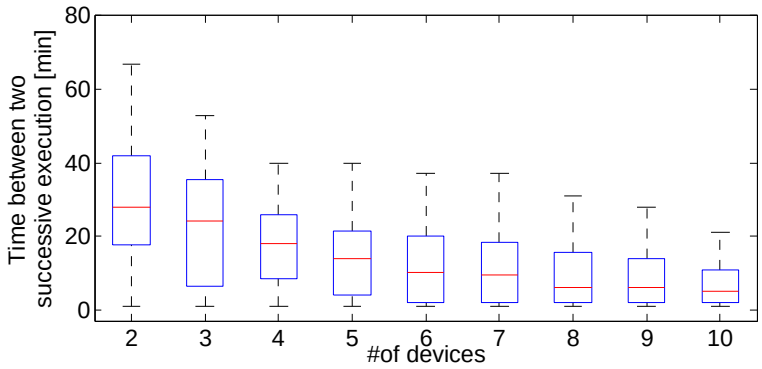


Figure 4.14: The time intervals between successive re-execution of algorithm in the case study

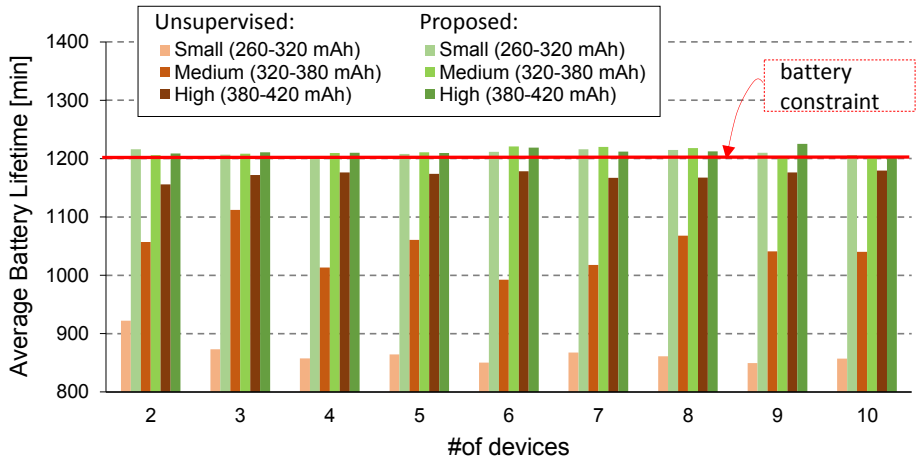


Figure 4.15: The average battery lifetime of devices in the system and the unsupervised system for different battery sizes

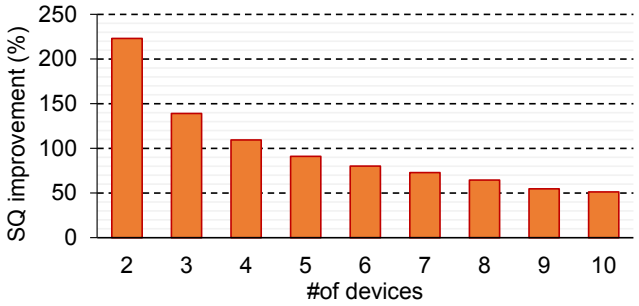


Figure 4.16: The accumulated SQ (utility) in the system compared to the unsupervised system for different number of devices

In the second scenario, the IoT devices in the unsupervised system select a low SQ level to make sure that the battery lifetime constraint is met. Figure 4.16 presents the average achieved SQ with the proposed solution compared to the unsupervised system. In a system with only a few devices (i.e. 2 or 3), the gateway resources are not scarce and therefore, this solution selects a high SQ level for the IoT devices. As the number of devices increases, the gateway's resources become saturated and thus this solution assigns a low SQ level to IoT devices. By employing this solution, the overall SQ of devices is at least 50% more than unsupervised system.

4.2.5 Summary of Distributed SQ Management Technique

This section studied the problem of SQ management in IoT systems, where the IoT devices can provide different SQ levels and can offload a share of their workload. The SQ management has to fulfill constraints for the battery lifetime of IoT devices, communication bandwidth to the gateway, and processing capability of the gateway (for offloading). This section presented an ILP formulation for this problem and decomposed it into separated device and gateway problems. This allowed to reduce the search space and to distribute a part of the problem calculation to the IoT devices. The proposed solution benefited not only from reusing its sub-solutions (based on dynamic programming), but also from previous instances of the problem. The effectiveness of the proposed approach has been demonstrated by using a case study of ECG processing in a personal healthcare monitoring application. The experiments showed that this solution improves the overall SQ (i.e. utility) by 50% compared to an unsupervised system while both meet the constraints.

4.3 Novel Memorization for Fast and Efficient Operation Mode Selection

This section aims at improving the efficiency for operation mode selection where the system optimization objective is in the form of max-min goal. It also presents a novel memoization technique for the resource allocation techniques which are based on dynamic programming. This novel technique is especially advantageous when the shared resources are divisible. The contributions of this section are also based on [STB+17b].

As described in Chapter 5, IoT devices support multiple operation modes to be able to adapt to varying conditions of the system at runtime (e.g. battery status). The operation modes of IoT edge devices must be selected at runtime such that the efficiency (e.g. energy efficiency) among all devices is improved. Since optimizing the *overall* efficiency or reaching maximum efficiency might lead to poor efficiency in some IoT devices (e.g. starvation where some devices consume most of the shared resources and starve all other devices), this section considers the *'max-min'* goal for optimization. In this problem, the lowest efficiency among all IoT edge devices shall be maximized. Therefore, the overall efficiency among all the devices is improved.

The novel contributions of this section are as follows:

- It presents a fast and low-overhead scheme to select the operation modes of IoT devices. The scheme is based on dynamic programming.
- It introduces a novel memoization technique for the dynamic programming scheme in which each sub-problem determines the solution to a range of sub-problems instead of only one. It reduces the required memory to store the sub-problems significantly, which is important for IoT systems.
- It identifies the properties of the mode selection problem and then leverages them to *prune* the search space, which reduces both memory and execution time overhead.

4.3.1 Problem Formulation

Gateway Model

The constrained resources of the gateway are shared between the connected IoT devices. $\mathscr{R}$ is the vector of length g showing the shared resources on the gateway where $\mathscr{R}[j]$, $j = 1, \cdots, g$, shows the amount of resource j on the gateway that is dedicated for edge processing and that is thus shared among all IoT devices. For instance, the communication bandwidth ($j = 1$), the processing capability ($j = 2$), and internal memory ($j = 3$) can be considered as scarce resources. See Example 4.3.1 for more details. This vector is determined after considering the other functionalities of the gateway, i.e. resources that are needed for the operating system, control/management tasks etc. are excluded from it. Only resources that go beyond these requirements are dedicated to the edge computing purpose. Note that the available resource of the gateway may change at runtime, but it is assumed that it is not more often than one round of the problem.

Definition 4.3.1: For resource vectors $\mathscr{R}_1$ and $\mathscr{R}_2$, let us define $\mathscr{R}_1 \leq_v \mathscr{R}_2$ if and only if for all types of resources ($j = 1, \cdots, g$), it holds that $\mathscr{R}_1[j] \leq \mathscr{R}_2[j]$. This definition is used later for determining the feasible solutions.

IoT device Model

Consider a local network with a set of N IoT edge devices connected to the gateway (see Figure 4.17). Each edge device is uniquely identified by an integer value $d \in \{1, \cdots, N\}$ and is specified by a tuple:

$$I_d = \Big(Q_d, L_d, E_d(\cdot,\cdot), R_d(\cdot,\cdot), f_d(\cdot,\cdot)\Big) \tag{4.16}$$

where

- Q_d denotes the number of possible *SQ levels* of device d. At each point of time, the device is operating at one SQ, $q_d \in \{1, \cdots, Q_d\}$, with each level having a different input data rate and thus providing a different quality of service. The SQ depends on the sensor sampling frequency and data resolution (see Chapter 5).
- L_d denotes the number of different *offloading levels* that the application on the device d offers. Offloading level determines how much of data processing is not performed on the edge device (on-board processing), but instead transmitted to the gateway (offloaded). At each point of time, device d operates at one offloading level $l_d \in \{1, \cdots, L_d\}$.
- $E_d(q_d, l_d)$ is the average power consumption of device d when offering the SQ level q_d and operates under offloading level l_d.
- $R_d(q_d, l_d)$ is the vector of length g showing the amount of shared resources that device d requires on the gateway to be able to operate at SQ level q_d and offloading level l_d. It is

assumed that for all the SQ and offloading levels $R_d(q_d, l_d) \leq_v \mathscr{R}$, otherwise the operation mode would not be feasible at all, as it would require more resources than are available.

– $f_d(q_d, l_d)$ denotes the efficiency factor of device d when it offers the SQ level q_d and operates under offloading level l_d. This is a general factor defined by the designer that can support different metrics, e.g. energy efficiency. Depending on the system objectives, the designer can define this factor by considering critical parameters such as energy consumption, service quality, battery lifetime, etc. The following definition is used in this section which depends on the provided SQ and consumed energy:

$$f_d(q_d, l_d) = \frac{U(q_d)}{E_d(q_d, l_d)} \tag{4.17}$$

where $U(\cdot)$ is a strictly increasing function which shows the quality of service and user satisfaction (i.e. utility [TZGX15]) for each SQ level. If the user has any constraint on the quality of service that he/she receives, the utility of the SQ levels lower than the constraint would be set to zero.

Equation (4.17) illustrates how efficient the energy is used to provide a certain quality of service. Alternatively, the current battery status (i.e. its residual energy) could be included in the factor as well.

Given two different operation modes with the resource usages of R_1 and R_2 and efficiency factors f_1 and f_2, if $R_1 \leq_v R_2$ and $f_1 \geq f_2$ then the first operation mode outperforms the second mode (i.e. less resource usage but higher efficiency). Hence, the second mode can be excluded from the problem without effecting the final solution. However even without excluding it, the overhead of the proposed algorithm for solving the problem remains the same (see Section 4.3.2).

It is worth noting that device parameters might change at runtime too, however it is assumed that they do not change before one instance of problem is solved.

Problem Statement

The IoT system model is summarized in Figure 4.17 where several IoT devices share a gateway to connect to the Internet and consequently share the resources on the gateway. The operation mode of each device should be selected at runtime such that the constraints of shared resources on the gateway as well as the constraints of each device are met. The problem that is targeted in this work can be solved by deciding the SQ level q and the offloading level l for each IoT device d at runtime, such that the constraints of shared resources on the gateway are fulfilled (Eq. (4.18)) and the minimum efficiency among edge devices is maximized (Eq. (4.19)).

$$\text{Resources constraint:} \quad \sum R_d(q_d, l_d) \leq_v \mathscr{R} \tag{4.18}$$

$$\text{Optimization goal:} \quad \text{maximize} \min_{\forall d} \left\{ f_d^0 + f_d(q_d, l_d) \right\} \tag{4.19}$$

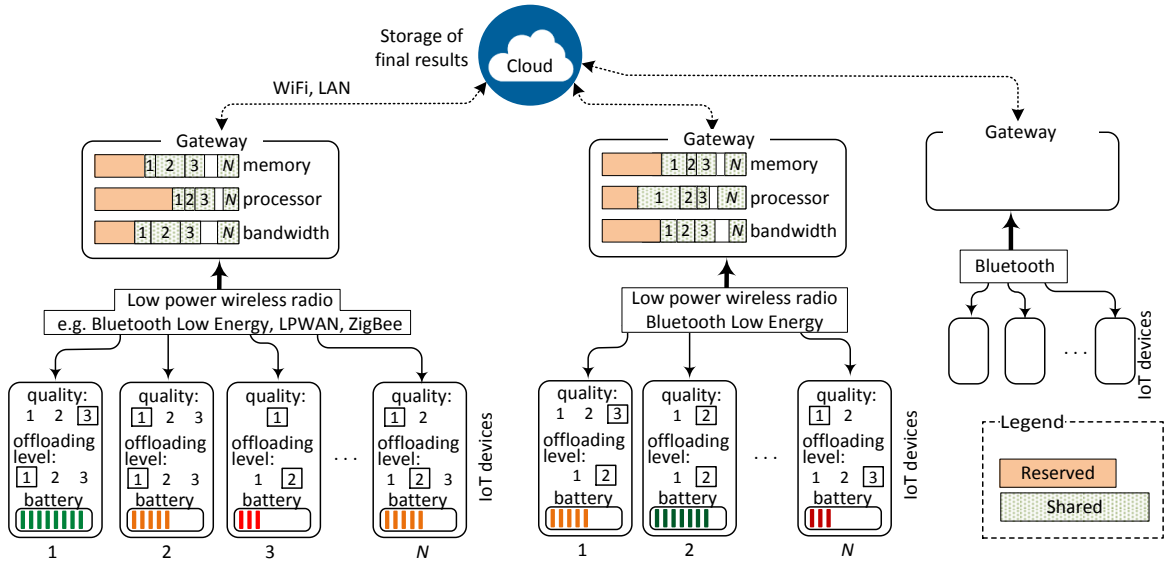


Figure 4.17: A share of gateway's resources is reserved for its management tasks (e.g. operating system, connectivity, etc.), the rest is shared between IoT edge devices with multiple operating choices (quality level, offloading level, etc.)

where the relation $\leq_v$ is defined in Def. 4.3.1, and f_d^0 is the efficiency factor from the previous rounds of mode selection. If this is the first round of problem, then $\forall d : f_d^0 = 0$.

4.3.2 Fast and Low-overhead Operation Mode Selection Scheme

This subsection presents a dynamic programming approach as the core of the solution similar to the state of the art [STX+16b]. Dynamic programming provides the opportunity to solve this problem optimally and quick. It avoids recomputing the same sub-problems repeatedly by memoizing their solution. Then two novel approaches will be proposed to enhance the solution which make it faster and reduce the memory requirements.

Complexity Analysis of the Problem

This subsection shows and proves that the problem of mode selection under resource constraints that is formulated by Eq. (4.18) and (4.19) is NP-Hard. The *Multiple Choice Multi-dimensional Knapsack Problem (MMKP)* is a well-known NP-Hard problem [CH09]. One can reduce the MMKP problem to the above-mentioned problem in polynomial time, as follows. The operation modes of each device is the class of items from which example one must be chosen. Each operation modes consumes multiple resources on the gateway which corresponds to the knapsack's capacity. The efficiency of each operation mode is the profit that user gains from the chosen item.

Since the mode selection problem is NP-Hard, there is no polynomial-time solution for it. However, the number of IoT devices connected to a single gateway is bounded due to the practical limitations. For instance in typical applications, the communication bandwidth of the gateway is only enough to support a couple of devices [GRW+15, ZMK+15]. Another limitation might

be because of wireless radio protocol. For instance, Bluetooth can supports up to 7 devices connected to a gateway.

In practice, the number of IoT devices in typical applications and scenarios does not grow. Practical limitations do not allow very large number of IoT devices to connect to one gateway. In addition, typical IoT applications do not have very dense deployment of IoT devices. Therefore, in such problems the absolute overhead of solution is more important than its complexity.

Addressing Critical Applications' Requirements

In many IoT systems some IoT edge devices may have a more critical application. This level of criticality may even change at runtime for some devices either due to the changes in the physical world and input data or upon user's preferences. The proposed mode selection scheme supports the devices with different priorities. The designer can distinguish important and critical devices by assigning a lower utility to their operating modes (see Eq. (4.17)). Since the proposed scheme aims at improving the lowest efficiency among devices, it consequently allocate more resources to the critical devices in order to improve their efficiency. Therefore, the utility of the device is a controlling parameter to address the criticality of its application at runtime.

Dynamic Programming Solution for Mode Selection

– Formulation & Example

Let $F(d,R)$ denote the output of Eq. (4.18) and (4.19) (highest minimum efficiency) that can be gained from the first d devices under constraint R for the shared resources of the gateway.

$$F(d,R) \;=\; \max_{\substack{1\le q\le Q_d\\ 1\le l\le L_d}} \left\{ \min\left\{ F\big(d-1,R-R_d(q,l)\big),\;\; f_d^0+f_d(q,l) \right\} \right\} \tag{4.20}$$

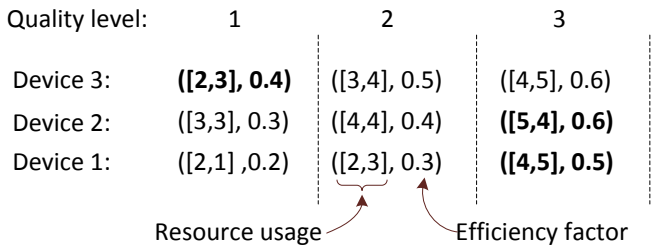
There are two boundary conditions:

$$F(d,R) = \begin{cases} -\infty & \text{if } \exists 1\le j\le g : R[j]<0 \\ \infty & \text{if } d=0 \end{cases} \tag{4.21}$$

The first condition prevents the violation of resource constraints (i.e. keeps the allocated resources less than or equal to the available resources on the gateway). The solution to Eq. (4.18) and (4.19) is $F(N,\mathscr{R})$.

Example 4.3.1: The following example is used to explain the details of the proposed solution in the rest of this section. Let us assume the gateway has two shared resources (i.e. $g=2$): $\mathscr{R}=[11,12]$. Three devices are connected to this gateway (i.e. $N=3$), each of which has 3 levels of quality (i.e. $Q_d=3,\ d=1,2,3$) and one level of offloading (i.e. $L_d=1,\ d=1,2,3$). Hence, the number of total operation modes for each device is $L_d\times Q_d=1\times 3=3$. The

resource usage of devices in different modes and associated efficiency factors are as follows:



The solution to this example is $F(3,[11,12])$. Equation (4.22) shows the first level of recursive calls to solve it:

$$\begin{aligned} F(3,[11,12]) = \max\{ & \min\big(F(2,[11-2,12-3]),0.4\big), \\ & \min\big(F(2,[11-3,12-4]),0.5\big), \\ & \min\big(F(2,[11-4,12-5]),0.6\big)\} \end{aligned} \tag{4.22}$$

The final solution to this example is $F(3,[11,12]) = 0.4$ which corresponds to these selected modes (quality level):

- Device 3 operates at mode 1, (i.e. $R = [2,3], f = 0.4$),
- Device 2 operates at mode 3, (i.e. $R = [5,4], f = 0.6$), and
- Device 1 operates at mode 3, (i.e. $R = [4,5], f = 0.5$).

– Memoization

While a bottom-up dynamic programming approach requires computing all sub-problems (even those that might never be called), a top-down approach allows to compute required sub-problems on demand. Therefore, a top-down approach for this problem has less execution time, consumes less energy and requires less memory to store the solutions. The recursive relation in Eq. (4.20) and (4.21) allows a top-down dynamic programming approach in which the solution to the sub-problems are stored after being called and re-use them in subsequent calls.

A linked-list structure is used to store the sub-problems which reduces the memory usage as it only allocates memory for the sub-problems that are called (instead of a statically allocate an array for the worst case). Although a linked-list is more memory efficient for storing the sub-problems, it has a disadvantage: for each sub-problem the list has to be searched to find the solution, which imposes additional time overhead. Therefore the advantages of having a short linked-list (by eliminating unnecessary entries) are twofold: i) reduce the required memory and ii) reduce the execution time. The next two subsections propose two novel approaches to reduce the length of the linked-list (number of stored sub-problems) and to decrease the execution time.

To reduce the memory usage, one can avoid storing some sub-problems that will never occur more than once. Since the operation modes supposedly have different resource usage (i.e. $R_d(\cdot,\cdot)$) the sub-problems $F(N-1,\cdot)$ occur only once, and therefore are not stored. In addition, the cost (memory usage and execution time) of memoizing some sub-problems may be more than the cost of computing them. The computation cost of the function calls for the $F(0,\cdot)$ is very low as they need to just check the boundary conditions (see Eq. (4.21)) for $L_1 \times Q_1$ modes. Therefore, one can avoid memoizing these sub-problems too.

Figure 4.18 shows the tree for recursive function calls (i.e. sub-problems) corresponding to Example 4.3.1. According to the above discussion, only the sub-problems $F(1,\cdot)$ are stored which are 9 in overall. In this example, two sub-problems benefit from memoization as they occur more than once and therefore after the first time, their solution is retrieved from memory instead of re-computation. The sub-problems that have not been re-computed (after memory hit) are marked with a red label in the figure.

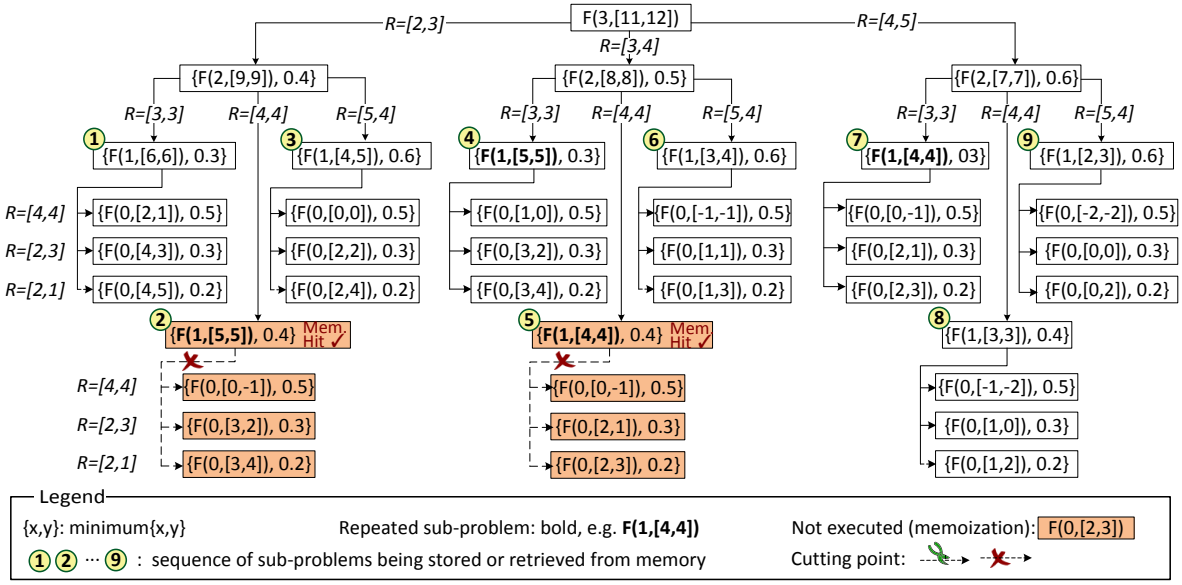


Figure 4.18: Recursion tree for the sub-problems in Example 4.3.1 with conventional memoization (used in state-of-the-art approach).

Cover a Range of Sub-problems

Based on the observations, when there are multiple types of shared resources (i.e. $g > 1$), as in Example 4.3.1, the number of distinct sub-problems increases. Therefore, the chance of recurring the same sub-problem decreases. It means that many sub-problems will be stored (i.e. huge memory overhead), but would have a low hit ratio. In Example 4.3.1, the hit-per-entry ratio is $\frac{2}{7}$ (7 sub-problems are stored with 2 hit access in overall). However, if the number of shared resources in Example 4.3.1 was $g = 1$ (ignoring the first resource), then the number of stored sub-problems would have decreased to 4, and the number of hits would have increased to 5. The more shared resource types there are (i.e. larger g), the larger the memory overhead becomes.

In the following, it is shown that even though some sub-problems are distinct (e.g. $F(1,[2,3])$ and $F(1,[4,4])$ in Example 4.3.1), they still have the same solution. This will help us to reduce the number of stored sub-problems, increase the hit ratio and decrease the execution time.

Definition 4.3.2: Vector $\bar{R}_{d,R}$ denotes the 'surplus' (i.e. leftover) resources after optimal allocation of resources by solving Eq. (4.20). For instance in Example 4.3.1, if $R = [4,4]$ then $\bar{R}_{1,[4,4]} = [2,1]$. According to Eq. (4.20), $F(1,[4,4]) = 0.3$ which corresponds to the second operating mode of device 1 whose resource usage is $[2,3]$. Therefore, the leftover of resources would be $[4,4]-[2,3]=[2,1]$.

Property 4.3.1: The optimal solution to $F(d,R)$ is identical to the solution of a range of sub-problems $F(d,R_2)$ where

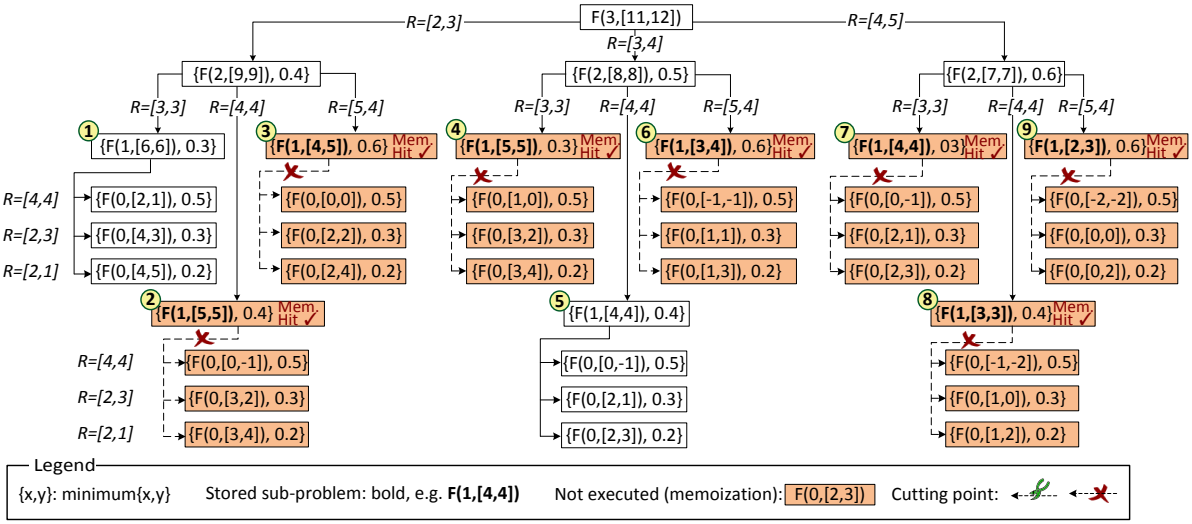
$$(R-\bar{R}_{d,R}) \leq_v R_2 \leq_v R$$

(refer to Def. 4.3.1 for the definition of $\leq_v$).

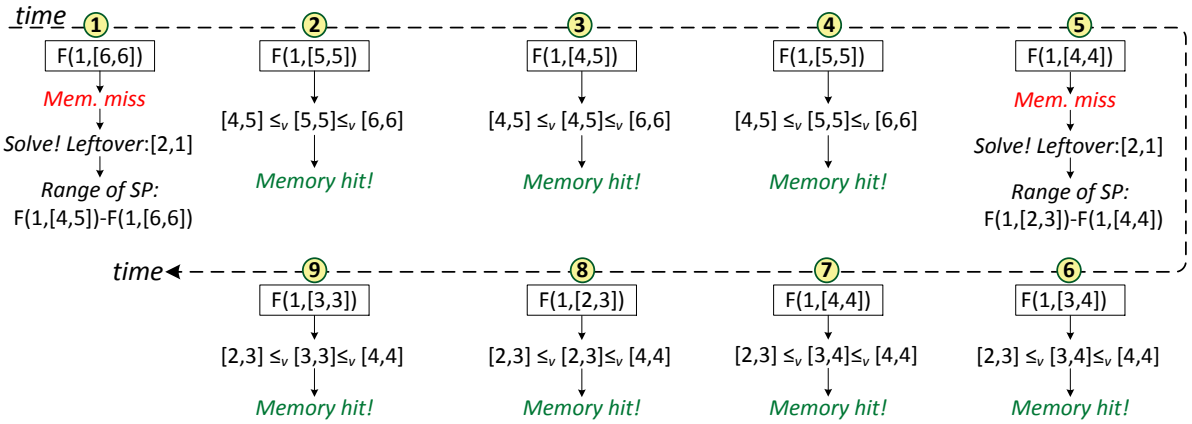
In the above-mentioned example, $F(1,[4,4])$ is equal to $F(1,[2,3])$ and $F(1,[3,3])$ (as well as $F(1,[2,4])$, $F(1,[3,4])$, and $F(1,[4,3])$, which are not called in this example). Therefore, using this property, it is possible to cover the solution of several sub-problems using the solution of only one. After solving each sub-problem, this scheme calculates the leftover (i.e. surplus) resource vector (i.e. $\bar{R}_{d,R}$). Instead of associating this solution to a sub-problem, it is associated with the range of sub-problems that it supports (see Algorithm 2 for more details). It results in a significant reduction in function calls and the required memory in the dynamic programming algorithm.

Figure 4.19a shows the tree for recursive calls (i.e. sub-problems) corresponding to Example 4.3.1 when using the novel memoization technique. Compared to the conventional memoization (see Figure 4.18), the number of stored sub-problems reduces from 7 to 2, but the number of memory hits increases from 2 to 7. Figure 4.19b depicts the timeline of the sub-problems being stored and retrieved over the time. Upon computation of sub-problem $F(1,[6,6])$, the leftover resources equals $[2,1]$ which makes the solution cover this range of sub-problems (SP): $F(1,[4,5])-F(1,[6,6])$. The three subsequent SPs fall into this range and encounter a memory hit. Sub-problem $F(1,[4,4])$ is not covered by any entry in the memory, therefore needs to be computed. The leftover resources, similarly, equals $[2,1]$ which leads to a solution that can cover the range of $F(1,[2,3])-F(1,[4,4])$. The four subsequent SPs falls into this range and consequently have the memory hit. Therefore their computation is avoided. This example shows that the novel memoization technique offer a great potential to reduce the memory required for dynamic programming solutions as well as their computation cost.

To benefit most from the proposed memoization technique, the operation modes of each device shall be examined in an increasing order of resource usage. Technically, in the data structure of each memory entry, instead of having one 'index', the scheme stores the 'start index' and 'end index'. According to the experiments on a 32-bit microcontroller unit (see Section 4.3.3), the size of each memory entry increases by (at most) 4 Bytes, but the number of entries to



(a) Recursion tree for the sub-problems in Example 4.3.1 with the novel memoization technique.



(b) The timeline for sub-problems of Example 4.3.1 being stored and retrieved.

Figure 4.19: An example for the proposed novel memoization technique.

store decrease significantly. Therefore, the memory requirement for storing the sub-problems decreases (evaluated in Section 4.3.3).

Pruning

The problems with max-min goal provide a key opportunity to further reduce the computation as well as the required memory by *pruning* the solution space. The intuition behind the pruning approach is as follows:

Property 4.3.2: Assume an arbitrary function $H(x)$ and two known scalars a_1 and a_2. To calculate the following term,

$$\max\{\ \min(a_1, H(x_1)),\ \min(a_2, H(x_2))\ \}$$

if $a_1 \geq a_2$ **and** $H(x_1) \geq a_2$ then the value of $H(x_2)$ does not matter. The proof follows immediately from the definition:

$$\left.\begin{array}{r} a_2 \leq a_1 \\ a_2 \leq H(x_1) \end{array}\right\} \Rightarrow a_2 \leq \min(a_1, H(x_1)) \left.\begin{array}{r} \\ \\ \min(a_2, H(x_2)) \leq a_2 \end{array}\right\} \Rightarrow \begin{array}{r} \min(a_2, H(x_2)) \leq \\ \min(a_1, H(x_1)) \end{array}$$

so under these assumptions, the value of $H(x_2)$ does not need to be computed.

Since the efficiency factors (i.e. $f_d(\cdot)$) are known and given as input to the problem, under the above-mentioned circumstance, some of the function calls of $F(\cdot)$ in Eq. (4.20) can be *pruned*. To benefit from pruning, the operation modes should be examined in an descending order with respect to their efficiency factor. For instance in Eq. (4.22), the value of $F(2,[7,7])$ is calculated first, and then the pruning condition is checked (see Property 4.3.2). If it does not hold, then the value of $F(2,[8,8]$ needs to be calculated, and so on. Otherwise, the scheme stops checking other sibling nodes in the recursion tree (i.e. prune them).

Example 4.3.2: In Example 4.3.1, the value of function $F(2,[9,9])$ depends on the values of $F(1,[4,5])$, $F(1,[5,5])$, and $F(1,[6,6])$. The value of $F(1,[4,5])$ equals 0.5. Since $0.5 > f_2(2,1) = 0.4$ **and** $0.4 > f_2(1,1) = 0.3$, according to Property 4.3.2, the value of those last two functions does not change the output and $F(2,[9,9])$ equals 0.5. Therefore the function calls to $F(1,[5,5])$ and $F(1,[6,6])$ are pruned.

Example 4.3.3: This example illustrates the effect of the proposed approaches (i.e. covering the range of sub-problems using surplus resources and pruning) with more details. In Example 4.3.1, the solution to the following sub-problems are memorized and retrieved:

sub-prob.	#hits	sub-prob.	#hits	sub-prob.	#hits
F(1,[2,3]):	0	F(1,[3,3]):	0	F(1,[4,4]):	1
F(1,[3,4]):	0	F(1,[5,5]):	1	F(1,[4,5]):	0
F(1,[6,6]):	0				

which results into 7 memory entries but with only 2 hit accesses. By exploiting Properties 4.3.1 and 4.3.2, the solution to these sub-problems are memorized and accessed:

sub-prob.	#hits	sub-prob.	#hits
F(1,[2,3]) – F(1,[2,3]):	0	F(1,[2,3]) – F(1,[3,3]):	0
F(1,[2,3]) – F(1,[4,4]):	2	F(1,[4,5]) – F(1,[5,5]):	1

which results into 4 memory entries and 3 hit accesses. It is noteworthy that some of the entries are a sub set of other entries. For instance $F(1,[2,3]) - F(1,[3,3])$ is a sub set of $F(1,[2,3]) -$

$F(1,[4,4])$. The reason is that the former is stored before the latter (its sub-problem has occurred first). Comparing Figure 4.20 with Figure 4.18, one can see that in the thirds level of tree (i.e. the sub-problems in form of $F(1,[\cdot,\cdot])$) the proposed technique reduces the number of sub-problem computations from 6 to 3.

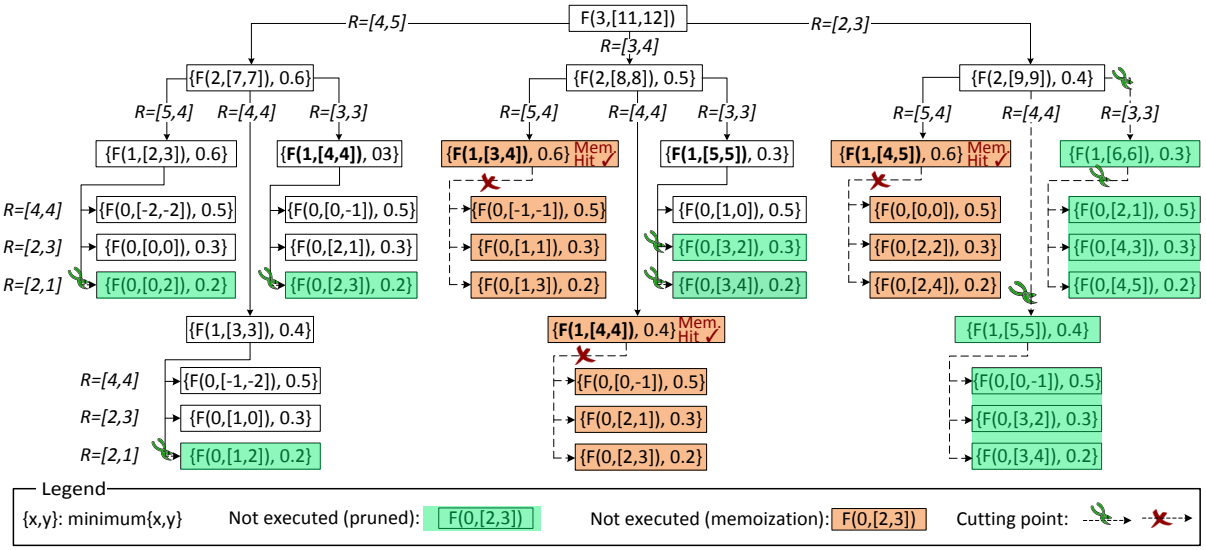


Figure 4.20: Recursion tree for sub-problems of Example 4.3.1 using the proposed approach. Some sub-problems are 'pruned' and more sub-problems benefit from memoization compared to Figure 4.18 (i.e. conventional memoization with no pruning).

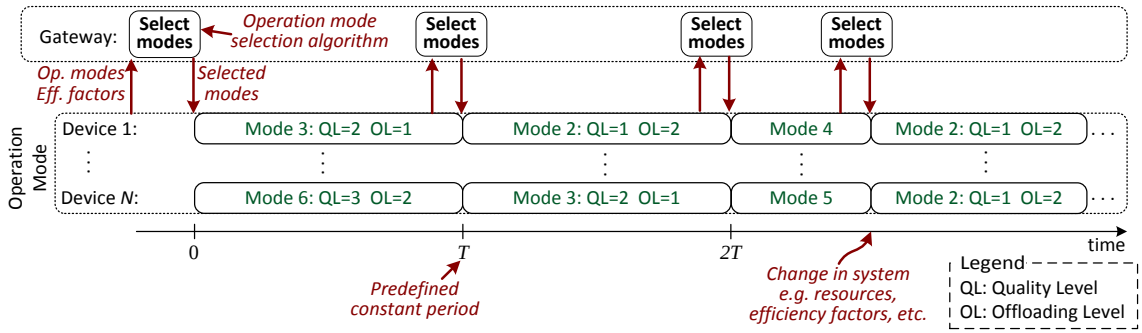


Figure 4.21: The system level overview of selecting and updating the operation mode of IoT devices

Figure 4.20 shows the tree of sub-problems (recursion tree based on Eq. (4.20)) associated with Example 4.3.1 using the proposed techniques. The sub-problems that are labeled with green are pruned. Therefore neither of them nor their sub-nodes in the recursion tree require calculation. In addition, the sub-problems that are bold and labeled with red are not executed as their solution was stored and available in the memory. Comparing Figures 4.18 and 4.20 shows that the proposed techniques not only reduce the number of function calls (sub-problems with no label: white boxes) from 32 to 15, but they also reduce the number of memory entries from 7 to 4. This example shows that this approach reduces the execution time at a lower memory usage.

Algorithm 2 illustrates the pseudo-code of the proposed approach. Lines 3-8 state the boundary conditions of Eq. (4.21). In Lines 9-13, the memory is checked for the solution of this sub-problem. If the solution is not memoized, it is calculated first (Lines 14-26), and then stored

it into the memory for subsequent calls (Lines 27-31). The algorithm will be implemented as management task in the middle-ware of the gateway. Gateway needs to receive the updated operation modes and their properties (i.e. efficiency factors and resource usage) from each device. It selects the optimal operation mode for each device with respect to its available resources and then informs the devices to operate at the selected modes.

Algorithm 2: Pruning & Novel Memoization for Dynamic Programming

```
1  Inputs : Sorted set of operation modes from devices, and resource vector of gateway R
2  Function F(d,R)
3      if !(0 ≤_v R) then          // resource constraint violated
4          solution -> eff = −∞;
5          return solution;
       else if d == 0 then         // last level in call tree: i.e. last device
6          solution -> leftover = R;
7          solution -> eff = ∞;
8          return solution;
       end
9      index ← calculate index of sub-problem F(d,R) in memory;
10     solution = get_memoized(index)    // retrieve solution from memory;
11     if solution != null then                          // memory hit
12         solution -> start_ind = index;
13         return solution;
       else // memory miss, so calculate the sub-problem:
14         for i ← |M_d| downto 1 do
15             get q and l corresponding to mode i;
16             f_{d,q,l} ← efficiency factor of this mode;
17             R_ql ← R_d(q,l);    // resource usage of d at q and l
18             solution ← F(d − 1, R − R_ql);
19             m ← min(solution -> eff, f_{d,q,l});
20             if m > max_min then
21                 max_min = m;
22                 trace = solution -> trace;
23                 leftover = solution -> leftover;
24                 best_id = i;
25                 if m ≥ efficiency of mode (i + 1) then        // prune!
26                     break;
                   end
               end
           end
       end
27     solution -> eff = max_min;
28     solution -> trace = concat(trace, best_id);
29     solution -> end_index = index;
30     solution -> start_index = get_Index(d, R − leftover); // supporting the range for sub-problems
31     add_to_memory(solution);
32     return solution
```

Systems-level Scheme for Operation Mode Selection

Figure 4.21 illustrates the timeline for operation mode selection in an IoT network where devices have 6 operation modes (3 quality levels $\times$ 2 offloading levels). After selecting the operation mode for IoT devices, the whole system keeps operating with this setup. However, the operation modes of devices need to be updated under one of these circumstances:

1. once there is a change in the problem's parameter (e.g. efficiency factor of IoT devices, number of IoT devices connected to the gateway, available resources on the gateway, etc.).
2. after a predefined constant period of time. This parameter can be determined by the system designer by considering the frequency of changes in the system and application's property. Therefore, it is assumed to be given as input.

4.3.3 Evaluation and Results

As explained in Chapter 2, Ref. [VQA14] targets the multi-dimensional resource allocation. It uses dynamic programming in core of its solution but it uses conventional memoization without any pruning. This is the nearest state-of-the-art competitor to the proposed work in this section, and therefore is compared against it. In [STX+16b], too, the core of the solution is dynamic programming with conventional memoization.

In the following, 'conventional memoization, no pruning' (CM-NP) is used to refer to the state-of-the-art techniques that use dynamic programming such as [VQA14] and [STX+16b]. The algorithms are implemented on the same platform, to have a fair comparison. Since both approaches (the proposed and [VQA14]) reach the optimal solution for operation modes, their overhead are compared in terms of execution time and required memory.

Experimental Setup

The experimental analysis includes measurements on the actual IoT devices and trace driven network simulation to study the behavior of the whole system under the proposed mechanism.

To efficiently model the characteristics of the case study for different configurations, the real execution of IoT applications are profiled (see Chapter 5) and interpreted as proposed and utilized in [STX+16b]. The profiling is performed on an Intel Quark SoC which has already proposed and used for wearable IoT devices [AB15, AMJ15].

– Target IoT devices The Intel Quark SoC platform is considered as the gateway which has been proposed and used widely as the IoT gateway. Its clock frequency is up to 400 MHz, and features 512 KB SRAM as the internal memory [int]. Two types of shared resources are assumed on the gateway (i.e. $g = 2$ in Section 4.3.1): i) processing power or CPU utilization and ii) communication bandwidth. In idle mode, when there is no workload for the gateway,

the CPU utilization on the hardware platform is about 40% which is due to the operating system's tasks. 50% of the CPU is dedicated to be shareable between IoT devices to perform the offloaded computation. The rest is dedicated for operating system, management tasks (e.g. 10% is dedicated to the management of IoT devices for operation mode selection).

Different number of IoT devices are considered to be connected to the gateway ranging from 5 to 10. In addition, different number of operation modes are assumed based on quality levels and offloading levels including 3 modes (i.e. 3 quality $\times$ 1 offloading levels or vice versa), 4 modes (i.e. 2 quality $\times$ 2 offloading levels), 6 modes (i.e. 3 quality $\times$ 2 offloading levels or vice versa), 8 modes (i.e. 4 quality $\times$ 2 offloading levels or vice versa), and 9 modes (i.e. 3 quality $\times$ 3 offloading levels).

Experiments & Results

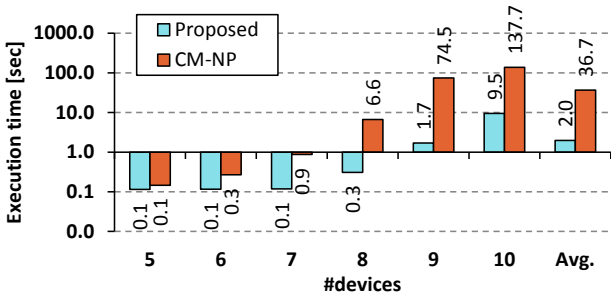
Both CM-NP and proposed technique in this section achieve the optimal solution for Eq. (4.18) and (4.19). The overhead of both algorithms are measured in terms of i) execution time overhead (number of CPU cycles) and ii) the memory overhead for storing the sub-problems. The algorithms are implemented and evaluated on the *Intel Quark* SoC gateway.

– Execution Time Overhead

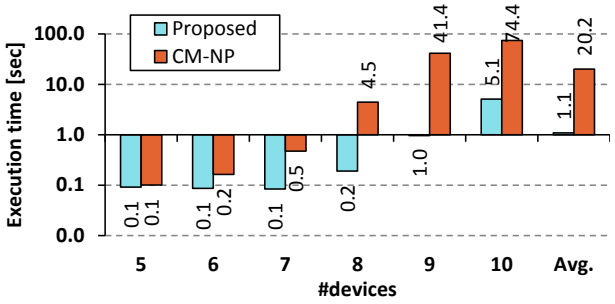
In the experiment, the CPU usage of processes is limited by means of *Linux Control Groups*, a standard tool provided by *Yocto*. The following four scenarios are considered for the CPU limit of the management algorithms: 10%, 20%, 50%, and 100% (i.e. no CPU limit). The execution time overhead of CM-NP and the proposed technique are measured and reported for these three parameters: 1) different number of devices, 2) different number of operation modes, and 3) the percentage of *Quark*'s CPU that is dedicated to algorithms.

In Figure 4.22, the number of connected IoT devices to the gateway varies from 5 to 10, while the number of operation modes is 4. As the number of devices increases the execution time overhead for CM-NP increases exponentially compared to the proposed. When the CPU dedication is 10%, the average of execution time overhead is reduced from 36.7 s to 2 s. For the CPU dedication of 100%, the average overhead of CM-NP is 6.8 s while the proposed scheme's is 0.4 s. With increasing the dedicated CPU from 10% to 100%, the runtime overhead of both algorithms decreases. However the ratio remains almost the same. For instance, when the number of connected devices is 10, the CM-NP is around 14$\times$ slower than the proposed algorithm for all the four scenarios (Figures 4.22a to 4.22d).

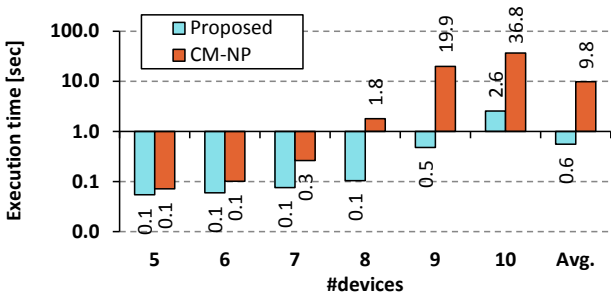
Figure 4.23 shows the the execution time overhead when the number of operation modes increases. The number of devices connected to the gateway is maintained constant as 7. As the number of operation modes increases, the runtime overhead of the proposed scheme increases moderately. while the overhead of CM-NP increases exponentially. The proposed algorithm decreases the average runtime overhead from more than 15 seconds to 0.1 seconds when the CPU has no limit to run the algorithms (Figure 4.23d).



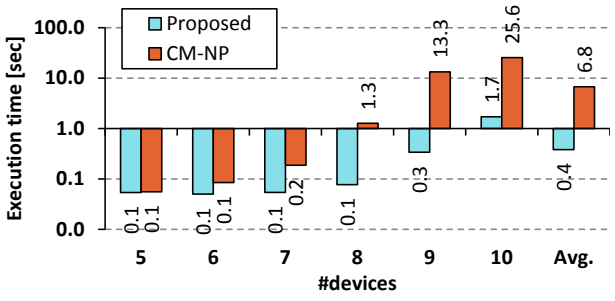
(a) dedicated CPU: 10%



(b) dedicated CPU: 20%

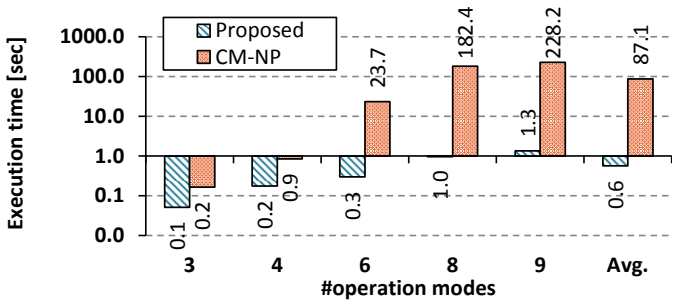


(c) dedicated CPU: 50%

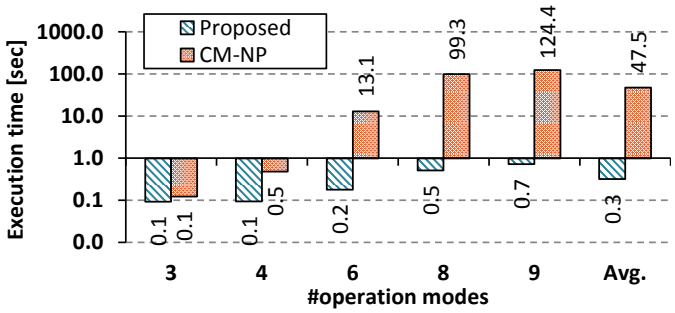


(d) dedicated CPU: 100%

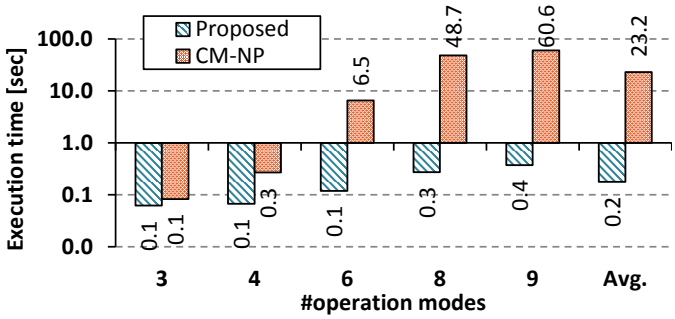
Figure 4.22: The execution time overhead of CM-NP compared to the proposed technique for different number of devices [number of operation modes is 4]



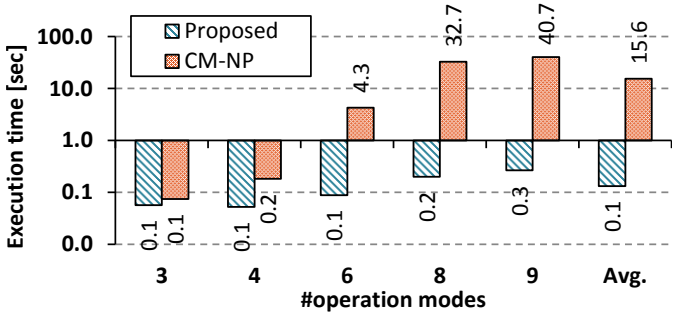
(a) dedicated CPU: 10%



(b) dedicated CPU: 20%



(c) dedicated CPU: 50%



(d) dedicated CPU: 100%

Figure 4.23: The execution time overhead of CM-NP compared to the proposed technique for different number of operation modes [number of devices is 7]

– Memory Overhead This subsection reports the overhead of the proposed algorithm and CM-NP in terms of memory usage for storing the sub-problems. The absolute required memory is presented when the number of connected devices and operation modes changes.

Figure 4.24a shows the size of required memory to store the sub-problems in the proposed method compared to the CM-NP method for different number of devices ranging from 5 to 10. The number of operation modes for each device in this experiment is 4 (e.g. two quality levels

and two offloading levels). Thanks to the novel memoization scheme, the required memory for the proposed algorithm is more than 50% less compared to state-of-the-art.

In Figure 4.24b, the required memory to store the sub-problems is presented for the proposed method compared to the CM-NP. The number of operation modes varies from 3 to 9 and the number of connected IoT devices in this experiment is 7. While the required memory for CM-NP is up to 61 KB, the proposed schemes uses less than 17.7 KB. The reduced memory requirements of the proposed scheme show its suitability and efficient run-time potential even for gateways with much lower internal memory. For instance, Texas Instrument microcontroller units which have an ARM Cortex-M4F core are used and deployed in some of TI gateways [Fol15b, Fol15a]. Their internal memory system has 64 KB to 256 KB SRAM. Note that the designer can limit the size of memory to store the sub-problems depending on the available memory on the gateway. In that case, when the memory quota is finished, the new distinct sub-problems will not memorized anymore.

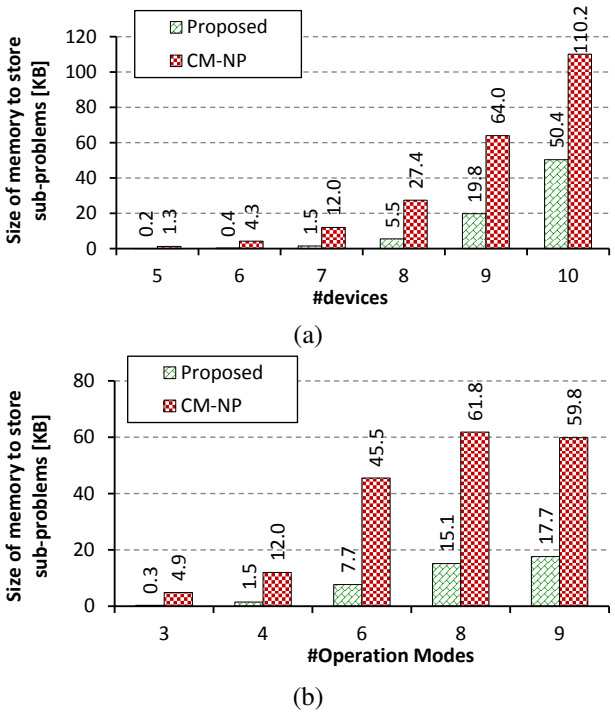
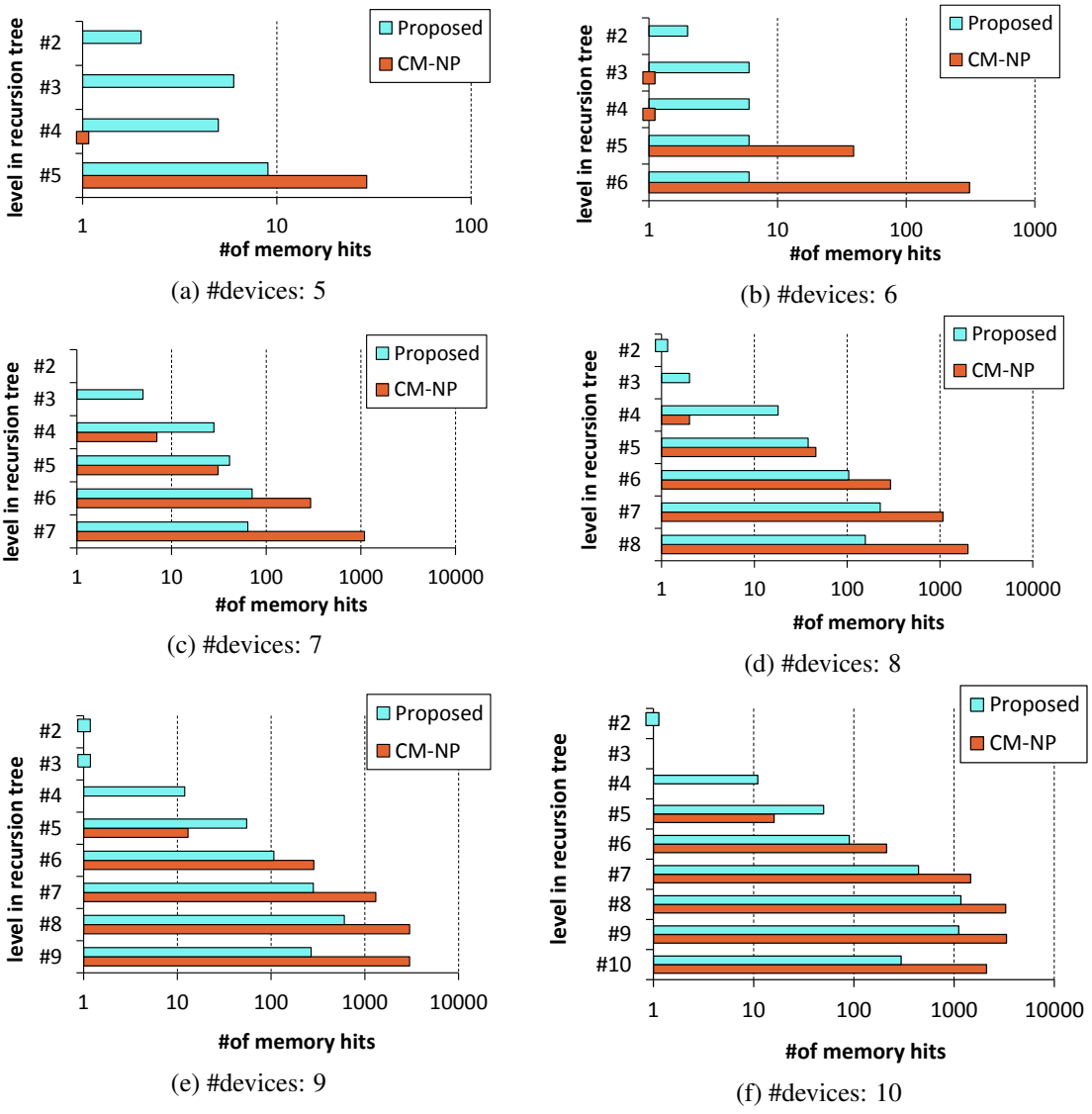


Figure 4.24: The size of memory to store the sub-problems for (a) different number of devices [number of operation modes is 4] and (b) different number of operation modes [number of devices is 7]

– Memory Hits

To further analyze the proposed memoization scheme, the number of memory hits for the sub-problems at different levels of the recursion tree (see Figure 4.19) are measured and reported. If the solution of a sub-problem is among the memorized and stored sub-problems, it is a *memory hit* and avoids re-computation of that sub-problem. When the memory hit happens in the early (higher) levels of recursion tree, it saves much more computation and therefore reduces the execution time more intensely.

Figure 4.25 shows the number of memory hits for sub-problems in each level of recursion tree. Note that the X-axis is in logarithmic scale, and the zero values are not (cannot be) shown. In addition, the values '1' cannot be shown in logarithmic figures, therefore to distinguish the '0' and '1' values in these figures, a small square on the Y-axis is used to indicate '1' memory hit. Another important note is that the first level of recursion tree is not stored as it does not occur more than once (see the recursion trees in Figures 4.18, 4.19a and 4.20). According to this experiment, the total number of memory hits in CM-NP is more than the proposed scheme. However in the proposed scheme, the memory hits in the early (first) levels of recursion tree are greater than in CM-NP. As explained, the influence of memory hits in the early levels is much greater.



(a) #devices: 5

(b) #devices: 6

(c) #devices: 7

(d) #devices: 8

(e) #devices: 9

(f) #devices: 10

Figure 4.25: Memory hits for stored sub-problems and the level of recursion tree where they occur

Comparison with Heuristic Solution

Since both CM-NP [VQA14] and the proposed scheme calculate an optimal solution, they have the same effect on the efficiency of the devices. Hence, a heuristic solution based on simulated annealing is exploited, which iteratively searches for an approximated solution. In each iteration, the heuristic algorithm produces new solutions based on the already calculated ones by examining neighboring solutions in a probabilistic manner. The total number of performed search iterations is the critical parameter which affects the efficiency of the outcome of the heuristic.

Figure 4.26 shows the achieved efficiency of the heuristic scheme normalized to the optimal solution (which is achieved by the proposed technique and CM-NP) with respect to its execution time. In this experiment the number of devices is 8, the number of operation modes is 4 and CPU of gateway is fully dedicated to the heuristic algorithm. As expected, the efficiency in the heuristic solution approaches the proposed scheme as the number of iterations increases. Nevertheless, the acquired results are highly affected by the inherent internal randomness of the simulated annealing algorithm, which cannot maintain the same quality of solutions across varying input problems.

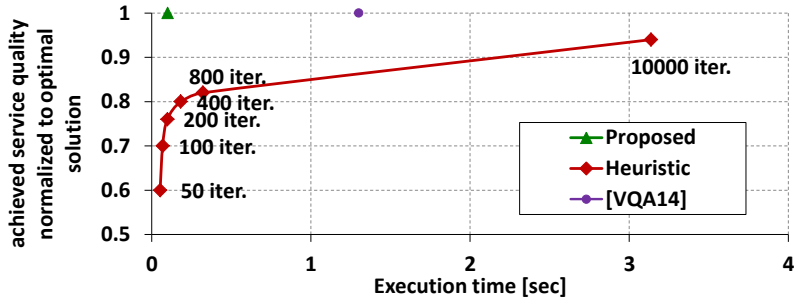


Figure 4.26: Achieved efficiency (normalized to the optimal solution) with respect to the execution time

4.3.4 Summary of Novel Memoization and Efficient Operation Mode Selection

In this section a fast and low-overhead mode selection algorithm has been proposed to improve the efficiency among IoT edge devices. This algorithm is based on a top-down dynamic programming. A novel memoization technique has been introduced in which each sub-problem determines the solution to a range of sub-problems instead of only one. It leaded to a significant reduction in the required memory to store the sub-problems. The proposed technique also leveraged the property of the problem and reduced the search space by pruning some sub-problems, which reduced both execution time and memory overhead. Experimental results showed up to 2 orders of magnitude in overhead reduction compared to state-of-the-art.

4.4 Distributed Trade-based Edge Device Management in Multi-gateway IoT

This section addresses the management of IoT devices and their operation mode selection in a multi-gateway system where some IoT devices are reachable by more than one gateway. After presenting the motivation and problem model, a trade-based mechanism is presented which starts with an initial setup for the IoT devices and then step-by-step improves the overall service quality by exchanging and migrating some IoT devices between gateways (when it is beneficial). The proposed mechanism, experimental results and findings are presented in [STB+17a] and [STX+16a].

4.4.1 Motivation

Existence of multiple gateways provides some IoT devices more than one option to connect and receive gateway service (referred to as *binding problem*). Binding problem arises when the range of wireless radio of gateways overlap and there are some IoT devices in the overlapping areas. Therefore, those IoT devices have multiple choices as their gateway. The importance of efficient binding is to avoid situations where some gateways are overloaded while other gateways are underutilized. For instance, if several IoT devices with high data transmission demand are connected to the same gateway, they must reduce their service quality to meet the constraint on the limited shared communication bandwidth. Consequently, the overall SQ of applications decreases. The binding problem is dynamic and requires online solution. Consider an example where some devices are mobile (i.e. users are moving in the building). As the location of an IoT devices changes, the gateways that can reach it will change too. Figure 4.27 shows a simple example of multi-gateway systems where two IoT devices are reachable by two gateways.

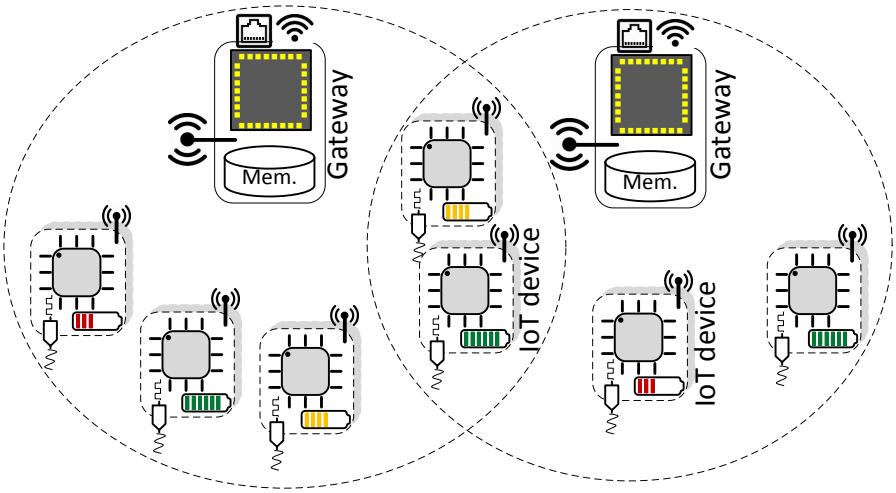


Figure 4.27: An example of multi-gateway IoT systems where some IoT devices are reachable by more than one gateway

A mechanism is needed to capture the dynamism in an IoT network and help the IoT devices to choose (i) their gateway, and (ii) their best operation mode to specify their SQ level and offloading scheme. These decisions are interdependent and should be made conjointly. The SQ management under battery lifetime constraints in a multi-gateway IoT system needs dynamic online solutions due to the following reasons:

1. The remaining energy of IoT devices varies over time due to consumption or recharge.
2. The available bandwidth or processing capability of the gateways may change over time according to their available power and their resource demand for management tasks.
3. The number of IoT devices connected to the gateway may change [Kim15] due to the mobility of the IoT devices (i.e. user) [KK15].

It necessitates an online mechanism to address efficient SQ management and offloading in such a dynamic IoT configuration.

The novel contributions of this section are as follows:

- It presents a resource management scheme for multi-gateway IoT systems with constraints on battery which jointly addresses efficient binding, bandwidth allocation and computation offloading.
- It analyzes in depth the new problem instance and suggests mechanisms to enable fine-grained SQ levels to be achieved.
- It proposes a distributed agent-based mechanism which improves the overall SQ by trading IoT devices between gateways.

IoT device Model

This section considers an IoT network consisting of N portable IoT devices, where each device I_d, $d \in \{1, \cdots, N\}$, is described by a tuple:

$$I_d = \Big(X_d, R_d, B_d, e_d, U_d(\cdot), C_d(\cdot)\Big) \tag{4.23}$$

where

- X_d denotes the set of possible *input data rates* of device d. They depend on the sensor sampling frequency and data resolution. An IoT device offers its service at M_d different *SQ levels*, with each level having a different input data rate and thus providing a different service quality.

$$X_d = \big\{x_{d_i} \mid i \in [1, M_d]\big\} \tag{4.24}$$

- R_d denotes the set of possible *transmission data rates* of device I_d. They depend on the input data rate x_{d_i} and the computation offloading strategy of the IoT device. offloading determines how much input data is not processed on the device (on-board processing), but transmitted to the gateway (offloaded) instead. An IoT device offers Q_d different *offloading levels*. The data transmission rate $r_{d_{ij}}$ depends on the SQ level i (input data rate) and the offloading level j. It corresponds to the share of input data rate that is offloaded plus the (intermediate) results from the on-board processed data.

$$R_d = \left\{ r_{d_{ij}} \mid i \in [1, M_d],\ j \in [1, Q_d] \right\} \tag{4.25}$$

  The particular transmission data rates depend on the device and how it is used, which has to be determined by the user and thus is considered as given in this problem formulation.

- B_d denotes the minimum required battery lifetime (i.e. until the next recharge or battery replacement).

- e_d is the remaining energy in the battery of device d.

- $U_d(x_{d_i})$ is the utility function that quantifies the utility or service quality (SQ) provided to the user when the device is capturing input data at rate x_{d_i}.

- $C_d(i, j)$ is the total power consumption for sensing and capturing input data at rate x_{d_i}, processing it under offloading level j and transmitting the data at rate $r_{d_{ij}}$. It includes the power consumption for sensing, computation and communication.

The battery lifetime of IoT devices depends on 1) the remaining energy and 2) the total power consumption rate:

$$b_{d_{ij}} = \frac{e_d}{C_d(i, j)} \tag{4.26}$$

where $b_{d_{ij}}$ denotes the expected battery lifetime when the device captures input data at rate x_{d_i}, processes it, and then transmits at rate $r_{d_{ij}}$.

Gateway Model

The gateway connects devices to the Internet. It receives data from IoT devices, processes it and transmits the final result to the Internet.

Assume a set of G gateways where each gateway g is specified by a tuple:

$$\text{gateway } g: \ \left(p_g(\cdot), R_g, P_g \right)$$

where:

- $p(r_{d_{ij}})$ shows the required processing capability of the gateway to perform the necessary operations on the received data at rate $r_{d_{ij}}$ and offloading level j,

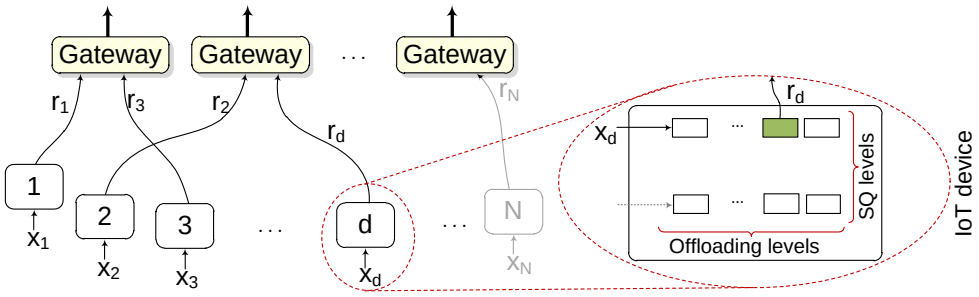


Figure 4.28: Problem model: IoT devices with different SQ and offloading levels resulting in different transmission data rates. Multiple gateways to receive and process the data.

- R_g is the total available bandwidth of the gateway to receive data from IoT devices.
- P_g shows the total processing capability of the gateway.

The effect of environment and surrounding devices (e.g. wireless interference) on the transmission can be modeled in gateways' parameter, R_g and IoT devices' parameter, $C_d(\cdot)$. For instance, if an IoT device observes an increase in re-transmission rate, it can increase the cost of transmission. However, modeling the effect of interference on the transmission parameters is beyond the scope of this work.

Network Model

While the gateways are assumed to be stationary and non-mobile, the IoT devices are *quasi mobile*. The IoT devices are mobile but their location does not change significantly and frequently, which results in low mobility. For instance, the IoT devices used for patient monitoring in a smart hospital or smart home have a mobility pattern of natural movement of patients [LCM10].

Depending on the location of IoT devices and gateways, each IoT device may reach some gateways (at least one). Each IoT device should connect to one and only one gateway in order to transmit the data/results to the Internet through it. For those IoT devices that reach multiple gateways, the binding needs to be decided. Let matrix $A[\cdot]_{N\times G}$ show which IoT devices reach which gateways:

$$A[dg] = \begin{cases} 1 & \text{if device } d \text{ reachs gateway } g \\ 0 & \text{otherwise} \end{cases}$$

Problem Statement

The system is summarized in Figure 4.28. The problem that is targeted in this article can be solved by 1) deciding the binding of IoT devices to the gateways and 2) choosing the SQ level i and the offloading level j for each IoT device d at runtime, such that the bandwidth,

computation, and lifetime constraints are fulfilled (Eq. (4.27) to (4.29)) and the overall benefit (Eq. (4.30)) is maximized.

$$\text{Bandwidth constraint:} \quad \sum r_{d_{ij}} \leq R_g \qquad \forall d \text{ connected to } g \tag{4.27}$$

$$\text{Computation constraint:} \quad \sum p(r_{d_{ij}}) \leq P_g \qquad \forall d \text{ connected to } g \tag{4.28}$$

$$\text{Lifetime constraint:} \quad \forall\, d:\ b_{d_{ij}} \geq B_d \tag{4.29}$$

$$\text{Optimization goal:} \quad \text{maximize} \sum_{\forall d} U_d(x_{d_i}) \tag{4.30}$$

4.4.2 Proposed Solution

Decomposing the Problem

The targeted problem has two sets of constraints: one for IoT devices and one for gateways. The selected configurations for devices d (i.e. x_{d_i} and $r_{d_{ij}}$) should meet the lifetime constraint (Eq. (4.29)). Given the selected configuration for IoT devices, the gateway's constraints to meet are the bandwidth and computation (Eq. (4.27) and (4.28)).

The device's constraint depends solely on device parameters. To reduce the search space, the optimization problem can be decomposed into 1) device's problem and 2) network (gateways) problem. In the device's problem, each IoT device excludes those configurations that violate its lifetime constraint to reduce the search space. Then, the network (gateways) problem is solved by considering the reduced search space.

Device Problem: Addressing Battery Lifetime Constraint

Considering its lifetime constraint, each device has a set a valid configurations. It finds the efficient feasible configurations (EFC) each of which corresponds to a SQ level with the minimum data transmission rate. Each EFC is a pair containing 1) the utility and 2) the transmission data rate r_{d_i} of this configuration. The gateway extends each EFC set by including the processing requirement of the associated transmitted data (i.e. $p(r_d)$). The resulting EFC'_d set is shown in Eq. (4.31). Note that the latency constraint is important in some IoT applications. Each individual IoT device takes care of its latency constraint by providing the EFC set which fulfills this constraint. If one configuration does not satisfy the real-time constraint, the device excludes it from its EFC set, and naturally, this configuration will not be selected for this device. The evaluation of the constraints (latency, battery lifetime) is online and dynamic: the device can monitor and adjust the parameters to include/exclude the configurations at runtime. The details on finding the EFC set can be found in [STX+16b].

$$
\begin{aligned}
&EFC'_d = \left\{ \left(U_{df},\, r_{df},\, p_{df}\right)_{1 \le f \le \left|EFC_d\right|} \right\} \\
&U_{df} = U_d(x_{d_f}) && \text{// Utility of device } I_d \text{ for the } f\text{-th EFC entry} \\
&r_{df} = r_{d_f} && \text{// Corresponding transmission rate} \\
&p_{df} = p(r_{d_f}) && \text{// Corresp. gateway processing requirement}
\end{aligned} \tag{4.31}
$$

Each IoT device periodically checks its remaining energy e_d and updates the EFC set. In case the EFC set changes, the device sends the new set to its gateway, where it is used to solve/update the *Network problem*. Note that the EFC set only contains feasible solutions, i.e. the number of entries in the EFC set of a particular device may change over time.

Network or Gateways Problem

Given the EFC sets of IoT devices, the Efficient Multi-Gateway Allocation and Binding (EM-GAB) problem can be formulated as:

$$
\begin{aligned}
& && \max \quad \sum_d \sum_f (U_{df} \times w_{df}) && (4.32)\\
&\text{subject to} && \forall d: \quad \sum_f w_{df} = 1 && (4.33)\\
& && \forall d: \quad \sum_g v_{dg} = 1 && (4.34)\\
& && \forall d,g: \quad v_{dg} \le A[dg] && (4.35)\\
& && \forall g: \quad \sum_d \sum_f r_{df} \times w_{df} \times v_{dg} \le R_g && (4.36)\\
& && \forall g: \quad \sum_d \sum_f p_{df} \times w_{df} \times v_{dg} \le P_g && (4.37)
\end{aligned}
$$

where

$$
w_{df} = \begin{cases} 1 & \text{if the } f\text{-th EFC element from the } d\text{-th device is chosen} \\ 0 & \text{otherwise} \end{cases} \tag{4.38}
$$

$$
v_{dg} = \begin{cases} 1 & \text{if the } d\text{-th device is connected to the } g\text{-th gateway} \\ 0 & \text{otherwise} \end{cases} \tag{4.39}
$$

Equation (4.33) ensures that one configuration for each device is selected. Equations (4.34) and (4.35) ensure that each IoT device is connected to one gateway which is in its range. Finally, Eq. (4.36) and (4.37) ensure that the binding of IoT devices to the gateways and their selected configurations meet the constraints of each gateway.

Analysis of the Problem

As stated in Section 4.4.1, some IoT devices have multiple choices to connect to gateways while some others reach only one gateway (i.e. no decision is needed for binding, but they still need to choose the SQ level and offloading policy). Let H_d denote the set of gateways reachable by device d ($|H_d| \geq 1$). The total number of bindings to investigate is $\Pi_{d=1}^{N} |H_d|$. Then for a possible binding setup, there are G optimization problems to find the optimal SQ level and offloading policy for the IoT devices. The optimal solution for one instance (i.e. a single gateway) is presented in [STX+16b].

Lemma 1. *The EMGAB problem (presented in Equations* (4.32) *to* (4.39)*) is strongly NP-complete.*

Proof. The numerical parameters of EMGAB (i.e. N, X_d and R_d) are bounded by a polynomial. The reason is the limitation of practical setup in IoT systems. Now it is shown that for the bounded parameters, the EMGAB problem remains NP-complete: EMGAB is a generalization of the Multiple Choice Multidimensional Multiple-Knapsack Problem (M^3PK). Each gateway corresponds to a knapsack which has two constraints: bandwidth corresponds to the 'volume' and processing power corresponds to the 'weight'. Each EFC' set of a device corresponds to a class of items among which, one item must be picked. This transformation between EMGAB and M^3PK is done in polynomial time. The M^3PK problem is a generalization of Multiple Knapsack Problem (MKP), Multiple Choice Knapsack Problem (MCKP), Multiple Choice Multidimensional Knapsack Problem (MMKP) which are proven to be NP-complete. Hence, an NP-complete problem is reduced to the EMGAB in polynomial time. Due to the polynomial bound on the inputs, the EMGAB problem belongs to the class of strongly NP-complete problems. □

The problem is computationally difficult to solve in a centralized fashion. Besides, the nature of IoT systems, with multiple devices and multiple gateways, is distributed. Therefore, distributed or decentralized strategies are promising solutions that take autonomous decisions for IoT devices and gateways based either on local information, or on an incomplete picture of the global network status. Market oriented approaches are usually used for distributed resource allocation problems and can be classified into three models: 1) price-based, 2) auction-based, and 3) trade-based [Smo00]. Mechanisms that are based on price and auction usually require a centralized entity with a full picture of network conditions (e.g. an auctioneer to run the auction or a decision-maker to calculate the price) [Smo00, Kim15, MPW07]. Due to the nature of the system and problem, a trade-based distributed solution seems more effective.

Distributed Solution and MGAB Protocol

This subsection presents a distributed solution to MGAB problem (i.e. multi-gateway allocation and binding) and the detailed protocol to implement it. The solution is based on trading, in

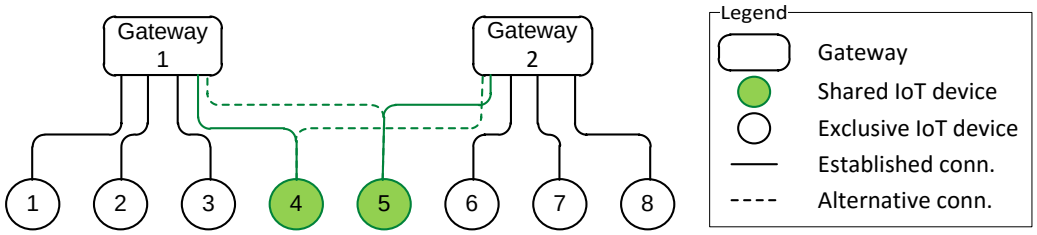


Figure 4.29: An example with two gateway sharing two IoT devices while each has three exclusive IoT devices

which gateways are modeled as intelligent agents that negotiate with each other to *take*, *give* or *exchange* the connected IoT devices.

Definition 4.4.1: *Common vs. Exclusive IoT devices:* From the perspective of gateways, an IoT device is either reachable by only one gateway which is called *exclusive*, or reachable by more than one gateway which is then *common* between them. Figure 4.29 shows an example with two gateways sharing two common IoT devices (i.e. devices ④ and ⑤) and three additional exclusive IoT devices per gateway. In the current *established connections*, IoT nodes ④ and ⑤ are connected to the gateways 1 and 2, respectively. However, there are three other *alternative connections* for them.

While the only action for exclusive nodes is to change the SQ and/or offloading level, there are two extra actions for common nodes:

1. Migration: One common node leaves its current gateway and joins the other one. For instance, if IoT device ④ disconnects from gateway 1 and connects to gateway 2, it is called migration.

2. Exchange: The two gateways exchange two common IoT devices with each other. For instance, IoT device ④ connects to gateway 2 in return of ⑤ being connected to gateway 1.

Given a binding/allocation setup, there are two situations that make it inefficient and an *exchange* or *migration* could address them.

Definition 4.4.2: *Fragmentation:* It a phenomenon in which gateways have unused resources that might not be enough for increasing the SQ level of the current connected IoT devices, but it might be enough for increasing the SQ level of a common IoT device which is currently connected to the other gateway.

Definition 4.4.3: *Heterogeneity of resource usage:* The IoT devices might be different in terms of resource usage, some might tend to use more bandwidth while some might tend to use more processing power. This can result in a situation where one gateway has much unused bandwidth and the other has much unused processing power. This phenomenon is called heterogeneity in resource usage.

Properties of Applications and Problem

This subsection derives and introduces some properties of the applications and the considered system that can be exploited to reduce the complexity of the problem.

Property 4.4.1: Considering device d and two elements of its EFC'_d set, (U_{di}, r_{di}, p_{di}) and (U_{dj}, r_{dj}, p_{dj}): If $U_{di} > U_{dj}$ then at least one of these conditions hold: 1) $r_{di} > r_{dj}$, 2) $p_{di} > p_{dj}$. The rationale behind it is that otherwise the i-th element dominates the j-th element and is always preferable to it.

Property 4.4.2: Considering device d and two elements of its EFC'_d set, (U_{di}, r_{di}, p_{di}) and (U_{dj}, r_{dj}, p_{dj}): The device d can operate in a way to provide any average utility (SQ) U'_d, $U_{di} \leq U'_d \leq U_{dj}$, from gateway's and application's perspective. This property is an extension to the proposed technique in [STX+16a].

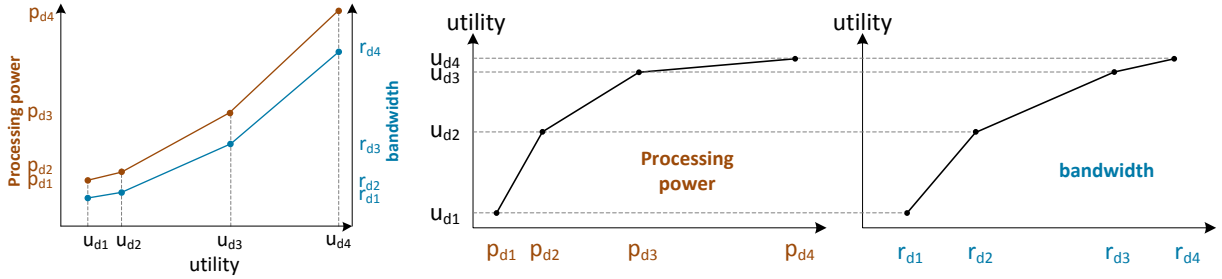
Proof. Consider a constant time interval of T. If the device operates at the j-th point for t_1 time and then changes the SQ level and operates at the i-th point for $(T - t_1)$ time, then the average utility, transmission rate, and processing power usages are:

$$\begin{aligned} U'_d &= \frac{t_1 \times U_{dj} + (T - t_1) \times U_{di}}{T} \\ r'_d &= \frac{t_1 \times r_{dj} + (T - t_1) \times r_{di}}{T} \\ p'_d &= \frac{t_1 \times p_{dj} + (T - t_1) \times p_{di}}{T} \end{aligned} \tag{4.40}$$

□

When a device is supposed to deliver an intermediate SQ level (e.g. U'_d in Eq. (4.40)), it uses the following scheme: It starts operating at the j-th configuration (which has higher utility and consumes more resources on the gateway compared to the i-th configuration). It keeps working at this configuration for t_1 time. However, it transmits its data to the gateway at the rate of r'_d ($r'_d \leq r_{dj}$). It buffers the rest of produced data (i.e. $r_{dj} - r'_d$) on the memory. Then after t_1, it switches to the i-th configuration that produces data at the rate of r_{di} ($r_{di} \leq r'_d$). It still transmits the data to the gateway at the rate of r'_d, which consists of previously buffered data and newly generated data. It keeps operating at the i-th configuration for $(T - t_1)$ time and then repeats this procedure as long as the requested utility remains U'_d. Therefore, while the device is only operating at the i-th and j-th configurations and switching between them, the gateway sees the device operating at an intermediate configuration mode with (U'_d, r'_d, p'_d).

In summary, this new operating point (i.e. configuration) is the application level function of the device. The device still operates only in discrete configurations, but it is transparent to the gateway and the average effect from the perspective of application and gateway is continuous.



(a) Processing power & bandwidth associated with utility values

(b) Piecewise-linear utility functions with respect to bandwidth and processing power

Figure 4.30: The extended utility function of device d derived from discrete EFC'_d set. The function is piecewise liner and weakly concave with respect to both variables (i.e. r and p).

– Forming Piecewise-linear Utility Function: Using Property 4.4.2, the discrete utilities of each device can be extended to a piecewise-linear function. However, since each discrete utility value (e.g. U_{di}) corresponds to two variables (e.g. r_{di} and p_{di}), there will be two uni-variable utility functions. These two variables are dependent. Figure 4.30 shows an example of these two utility functions for device d.

Property 4.4.3: *Concavity of Utilities* is a general property of typical applications [MPW07, BSLR10]. It is a natural restriction based on the "law of diminishing returns" from economic concepts [CG06, GN14].

According to Properties 4.4.2 and 4.4.3, the piecewise-linear utility functions are *weakly concave*. In the example of Figure 4.30b, the concavity of utility functions is illustrated.

Claim 1. *On each gateway that exploits Property 4.4.2, when the highest overall utility is reached, either at least one of the resources are fully utilized, or the IoT devices are operating at their highest SQ level.*

Proof. Suppose for the purpose of contradiction that gateway g reaches the highest overall utility but it still has $\hat{R} > 0$ and $\hat{P} > 0$ resources left, **and** there is one connected IoT device, d, which is not working at its highest SQ level (i.e. U_{d_M}). Let assume that this device is operating at i-th SQ level, $(i < M_d)$, which is the i-th element in its EFC' set (i.e. $(U_{d_i}, r_{d_i}, p_{d_i})$). Its next SQ level is associate with $(i+1)$-th element of its EFC' set described by $(U_{d_{i+1}}, r_{d_{i+1}}, p_{d_{i+1}})$. If $r_{d_{i+1}} \leq \hat{R}$ and $p_{d_{i+1}} \leq \hat{P}$ then the SQ can be increased by one level to $(i+1)$-th level, which is a contradiction. Therefore, at least one of these conditions hold: 1) $r_{d_{i+1}} > \hat{R}$, 2) $p_{d_{i+1}} > \hat{P}$. Let's assume that the first condition holds, then

$$\exists\, 0 < t_1 < 1 : \quad t_1 = \frac{r_{d_{i+1}} - \hat{R}}{r_{d_{i+1}} - r_{d_i}}. \tag{4.41}$$

If for each unit of time, the device operates t_1 portion at the i-th SQ level and $(1-t_1)$ portion at the $(i+1)$-th SQ level, it can provide $U' = t_1 \times U_{d_i} + (1-t_1) \times U_{d_{i+1}}$ utility which clearly is higher than U_{d_i}. This, too, contradicts the assumption. The same argument can be presented for the second condition (i.e. $p_{d_{i+1}} > \hat{P}$). If both conditions hold, the minimum t_1 value is selected.

To summarize, by using Property 4.4.2 it is possible to increase the overall utility which is a contradiction. □

Proposition 1. *According to Claim 1, the fragmentation problem can be addressed using Property 4.4.2.*

The fragmentation problem stems from discrete and coarse-grained configurations (i.e. EFC elements), which can be addressed by continuous utility functions derived using Property 4.4.2.

Details of Agent-based Approach

After addressing the fragmentation problem, it is needed to deal with the heterogeneity problem. In the following, the details of the proposed distributed solution is presented which is an agent-based negotiation mechanism between gateways, for migration and exchange of IoT devices. It starts with an initial setup and then converges step-by-step to the efficient solution by increasing the overall utility of IoT devices. Each gateway is modeled as an autonomous intelligent agent, where the interests (or goals) of each agent is consistent with the goals of the whole MGAB problem. The terms agent and gateway are used interchangeably from here on.

– Initial Phase

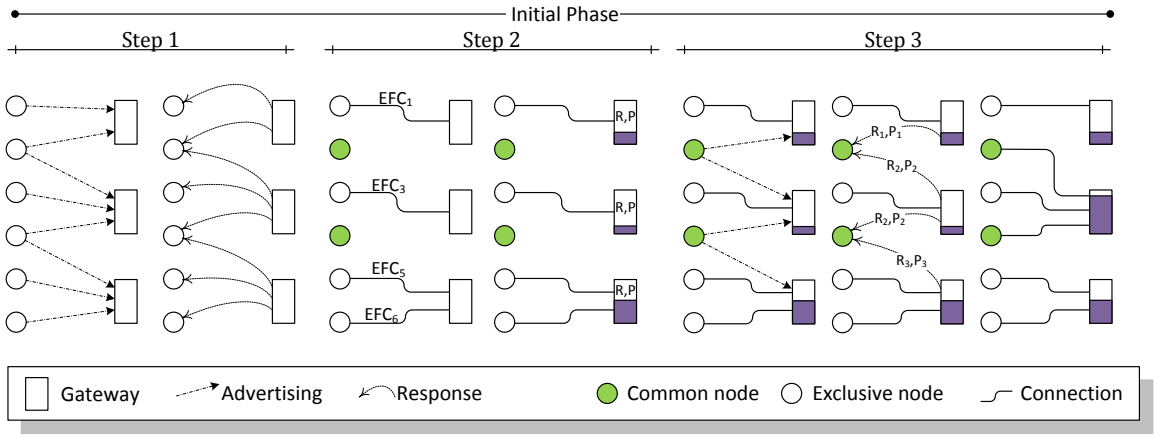


Figure 4.31: Different steps during the initial phase, followed by trade phase

During the initial phase, gateways try to establish connections with IoT devices to reach the first setup. Figure 4.31 shows an example for the initial phase with 6 IoT nodes and 3 gateways. It consists of three steps:

1. Advertisement and Discovery: First, the IoT devices broadcast advertising packets to find the gateways. Each gateway sends requests to establish the connection, one for each IoT device in its range (in response to the advertisement packets). All exclusive IoT devices receive only one request (as they reach only one gateway); others receive more than one request.

2. Exclusive Connections: Then, each exclusive IoT device accepts the request of its gateway and connects to it. It sends its EFC set to the gateway. After establishing the connection and receiving the EFC set of devices, the gateway checks the lowest SQ level of connected devices, and calculates the remaining resources to make sure that it still has enough resources to provide service to new devices. If not, it should stop sending requests to other IoT devices and accepting new devices.

3. Other Connections: Finally, common IoT devices send new advertisement packets. In return, the gateways in range send updated requests along with the information on their remaining resources. Each common IoT device accepts the first request for connection which has enough resources to support its lowest SQ level.

The advertisement and discovery mechanism is based on the BLE protocol where slave nodes (IoT devices) initiate the advertising and master node (gateway) sends the request in response to the advertisement [TCD+14]. However, other protocols or wireless technologies (e.g. ZigBee) can adopt a similar mechanism.

Once the initial phase ends, each gateway finds the optimal setup (SQ level and offloading policy) for its currently connected IoT devices based on the received information from its devices (as presented in [STX+16b, STX+16a]).

– Trade & Negotiation Phase After the initial phase, each IoT node is bound to one gateway. This setup is referred as 'initial binding'. Each gateway finds the optimal SQ level for its connected IoT devices. Since the number of connected devices to a gateway is practically small and limited, the gateway can find the optimal solution by executing a brute-force search or alternatively using faster algorithms similar to the solution proposed in [STX+16b]. The resulting configuration is only optimal for the current binding (i.e. single gateway level), but still not optimal for the whole network. This is called the initial allocation and is optimal for the initial binding.

The initial phase is followed by a trade and negotiation phase, where agents talk to each other to adapt the binding and move it towards the optimal binding. This is done by a *market-based* approach, where agents make offers for exchanging or migrating the connected IoT devices.

In the negotiation phase, two agents are involved in a trade: an agent initiates a trade by sending a request either to migrate one of its shared devices to the other agent or to exchange a shared device with another one. The other agent, after doing some evaluations, may 1) accept the offer, 2) reject the offer, or 3) make another offer in response to the request. This decision is in the direction of increasing the overall SQ of the system. In other words, each trade should be a *Kaldor-Hicks improvement* [Col79], where those IoT devices that are made better off gain more than what other IoT devices that are made worse off lose.

A. Migration:

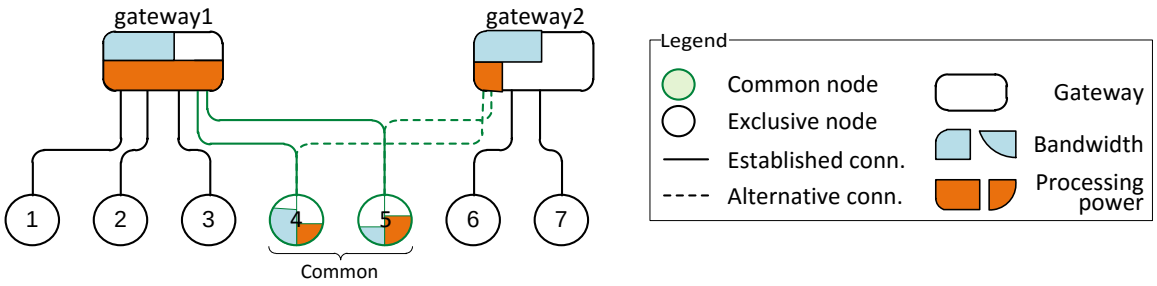


Figure 4.32: Heterogeneity in resource usage of IoT devices and gateways.

The migration is firstly and mainly meant to address the uneven binding (i.e. overloaded gateways). However, it can address the heterogeneity issue too. First, each gateway g, obtains 2 parameters: $\hat{R}_g$ and $\hat{P}_g$ which denote the amount of bandwidth and processing power left, respectively. Note that these values are non-zero for the gateways whose connected nodes are at their highest SQ level (see Claim 1). The migration is feasible only if one gateway (i.e. source) has no resources left (i.e. $\hat{R}_g \times \hat{P}_g = 0$), but the other gateway (i.e. destination) still has available resources.

Using the example shown in Figure 4.32, the negotiation of two agents is explained. In this example, nodes 4 and 5 are common between gateway1 and gateway2. In the initial binding, both nodes are connected to gateway1. The trade and negotiation details of a migration are as follows:

1. **Picking the candidate to migrate:** Gateway1 selects one of its common (i.e. non-exclusive) IoT devices for migration. It gives the highest priority to the node which consumes the scarcer resource more (i.e. makes the heterogeneity issue worse). For instance in Figure 4.32, node 5 consumes more processing power which is the scarcer resource on gateway1. Therefore it is prioritized over node 4.

2. **Checking the usefulness of migration:** Gateway1 calculates the 3-tuple of $(\overset{+}{u}, \overset{+}{r}, \overset{+}{p})$. This triple describes the best possible improvement among IoT devices *if* the candidate common device is migrated to another gateway. Therefore, $\overset{+}{u}$ is the gained utility, while $\overset{+}{r}$ and $\overset{+}{p}$ show the remaining bandwidth and processing power of gateway1 after this improvement, respectively. For instance, if node 5 migrates from gateway1 to gateway2, then other nodes might be able to increase their SQ level at the cost of getting more bandwidth and processing power from gateway1. If the candidate device migrates to another gateway, it releases its resources on the first gateway, freeing up r^* and p^* bandwidth and processing power, respectively.

3. **Make an offer:** Gateway1 sends an offer of migration to gateway2, which includes the information about 1) its benefit from improvement (i.e. $\overset{+}{u}$), 2) the EFC set of the candidate node, and 3) the current operating configuration of the candidate node (i.e. (u_5^*, r_5^*, p_5^*) in the above example).

Upon receiving this offer (i.e. migration request), the destination gateway (i.e. gateway2) evaluates the offer and replies to it. The offer is evaluated to check if this move can improve the

overall SQ of the system. The outcome of this evaluation determines whether to accept or reject the offer. The offer evaluation has two cases:

1. First, it compares its remaining unused resources (i.e. $\hat{R_2}$ and $\hat{P_2}$) with the requirements of the offered candidate. If it has enough unused resources to host the migrated device, i.e. $\hat{R_2} \geq r^*$ and $\hat{P_2} \geq p^*$, it accepts the offer.

2. Otherwise, it checks if it can reduce the SQ of other devices to be able to host the migrated note, while still improving the overall SQ. Gateway2 calculates the optimal allocation considering if the candidate node would have connected to it. Then it calculates $\bar{u}$ which shows the utility loss if node 4 had connected to gateway2. The offer is acceptable if the decrease in the overall utility of gateway2 is less than the gained utility of gateway1, i.e. $\overset{+}{u} > \bar{u}$.

If the offer is accepted, the initiator (i.e. gateway1) is informed. Then it notifies the migration candidate (e.g. device 5). The migration candidate receives a request for connection from the new gateway (i.e. gateway2), and it joins the new network. In this case, the current trade is terminated. The gateways update the SQ levels of their connected nodes according to the new situation.

A. 1. Outcome of Migration After the migration trade between two gateways, these three situations are possible:

1. For both gateways, the connected devices reach their highest SQ level. It means that these two gateways have reached their local optimal configuration, and therefore no more trade is needed between them.

2. For both gateways, the SQ level of connected devices can be improved but there is no resources left (i.e. $\hat{R}_1 \times \hat{P}_1 = 0$ and $\hat{R}_2 \times \hat{P}_2 = 0$). In this case, no more migration can help, and the gateways can only exchange to improve the overall utility.

3. On one gateway the devices are operating at their highest SQ level, while on the other gateway there are not enough resources left to increase the SQ level of devices (e.g. $\hat{R}_2 \times \hat{P}_2 = 0$). In this case, the gateways keep migrating the devices until either the situation changes to (1) or (2), or there is no more candidate to migrate.

It is worth to emphasize that in migration, the gateways will not encounter a ping-pong effect in trading (migrating nodes between two gateways back and forth). The intuitive reason is that the node is migrated only if the gained utility on source gateway is strictly greater than the utility loss on the destination gateway. Hence, the reverse move is certainly disadvantageous and harmful.

B. Exchange:

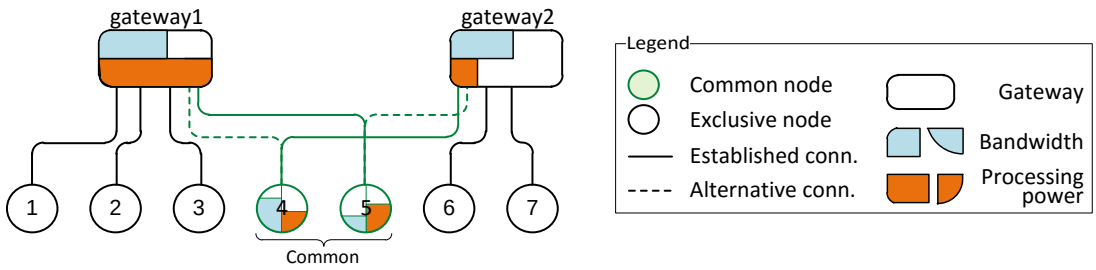


Figure 4.33: An example for exchange trade

Before proceeding to the details of the exchange move, some metrics are introduced that will be used to make decisions in the trade.

Definition 4.4.4: *Marginal value* is the increase (or decrease) in utility obtained for a fixed increase (or decrease) in the resources allocated to a device [BV04]. Loosely speaking, it is the derivative of the utility function (i.e. slope of lines in Figure 4.30).

The key policy of the proposed trade scheme is: *Having a unit of resource available on the gateway, allocate it to the device with the highest marginal value*. In other words, one shall allocate more resources to the IoT device that benefits from it the most. According to the concavity property (i.e. Property 4.4.3), the marginal value decreases (or remains the same, finitely) as the allocated resource to the node increases.

B. 1. Which Gateway & Which IoT Device? If a gateway has allocated the resources to its connected nodes such that they are all operating at their highest SQ level, this gateway has no incentive to initiate an exchange. Therefore, an exchange is initiated by a gateway that does not have enough resources left to increase the SQ of its nodes (i.e. $\hat{R}_g \times \hat{P}_g = 0$).

In the following, the details of the exchange trade are presented using the example shown in Figure 4.33.

- **Picking the candidate node:** If the initiator gateway (e.g. gateway1) has more than one candidate to consider for the exchange, it prioritizes the common node which uses the scarce resource the most. For instance, in the example shown in Figure 4.33, the scarce resource of gateway1 is the processing power. Hence, the common node which consumes the most processing power on gateway1 is the candidate to be exchanged. The intuition is that exchanging this node might resolve the heterogeneity issue, and free up more resources for other nodes to increase their SQ level (and consequently the overall utility).
- **Offer an exchange:** Once the candidate node is selected (node 5 in the example), the gateway sends an exchange offer to the other gateway (i.e. gateway2) which includes this information:
  - The EFC_5,
  - the current configuration of device 5, i.e. (u_5^*, r_5^*, p_5^*),
  - its remaining unused resources, $\hat{R}_1$ and $\hat{P}_1$, (at least one of them is zero).

The recipient of the offer, gateway2, needs to first select one candidate node and then evaluate the offer to check whether accepting this offer is in direction of increasing the overall utility of the system (i.e. is an Kaldor-Hicks improvement).

- **Picking the candidate node:** If it has more than one candidate to exchange, gateway2 prioritizes its common node which has an opposite resource usage profile compared to the offered node. It means that if node 5 is processing intensive, gateway2 looks for a node which is bandwidth intensive, and vice versa. In the considered example shown in Figure 4.33, the candidate node that gateway2 selects for the exchange is node 4.
- **Evaluate the offer:** Since the recipient of the offer (i.e. gateway2) has the overall information of the trade, it can make the decision to accept or reject the offer. Let $V_d(r,p)$ denote the maximum utility that device d can reach if the allocated bandwidth and processing power for it are r and p, respectively. This value can be found easily from the piecewise-linear utility function of the device (see Figure 4.30).

  Then, gateway2 needs to calculate and compare the possible change in utility of the exchanged devices:

  - First, it calculates $V_5(\hat{R}_2+r_4^*,\hat{P}_2+p_4^*)$, which describes the utility of node 5 in case it connects to gateway2, replacing node 4. It basically considers the amount of resources that it would have after exchange, $\hat{R}_2$ and $\hat{P}_2$ in addition to the resources that node 4 would release (i.e. r_4^* and p_4^*).
  - Then, it similarly calculates $V_4(\hat{R}_1+r_5^*,\hat{P}_1+p_5^*)$.

  Finally, to evaluate the received offer, gateway2 checks the expected change in the overall utility and decides about the offer:

$$\begin{cases} \text{accept} & V_5(\hat{R}_2+r_4^*,\hat{P}_2+p_4^*)+V_4(\hat{R}_1+r_5^*,\hat{P}_1+p_5^*) > u_5^*+u_4^* \\ \text{reject} & \text{otherwise} \end{cases} \tag{4.42}$$

Once the gateways agree upon a trade (exchange or migration), the IoT device will be disconnected from one gateway and connect to the other one. This process is usually very fast and takes only a few milliseconds (6 ms for Bluetooth Low Energy). During this time, the IoT device can buffer its data (if any) and transmit it after connecting to the new gateway. The state of process (i.e. execution context) on the gateway is very small for the IoT applications (negligible for the ECG processing case-study presented in Section 4.4.3). The reason is that most of IoT applications perform repeatedly an operation on the stream of input data. This operation is performed on a short 'window' of buffered data. Processing of one 'window' of data is independent of previous windows. Therefore the main communication overhead for exchange/migration is the negotiation of gateways for trading.

4.4.3 Evaluation & Experimental Results

In order to evaluate the effectiveness of the proposed mechanism, experiments and case-studies are conducted, which include real world measurements on an actual IoT device and trace driven network simulation to investigate the behavior of the system.

IoT Application Case Study

A healthcare monitoring application is considered in which the IoT devices capture the ECG signals from patients for abnormality detection and diagnosis (see Chapter 5). For the ECG analysis flow, actual patient ECG data records are used from *MIT-BIH Arrhythmia Database* [MM01]. It provides signals annotated by medical experts, which are used for feature extraction and classification. Original signals were sampled at 360 Hz. Down- and up-sampling has been used to generate ECG signals of differing sampling rates. These different sampling rates can be considered in two application scenarios:

1. In a medical scenario, the specialist requires the ECG trace to have a minimum sampling rate and SQ. This is the constraint given by the medical specialist (e.g. 360 Hz). Then only the SQ levels higher than this constraint would be considered (e.g. 360 Hz, 720 Hz, etc.). Higher sampling rate provides more accurate information which increases the probability of correct diagnosis.
2. In the normal usages (e.g. fitness and well-being), the user may accept a lower quality in collected data while still maintaining some crucial functionality, in return for longer battery lifetime.

Five SQ levels or input data rates are considered. The ECG analysis flow is pipelined in four stages: (I) Filtering, (II) Segmentation & heart beat detection, (III) Feature extraction, and (IV) Classification & Diagnosis. More details on the ECG analysis flow can be found in [STX+16b]. Table 4.2 summarizes the parameter values for combinations of SQ Levels and offloading levels stages for the ECG analysis application.

Table 4.2: Input data rates and transmission data rates for different SQ levels and offloading levels

			Transmission rate r_d [B/s] for offloading after a certain pipeline stage			
SQ level	Sampling freq. [Hz]	Input data rate x_d [B/s]	Stage 1	Stage 2	Stage 3	Stage 4
1	180	720	720	360	104	1
2	360	1440	1440	720	192	1
3	720	2880	2880	1440	372	1
4	1440	5760	5760	2880	564	1
5	2000	8000	8000	4000	1024	1

A key component of the system model is determining the utility functions of each device. The SQ value of each combination of SQ level and ECG processing stage was set proportional to

the ratio of the sampling frequency of ECG signal divided by the maximum available ECG sampling frequency (2 kHz), and multiplied by a diminishing factor of $(0.9)^{i-1}$, where i is the SQ level. It is possible to enable the creation of more complex profiles of IoT devices, i.e. by allowing the user to specify a factor of how important high SQ levels are for this device.

Experimental Setup

To conduct the experiments, a combination of experimentally derived data enhanced with nominal data from data-sheets of commercial devices is used for the values of the model parameters. Regarding the available energy of the IoT device (e_d), a battery consumption model of each IoT node is composed based on the instrumented CPU utilization. To complete the battery model of the IoT nodes, a rechargeable Lithium-Ion coin cell battery is chosen with nominal capacity of up to 420 mAh [DMHL12]. A realistic discharge model for the battery is used [ZLSX13] for various values of discharge currents to evaluate the available energy for Eq. (4.26).

An ARM Cortex-M3 device is considered as the gateway [MPV11]. The energy consumption values were acquired by hardware measurements and profiling the execution of the ECG analysis flow for all combinations of SQ levels and processing stages to measure the values of the parameters, e.g. $C_d(\cdot)$, $p(r_{d_{ij}})$, etc. The energy consumption of ECG acquisition was calculated based on [Han13].

Bluetooth Low Energy (BLE) is used for communication between IoT devices and gateway. Power consumption value of data transmission is 0.153 μW based on [Smi11] and transmission latency is 4 $\mu s/bit$ [SHNN12]. Since BLE exploits adaptive frequency hopping mechanisms, the probability of interference is very low and even if it happens, it would be for a very short period of time. Moreover, the gateways communicate with each other using either a different medium (wired) or a dedicated frequency band such as Sub-1 GHz (different from the band that IoT devices are transmitting their data). Therefore the interference among gateways is avoided and would not affect the IoT devices and their battery.

Simulation framework

The following experimental analysis has been conducted using a simulator, created as a part of this work, which implements all the models of the different devices of the system. The inherent distributed nature of the system under analysis, led us to create a simulator where each one of the previously described agents (Gateways and IoT nodes) corresponds to a different processing entity. In this way, message exchange and execution of the proposed resource allocation scheme can take place in parallel, thus creating a realistic interplay of the involved agents.

A very important goal was to be able to capture the topological characteristics of an examined IoT based system. In this respect, the simulation evolves in a virtual grid of devices, where in each position of the grid there can be either an IoT node or a Gateway. Using this feature, the framework simulates real-life conditions where communication of different nodes is affected

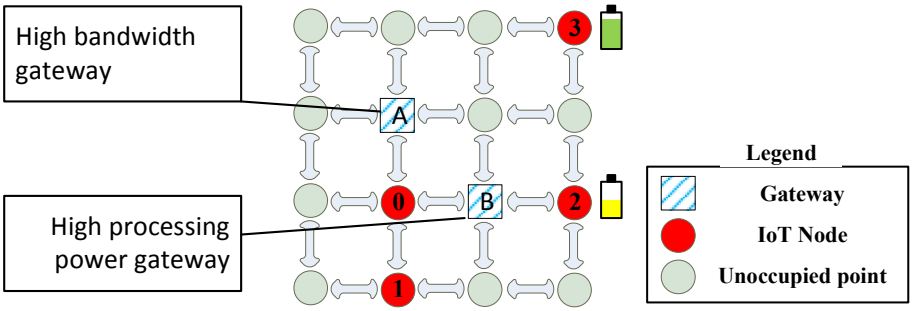


Figure 4.34: An example of an examined IoT based system

by their distance and interference with the rest of working devices in proximity. Every node is unique and can have different different initial values in its characteristics, such as battery capacity for an IoT node or available bandwidth and processing capabilities in a gateway.

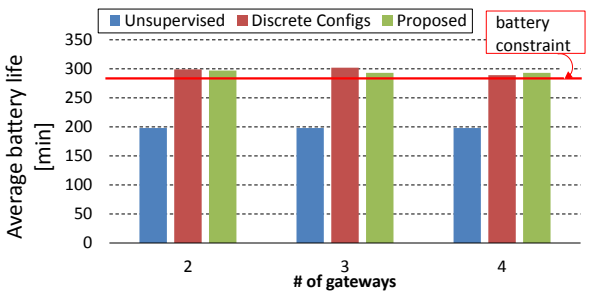
An example of an instance of the topologies than can be mapped to the simulation framework is illustrated in Figure 4.34. In this case, there is a 4x4 grid with 2 gateways and 4 IoT nodes. The rest of the grid points are unoccupied, but are available for use in other scenarios. Each one of the devices, has its own characteristics, e.g. IoT node 3 has increased battery capacity compared to IoT node 2 whose battery is low. The same applies for gateways, where A is rich in bandwidth while B in processing power.

From the point of view of the implementation, the simulator has been created as a C program running on Linux OS. Each different entity is mapped to a different process of the OS, in order to provide encapsulation and enable parallel execution. Inter-process i.e. inter-node communication is achieved by writing data packets in the shared memory of each entity. These data packets correspond to either signals or data which are transmitted from one node to the other. Semaphores have also been incorporated in the communication scheme, in order to help its orchestration and protect against the violation of the memory space of a process by another one.

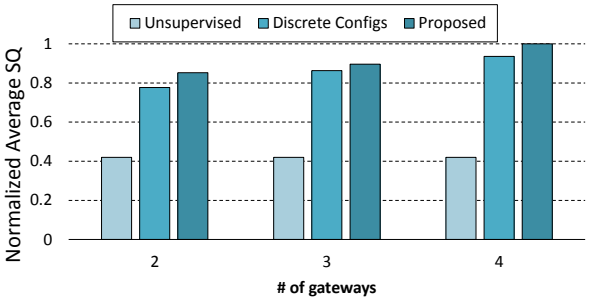
Results

An IoT system is considered to be located inside a big hospital room or a clinic ward. The systems consists of a number of patients whose vital signals and mainly ECG signals are monitored by wearable IoT edge devices. The room is simulated by a 8x8 grid where a different number of gateways and IoT edge devices are present per scenario.

Three distinct scenarios are identified where the number of patients inside the room is (i) low, (ii) medium and (iii) high. This affects the requirements of the connection to the available gateways, leading to increased demand as the number of patients rises. Furthermore, for each scenario a different number of available gateways is examined. The gateways are located diagonally in the room as shown in Figure 4.34.

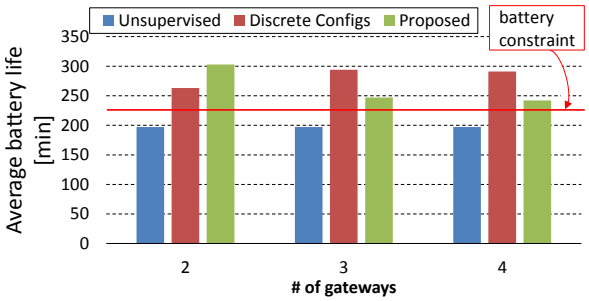


(a) Average battery lifetime

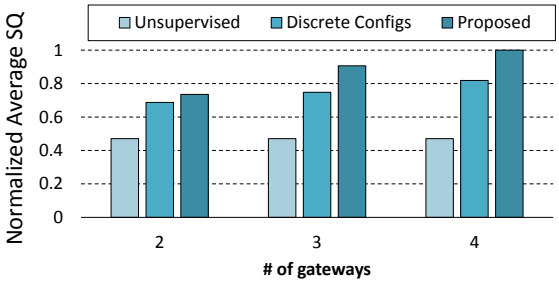


(b) Normalized average SQ level

Figure 4.35: The scenario of low number of IoT devices



(a) Average battery lifetime



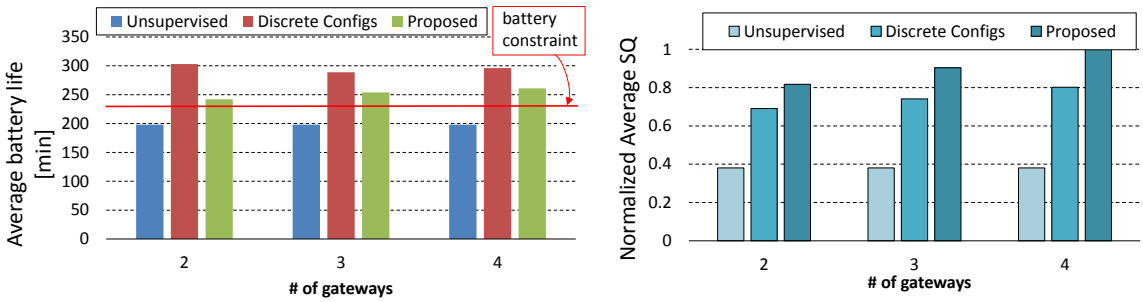
(b) Normalized average SQ level

Figure 4.36: The scenario of medium number of IoT nodes

The characteristics of each IoT node also differ since they are mapped to different patients. Each device has its own initial battery capacity and expected battery lifetime, accompanied by a coefficient indicating the importance of increased SQ level to the specific user.

The last critical design alternative examined in each scenario, is the algorithm employed by the gateways in order to accomplish the binding and allocation for the operation points of each IoT node. The available options are to execute the resource allocation algorithm either with discrete configuration points [STX+16b], or with the proposed scheme. In both cases, the ability for gateways to migrate or exchange nodes is activated. Another comparison is with the scheme where devices operate in the absence of any resource management scheme, where IoT node operates in a medium SQ level and only communicate their processing outcome to the gateway.

The results of the three examined scenarios are presented with two metrics: 1) Average battery lifetime of the IoT nodes and 2) normalized average achieved SQ level. The first examined scenario was the one with a low number of patients (i.e. IoT edge devices). Figure 4.35a illustrates the average achieved battery lifetime of the devices. This metric is chosen since there is no global expected lifetime value for all devices but each one operates under a unique constraint. To visualize the expected battery lifetime, the red line on the figure denotes the average expected battery lifetime, taking into account all devices involved in the scenario. In both techniques for SQ management, all devices meet their battery lifetime constraint. On the contrary, the unsupervised system behaves very poorly and misses the majority of the constraints.



(a) Average battery lifetime

(b) Normalized average SQ level

Figure 4.37: The scenario of high number of IoT node

In Figure 4.35b, the average achieved SQ level is presented, normalized in respect to the highest achieved value in this scenario. The proposed resource allocation scheme operating on a continuous utility function, achieves better SQ levels compared to the scheme with discrete utility function in all cases. Since the number of devices is low, the gain remains low as in both schemes there are enough available resources. The gain in SQ level of the proposed scheme comes at the price of decreased average battery lifetime of IoT nodes compared to the discrete version. However, this is acceptable since the battery lifetime constraints are still met.

This presented system behavior remains the same throughout the examined scenarios with medium and high number of devices as illustrated in Figure 4.36 and Figure 4.37. As expected in these cases, as the number of IoT devices increases, the available resources on the gateways become more and more scarce. Consequently, the ability of the proposed solution to fully utilize available bandwidth and processing resources of the gateway, results in improvement in achieved SQ levels. The same overall SQ level cannot be achieved by the discrete counterpart of the resource allocation scheme, since it suffers from the fragmented utilization of its resources. These gains reach up to 22% and 24.6% in the cases of medium and high number of edge devices, respectively. In overall the gains compared to the unsupervised version of the system start from 56% and can reach up to more than 100%.

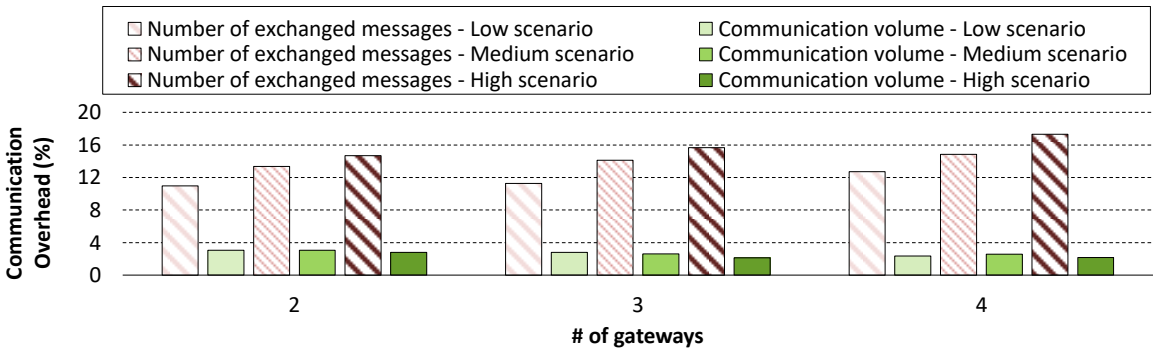


Figure 4.38: Communication overhead

Figure 4.38, presents the overhead of the resource management negotiations on the rest of the system communication, which corresponds to the data traffic generated by the applications running on the IoT nodes. Assuming that Tot_Mngmt_Msgs is the total number of messages ex-

changed for resource negotiations and Tot_App_Msgs is the total number of application related messages sent by the IoT nodes to gateways, then the overhead in message count is defined as

$$\frac{\mathsf{Tot\_Mngmt\_Msgs}}{\mathsf{Tot\_App\_Msgs}} \times 100\% \tag{4.43}$$

while the overhead in transmitted data volume equals to

$$\frac{\mathit{Sizeof}(\mathsf{Tot\_Mngmt\_Msgs})}{\mathit{Sizeof}(\mathsf{Tot\_App\_Msgs})} \times 100\%. \tag{4.44}$$

Experiments showed that the overhead in the number of exchanged messages ranges from 10% to 18% and is higher for increased number of IoT nodes (patients) and gateways, since more devices need to communicate in order to designate the operating points of the system. Conversely, the overhead on communication volume is almost constant and lower than 3% in all cases. To interpret this result, the differences in the structure of resource management and application related messages should be taken into account. Resource negotiation messages are small and contain only a few values as described in Section 4.4.2, while application messages transmitted to the gateway can contain many values according to the SQ and off-loading level of each device, e.g. a complete heartbeat window sent for processing. As a result, the resource negotiation related messages constitute a very small fraction of the total communication volume.

4.4.4 Summary of Trade-based Management in Multi-gateway IoT

This section studied the joint problem of binding and resource allocation for multi-gateway IoT systems, where the IoT devices can provide different service quality levels and can offload a share of their workload. The SQ management has to fulfill constraints for the battery lifetime of IoT devices, communication bandwidth to the gateway, and processing capability of the gateway (for offloading). This section presented an Integer Programming formulation for this problem and decompose it into separated device and gateway problems, and then showed that the gateway problem is NP-hard and practically impossible to be solved in a central fashion. It proposed a distributed agent-based mechanism to find the sub-optimal solutions. It demonstrated the effectiveness of the proposed approach using a case study of ECG processing in a personal healthcare monitoring application. The experiments showed that the proposed solution achieves up to 56 % accumulated SQ compared to an unsupervised system and up to 24.6% compared to a resource allocation scheme with discrete utility configurations while they both meet the constraints of edge devices.

4.5 Summary of Resource Management Techniques

This chapter has studied the problem of resource management and operation mode selection for the IoT devices at the edge of IoT network. It has presented a model for IoT applications in health monitoring domain, and then introduced multiple operation modes of IoT devices. Then,

it has proposed techniques to determine the operation mode of IoT devices at runtime while considering the constraints of IoT devices and the limited sharable resources of the gateway as well as the application requirements. Efficient and fast techniques have been presented to reduce the overhead for managing the shared resources through mode selection. It has also presented a distributed trade-based mechanism to manage the IoT devices in a multi-gateway system. The problem of resource management in a multi-gateway is more complicated since some IoT devices may have more than one option as their gateway to connect to. Therefore, the binding and allocation problems must be considered conjointly. The proposed mechanism starts from an initial setup and step-by-step improves the overall service quality of system by migrating and exchanging IoT devices between gateways.

5 Application-Specific Optimizations for Healthcare

The aim of this chapter is to study a class of IoT applications to derive a general model to describe their data processing pipeline and present application-specific optimizations for them. This chapter considers three different applications which involve signal processing and classification, and model their data processing steps. The applications belong to the remote health monitoring domain. These applications are:

1. EEG processing to predict epileptic seizures (discussed in Section 5.1),
2. ECG processing to detect heart abnormality (discussed in Section 5.2),
3. and physical activity monitoring (discussed in Section 5.3).

These applications are designed, implemented and analyzed to obtain a general model for the applications in the same domain. Section 5.4 presents a novel approximate compression technique for the IoT applications that process bio-signal data. Eventually, Section 5.5 concludes the chapter by summarizing the contributions of the chapter.

– Background: One of the main and trending application domains for IoT is the healthcare and well-being domain. Wearable healthcare monitoring devices are used to monitor elderly people or patients' health status while they are out of the hospital doing their daily activities. Wearable devices are also very popular and useful for sports, fitness and wellness to measure daily activity, sleep patterns, and other parameters related to well-being [SBH15]. They sense and process the vital signals of a person (e.g. ECG, EEG, EMG, skin temperature) anywhere and anytime for as long as it is needed, and transmit the (processed) data to the gateway. In the following, three applications are implemented and studied to show their similarities.

The presented model in Section 4.1 was based on the implemented applications that are described in this chapter. The model help the designer to characterize the software application, and then leverage the properties to optimize the application (both at hardware and software level) and achieve higher efficiency in terms of energy, power and resource usage. The resources in an IoT system include:

- Device's resources such as energy (battery or harvested), internal memory, processing power, etc.
- Network's resources such as communication bandwidth, processing power of the gateway, storage memory of the gateway, internal memory of the gateway, etc.

IoT devices capture input data using sensors from the physical world. The collected data – usually in forms of samples– needs to be processed to retrieve the *information*. The IoT applications can be categorized into two classes with respect to the relation of data samples and extracted information:

1. An individual sample does not provide meaningful information, but a sequence of samples are required. For instance in a heart rate monitoring application, one sample of ECG data cannot be used to determine the location of a beat, but a window of samples (e.g. 3 seconds long) is usually used to locate the beat or other crucial complexes [BAAR12]. In such applications the desired information, which is conveyed by input signal, is spread over time. The collected data from accelerometers, gyroscopes, sound and biomedical signals fall into this category. These applications are usually involved with signal processing.

2. In some applications, however, the individual samples can be interpreted solely and provide meaningful information. For instance, individual samples from temperature or humidity sensor can be sufficient in some applications like smart cooling systems. Force sensitive resistors (FSRs), gas sensors for air quality applications, pressure sensors, luminance sensors are among sensors used in such applications.

The applications that are implemented in the scope of this thesis and presented in the following sections belong to the first category. Their input data is biosignal that needs to be segmented first and they have to classify each segment.

5.1 EEG processing to predict epileptic seizure

EEG recording captures the electrical activity of the brain. EEG signal conveys essential information about the mental status, brain disorders, attention and meditation level, etc. [MFL+16]. Several applications use EEG signal to monitor or improve the health and wellness of users. Detection of sleep (transition from wakefulness to sleep) and monitoring the deepness of sleep are among the applications. The stress monitoring during the daily activity is another application which uses EEG signal.

One important application for EEG is the detection or prediction of epileptic seizures. Epilepsy is a chronic neurological disorder that, according to the Epilepsy Foundation, affects more than 65 million people worldwide [Sha]. It is associated with recurrent and sudden seizures which are due to temporary electrical disturbance and excessive neuronal discharge in the brain [Ias03]. Epileptic seizures result in altered consciousness or a whole body convulsion with uncontrolled movements. Due to the random nature of seizure occurrences, they are life-threatening and may increase the risk of serious injuries, especially if they occur while the patient is driving, exercising, etc. [LE02].

Figure 5.1 shows the high-level overview and stages for developing the seizure prediction method using machine learning classification based on the general method in [ANRS17]. The

EEG signal can be acquired from implantable devices or on-skin electrodes. The segmentation includes buffering the data for a specific window size (e.g. 60 seconds of EEG signal in [BWA+16]). The subsequent operations are performed on this chunk of data. Band-pass filters are used either to remove higher frequencies and noise or to obtain the frequency sub-bands. Many features including time-domain and frequency-domain information as well as the correlation between channels can be extracted from EEG channels, but not all have the same importance. The next step is to select those features with maximum relevance and minimum redundancy. Afterwards, the classifiers (e.g. support vector machine, artificial neural network, logistic regression, etc.) are trained. The output of the classifiers are smoothed using post-processing techniques [ANRS17]. If the performance of the model is not satisfactory, then the parameters are changed and a new model will be trained.

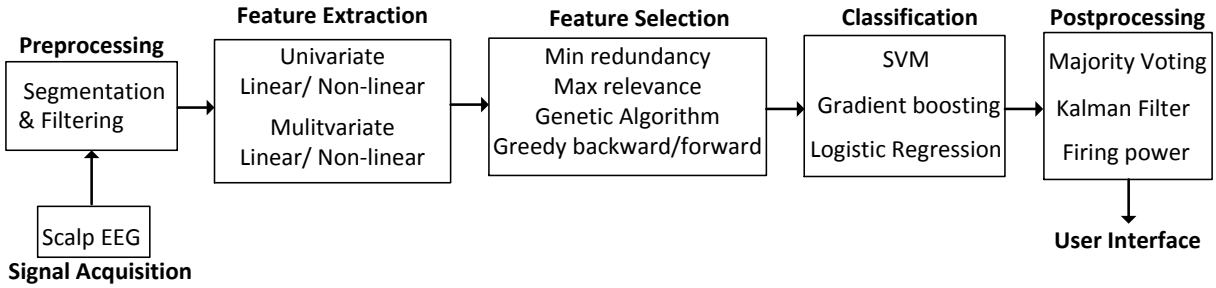


Figure 5.1: General overview of the stages for seizure prediction using EEG signals adopted from [ANRS17].

Once an accurate trained model is obtained, the setting of classifier can be ported to the IoT low-power device. Then, at runtime the captured EEG signal by the IoT device can be processed (i.e. applying filter, extracting the feature, etc.) and classified using the classifier. Figure 5.2 shows the overview of the epileptic seizure prediction algorithm proposed for constrained IoT devices in the scope of this thesis. New and simple features are introduced that allow elimination of other complex features and reduce the number of required EEG channels while still achieving accurate prediction. The proposed model respects the scarce internal memory and constrained computation resources on the low-power IoT devices and still fulfills the classification performance. The detailed presentation of the proposed system is published in [SPBH18].

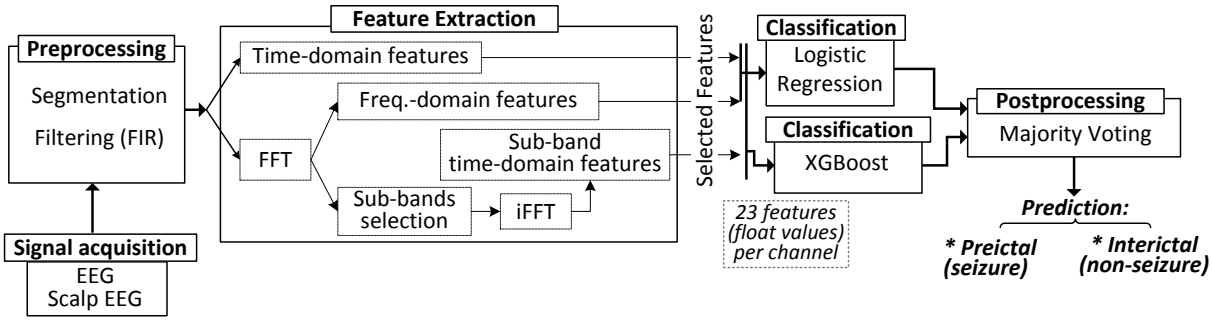


Figure 5.2: General overview of the stages for the proposed seizure prediction method

5.2 ECG processing to detect heart abnormality

ECG monitoring has recently received considerable industrial and academic interest [BGC13]. Analyzing the ECG signal, including heart rate or heart rate variability monitoring, provides essential information about the mental and physical condition of user which helps in early detection or diseases.

Figure 5.3 shows the structure of an ECG signal that consists of various peaks, waves and complexes. The fiducial points of an ECG signal are P, Q, R, S, and T as labeled in the figure. The duration and height of these peaks as well as the time intervals between them convey essential information about the heart [Con03]. These features can be calculated after locating the crucial points. R peak detection has received more attention mainly due to two reasons: for some applications, only the features that are related to the QRS complex (e.g. heart rate, hear rate variability [SRVMAA15], bundle branch blocks [WPW30], etc.) are required. Moreover, the R peak is the starting point for detecting other waves in most algorithms.

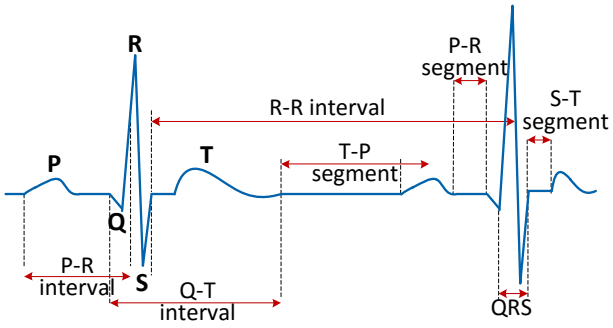


Figure 5.3: An ECG complex and its crucial points: P, Q, R, S, and T. Other features are calculated based on these crucial points [EEDA14].

Figure 5.4 illustrates the steps to process the ECG signal and delineate the crucial points and classify the heartbeats. After buffering one segment of ECG data (e.g. 3 seconds long), the data is filtered to remove the high-frequency noise. Then, the discrete wavelet transform is applied to the signal. The detail coefficients (from the high-pass filter) and approximation coefficients (from low-pas filter) of the wavelet transform can help removing the baseline wander from signal which has been introduced due to the motion artifact or muscle movements. The 'square and integrate' highlights the difference between peaks. Then, by exploiting an adaptive thresholding technique the R peaks can be detected. Once the R peak is detected and located, a forward and backward search is used to locate other peaks and waves.

5.3 Physical activity monitoring

Detecting the physical activity of user has several applications in the healthcare, wellness and fitness. For instance, sedentary behavior is proven to be hazardous to the health. It contributes to a number of health concerns including cardiovascular or heart disease, type 2 diabetes, obesity and some cancers [CCGT14].

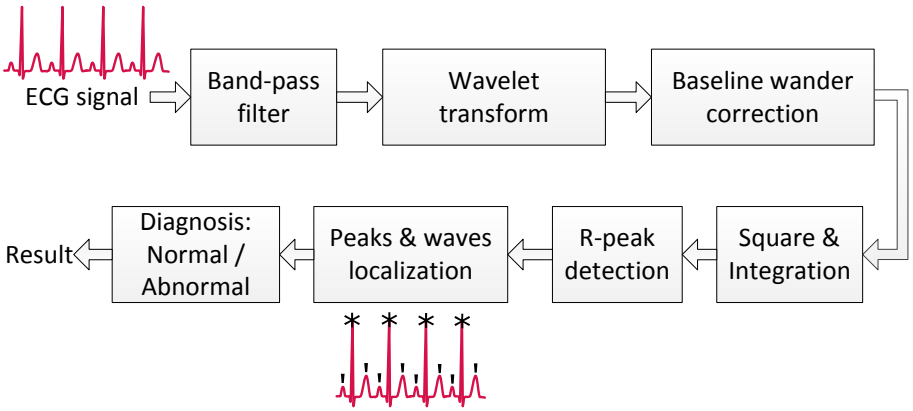


Figure 5.4: ECG processing application to detect the heart abnormalities

The accelerometer and gyroscope sensors can provide information about the movement, velocity and acceleration in different directions. Many wearable devices are equipped with such sensors. The input data include 3-axis (X, Y and Z) angular position from gyroscope and 3-axis acceleration. These input streams can be processed to classify the status of the body into 'running', 'sitting', 'walking', 'standing', 'cycling', etc.

Table 5.1 shows all the 35 features from three axes that were examined for the classification: mean, root mean square (RMS), zero crossing rate (ZCR), diff, variance, standard deviation and mean crossing rate (MCR). Among these features f15, f16, f17 are excluded and removed for the final classification as the Z-axis has shown an unpredictable noise. The other values related to the Z-axis can be used as they only measure comparative values.

Table 5.1: The set of features initially were examined

	mean	rms	zcr	diff	var	std	mcr
x-Axis	f1	f2	f3	f4	f5	f6	f7
y-Axis	f8	f9	f10	f11	f12	f13	f14
z-Axis	f15	f16	f17	f18	f19	f20	f21
pitch	f22	f23	f24	f25	f26	f27	f28
$\sqrt{x^2+y^2}$	f29	f30	f31	f32	f33	f34	f35

Figure 5.5 illustrates the stages to process the accelerometer and gyroscope data in order to detect and monitor the physical activity of user. After acquiring the input data (3-axes accelerometer), it is buffered and segmented. The optimized segment size is a parameter that needs exploration to be found. Then the buffered data is filtered using a median filter. Next stage extracted the features from X, Y and Z axes of accelerometer data. Different classifier types were investigated to find the optimized classifier (with low memory overhead, high accuracy and low execution time). The post-processing includes majority voting to smooth out the output.

Three difference types of classifiers are implemented and evaluated: Decision Tree (DT), Support Vector Machines (SVM) and Artificial Neural Networks (ANN). The physical activity mon-

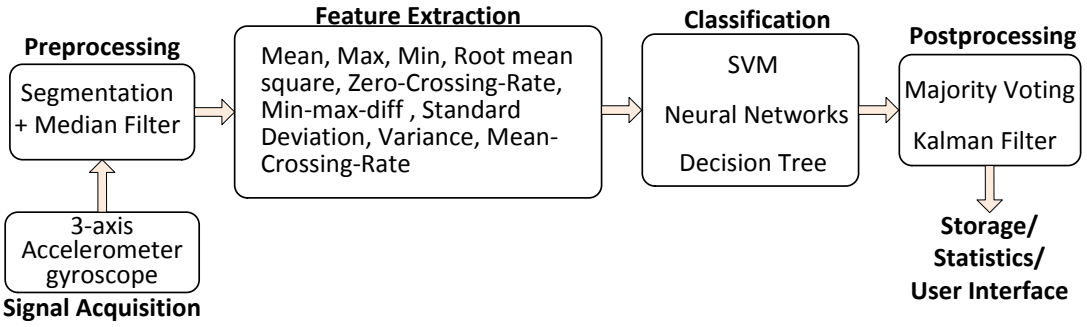


Figure 5.5: Stages to detect and classify the physical activity using accelerometer and gyroscope data

itoring application is implemented on the IoT prototype that has been developed in the scope of this thesis (see Section A.2). The accuracy of classifier, size of memory to store the setting of classifiers, and execution time for classification are measured on the designed prototype that features an ATmega328 microcontroller. Table 5.2 compares these three classifiers and shows the optimized size of buffered window (also known as segment). The settings of the SVM classifier require more memory than what is available on the designed prototype, therefore the execution time cannot be measured.

Table 5.2: Comparison of Classifiers

Classifier	Accuracy	Window (s)	Memory for Settings	Exec. Time (ms)
DT	91,4	5,0	328 Byte	1,90
ANN	86,0	3,0	592 Byte	13,55
SVM	91,3	4,5	100212 Byte	—

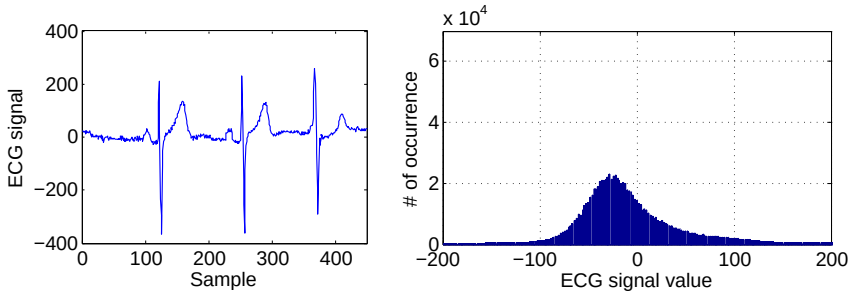
5.4 Approximate Compression for Health Monitoring Applications

As discussed in Section 3.5, many IoT applications inherently tolerate errors in their data processing and therefore deal with approximation. Since they are interacting with the physical world, their input data is usually noise [S+14]. Also their ADC converter introduces a quantization error before the data processing starts. Another stage that potentially deals with the approximation is the classification stage where the small errors might masked (i.e. slightly different inputs to the classifier can produce the same result [BMB+14, BBM+15, LDXW13, WJSX15]).

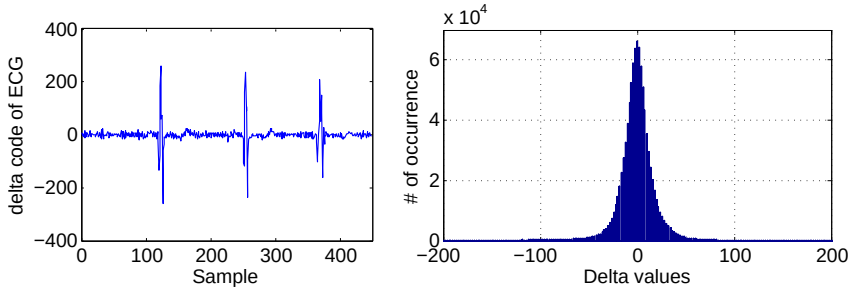
This section takes advantage of error tolerance in such applications and presents an approximate compression technique for biosignals in wearable IoT devices. The approximate compressor is based on Huffman coding but with a much smaller dictionary size and higher compression ratio. It finds the shortest code to compress the data while keeping the error in an (application-specific) acceptable range. This section uses ECG biosignal to explain and present the approximate compressor, but other biosignals such as EEG, EMG, etc. are also applicable. The proposed technique has been initially published in [SBH15].

5.4.1 Motivation

Figure 5.6a shows three seconds of an ECG signal (left) and the histogram of ECG values recorded for 2 hours, 10 minutes and 12 seconds (right). The ECG signal values have a Gaussian distribution, which indicates that some values (near zero) occur noticeably more often than the others. Gaussian distributions are popular candidates for being compressed by Huffman coding, which assigns short codes to high frequently occurring values and long codes to those that occur infrequently [YVH11, CHY04, PBM13]. Moreover, in a wide range of applications and systems, instead of transferring an entity (file, frame, value, etc.) in its entirety, only the difference with a reference (e.g. its previous value, a fix entity, or its previously transferred entity) is sent. Delta coding is also advantageous for ECG signals and can be exploited for storing and transmitting them [KYM+10, FG08]. Instead of sending absolute ECG sample values, the difference with the previous sample value is calculated, i.e. $d_i = s_i - s_{i-1}$. By having the initial value (i.e. s_0) and the sequence of delta values, the original ECG signal can be decoded and reconstructed. Figure 5.6b shows the delta coded ECG signal on the left side and the histogram of delta values on the right side (both correspond to the ECG signal in Figure 5.6a). The distribution of delta values implies that Huffman coding offers higher compression ratio for delta values than raw ECG values [FG08, KYM+10].



(a) A short sample of an ECG signal (left) and the data distribution as histogram for more than 2 hours of recording (right)



(b) Delta coding of the same ECG signal as absolute values (left) and histogram (right)

Figure 5.6: ECG signal (left) and its histogram (right) as absolute values (top) and delta values (bottom)

– Key Observation: Table 5.3 shows the code word length for some delta values compressed by Huffman coding. For example, the code word length for delta values 77 is 11 bits, while the code length of 78 is 14 bits. Assuming that the actual data can accept a slight variation of ± 1 (which is the case for ECG), delta values 78 and 81 can be replaced by 77 and 80 and thus be encoded by only 11 and 12 bits, which can save 3 and 2 bits, respectively.

Table 5.3: Code word length for some delta values

Delta Value	⋯	77	78	79	80	81	⋯
Code Length [bit]	⋯	11	14	13	12	14	⋯

This example demonstrates a significant potential for compression, and consequently, energy saving (for storing or transmission). Accordingly, this motivates the **key idea of *approximate compression*:** a novel data compression scheme for inherently error resilient signals such as biomedical signals (e.g. ECG) that can tolerate errors as long as the error amplitude is small enough and subsequent data processing (e.g. ECG delineation) is still possible. As it will be shown, approximating successive delta values may result in an aggregated error in the reconstructed signal, which can lead to an unacceptable error range. Therefore, this section proposes a method to keep the error in an acceptable range.

5.4.2 Details of Approximated Compressor

Assuming that the analog-to-digital converter (ADC) has W bits resolution, the ECG values can be represented by integer values in the range $[-2^{W-1}, 2^{W-1}-1]$. Let $s_i \in \mathbb{Z}$ denote the i^{th} ECG sample, $-2^{W-1} \leq s_i \leq 2^{W-1}-1$, and let us assume that for the target application an approximated value $s'_i = s_i + \varepsilon_i$ is accurate enough as long as $e_L \leq \varepsilon_i \leq e_H$, where $e_L, e_H \in \mathbb{Z}$ are the lower and upper bound of tolerable error, respectively (e.g. $e_L = -2, e_H = 2$ for a tolerable error of ± 2). This tolerable error is defined and given by the application and system designer for which the requirements of system for detection accuracy, robustness, etc. is taken into account. Given the basic definitions of delta codes (i) and approximated values (ii) on the left side of Equation (5.1), the right side shows that the delta value can tolerate the same error $\varepsilon_i = s'_i - s_i$ ($e_L \leq \varepsilon_i \leq e_H$) that was specified for the ECG samples. Thus instead of the exact delta value d_i, an approximate delta value $d'_i = d_i + \varepsilon_i$ can be used.

$$\begin{aligned} &(i)\ \ s_i = s_{i-1} + d_i \\ &(ii)\ \ s'_i = s_i + \varepsilon_i \\ &\qquad\ = s_{i-1} + d_i + \varepsilon_i \\ &\qquad\ = s_{i-1} + d'_i \end{aligned} \qquad (i),(ii) \Rightarrow s'_i - s_i = d'_i - d_i \tag{5.1}$$

This offers the opportunity to exploit the error tolerance of delta values by intentionally adding a permitted error to a delta value such that the resulting approximated delta value has a shorter Huffman code. This is shown in Equation (5.2), where $cl(d)$ represents the Huffman code length of a delta value d.

$$d'_i = d_i + \varepsilon_i : \quad cl(d'_i) = \min_{e_L \leq j \leq e_H} \big(cl(d_i + j)\big) \tag{5.2}$$

However, the calculation $s'_i = s_{i-1} + d'_i$ in Equation (5.1) assumes that the *exact* previous absolute value s_{i-1} is still available when decoding the approximated value s'_i. In practice, to be able

to actually benefit from the shorter Huffman codes, only the approximated value s'_{i-1} will be transmitted and thus available when decoding the delta value.

The entire flow is shown in Figure 5.7. The approximate compressor first calculates the delta value d_i, applies an intentional error ε_i to obtain the approximated delta value d'_i and then either directly transmits its Huffman code c_i or temporarily stores it in a non-volatile memory (e.g. in case the wireless connection could not be established). The receiver decodes the Huffman codes into the approximated delta values d'_i and then calculates $s'_i = s'_{i-1} + d'_i$ to retrieve the approximated ECG values s'_i.

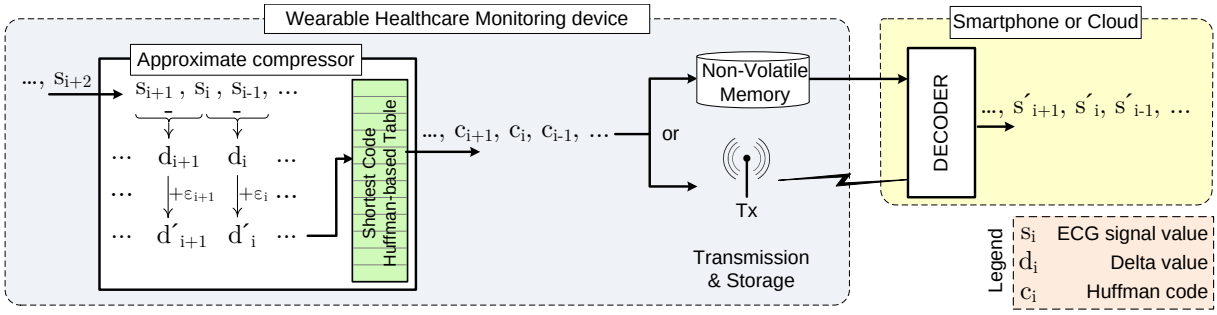


Figure 5.7: The scheme of the compression and coding method

The problem with this procedure is that it will lead to accumulated errors. For instance, if a particular sequence of approximated delta values always uses the largest tolerated error (i.e. $\forall i : \varepsilon_i = e_H$) then this will lead to an *error runaway* of the absolute reconstructed value. Let us consider a sequence $(d'_i, d'_{i-1}, \ldots)$ of approximated delta values $d'_i = d_i + \varepsilon_i$. Equation (5.3) calculates the error $E(s'_i)$ when comparing the approximated sample $s'_i = s'_{i-1} + d'_i$ with the exact sample $s_i = s_{i-1} + d_i$. The obtained recursion shows how errors are accumulated.

$$\begin{aligned} E(s'_i) = s'_i - s_i &= (s'_{i-1} + d'_i) - (s_{i-1} + d_i) \\ &= (d'_i - d_i) + (s'_{i-1} - s_{i-1}) \\ &= \varepsilon_i + E(s'_{i-1}) \end{aligned} \tag{5.3}$$

In the following, an effective scheme is presented to guarantee that the reconstructed ECG sample s'_i is always in the given tolerance range $[e_L, e_H]$ despite the error accumulation. Let us assume that $E(s'_{i-1})$ is within the tolerated error range (by using the proposed scheme), i.e. $E(s'_{i-1}) \in [e_L, e_H]$. Then the question is which error is tolerable for the next sample. Equation (5.4) shows how the bounds for the error ε_i of the next sample need to be modified, depending on the particular error $E(s'_{i-1})$ that was accumulated so far.

$$\begin{aligned} e_L \leq \varepsilon_i + E(s'_{i-1}) \leq e_H \\ \Rightarrow (e_L - E(s'_{i-1})) \leq \varepsilon_i \leq (e_H - E(s'_{i-1})) \end{aligned} \tag{5.4}$$

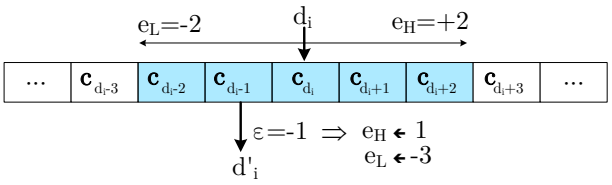


Figure 5.8: Approximating delta value with an error in acceptable range, and updated error range

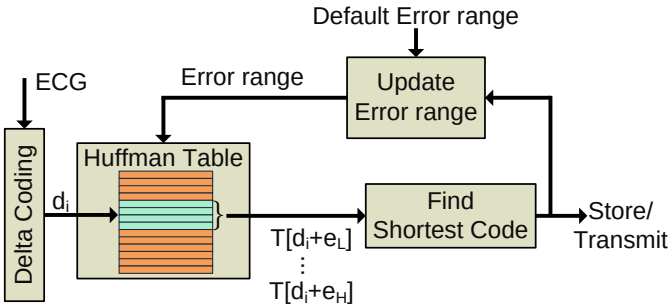


Figure 5.9: The proposed approximate compressor

Figure 5.8 shows an example where the tolerable error range is $e_L = -2$ and $e_H = +2$. The acceptable error that leads to the shortest Huffman code for this delta value is $\varepsilon = -1$. Therefore, the error range should be updated to $e_L = -2-(-1) = -1$ and $e_H = 2-(-1) = 3$.

5.4.3 Reducing Computational Overhead

Figure 5.9 shows the flow of approximate compression. For each delta value, depending on the error range, the adjacent Huffman codes are taken into account to find the shortest code. In the following, a technique is proposed to reduce the overhead introduced to find the shortest code.

As shown in Equation (5.2), finding the the best approximate delta value involves $(e_H - e_L + 1)$ memory reads accessing the Huffman table, as well as a total of $(e_H - e_L)$ comparisons to find the minimum code length. This must be done for every ECG sample, which would not only decrease performance and throughput, but would also cost energy. Therefore, a scheme is proposed to reduce this overhead, resulting in a more efficient approximate compressor architecture in which only one memory access is required for compressing one delta code.

While the original approximation scheme needs to search and find the shortest code in the tolerated error range for every delta value, a modified Huffman table can be used which contains the pre-computed shortest codes for the tolerated error. Let us first assume that the error range for each individual sample would be constant, i.e. $\varepsilon \in [e_L, e_H]$. The new table has two entries for each delta value d_i:

- shortest code in the tolerated error range sc_i:
  $sc_i = \text{argmin}\ \{cl(d_i + j)\}_{e_L \le j \le e_H}$
- bias b_i: the difference between d_i and the delta value whose Huffman code has been chosen as sc_i.

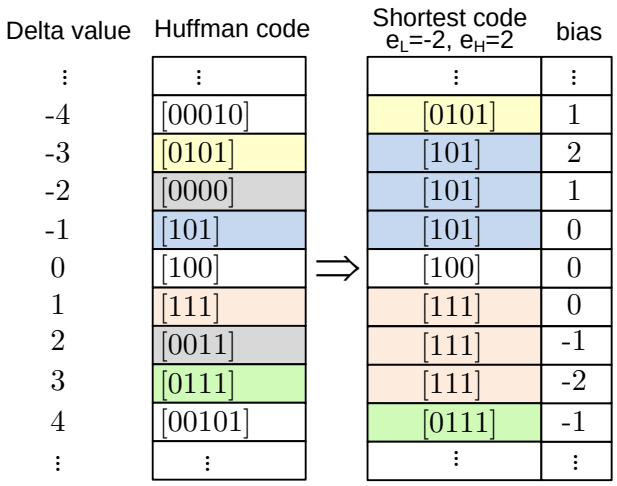


Figure 5.10: An SC table corresponding to the Huffman table and delta values

Bias values keep track of the amount of tolerable error used by a particular delta value. This is required to ensure that subsequent delta values remain within the tolerable error, as will be explained in the following subsections.

Figure 5.10 provides an example of how the modified Huffman table is created. It shows some delta values and their corresponding Huffman codes on the left side (i.e. the original Huffman table). Then for each delta value the shortest Huffman code in the error range $[-2,+2]$ is chosen. For delta values -1, 0 and 1, the shortest code is the original Huffman code, which means that these delta values are used with no approximation and hence, their corresponding bias values are 0. On the contrary, for delta value 3 the Huffman code that corresponds to delta value 1 is used. Thus, the delta value for 3 is approximated with a bias of -2.

Online and offline table construction

If the original Huffman table is known in advance, the shortest code (SC) table can be precomputed offline and stored in the memory. Alternatively, it could be built at runtime by doing the computation once and only once for each delta value. When a delta value appears for the first time, then its shortest code *sc* and bias value *b* would be computed by taking its adjacent delta values in the permitted range into account. The computed *sc* and *b* values would then be stored in the table for the next references to this delta value.

How to Use the Approximate Huffman Table

As explained in Section 5.4.2, the tolerable error range $[e_L, e_H]$ changes for successive approximate delta values. However, to construct the SC table in Figure 5.10 a constant error range is assumed, and all entries are calculated considering the default tolerable error, i.e. $[e_L, e_H]$. To deal with this issue, an *indexed addressing* is presented to obtain the actual table entry. As introduced in Section 5.4.2, $E(s'_{i-1})$ corresponds to the accumulated error of all approximated delta values including d_{i-1}. To calculate the tolerable error range for d_i, Equation (5.4) implies that

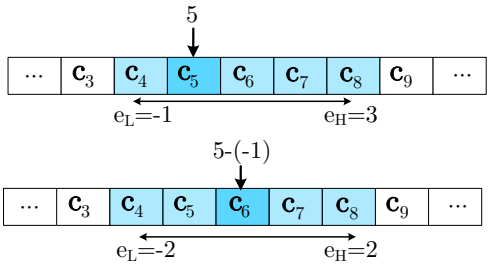


Figure 5.11: Index addressing to keep the default SC table usable for successive approximation. Accumulated error is -2, and the error range is ±2.

Algorithm 3: Compressing operation using shortest code table

```
Sp ← getSample();  // get the first signal's sample
bias ← 0;  // initial bias value is zero
while (True) do
    Sn ← getSample();  // new sample
    d ← Sn - Sp;  // delta value
    Sp ← Sn ;  // previous sample
    sc ← sc_table.get_code(d-bias);  // get shortest code
    bias ← sc_table.get_bias(d-bias);  // update bias
    packet.payload ← sc;
    send(packet);  // send the data
end
```

$\varepsilon_i \in [e_L - E(s'_{i-1}), e_H - E(s'_{i-1})]$, which basically means that for approximating d_i it is needed to examine the range $[d_i + e_L - E(s'_{i-1}), d_i + e_H - E(s'_{i-1})]$.

$$\left[\left(d_i - E(s'_{i-1})\right) + e_L, \quad \left(d_i - E(s'_{i-1})\right) + e_H\right] \tag{5.5}$$

According to Equation (5.5), looking up the Huffman code for $(d_i - E(s'_{i-1}))$ instead of d_i serves the purpose. Actually when looking up $(d_i - bias_{i-1})$, this corresponds to the accumulated error. Therefore the SC table that is constructed for a default error range of $[e_L, e_H]$ can be still used, by exploiting this indexed addressing approach.

As an example, consider the SC table shown in Figure 5.10. If the first delta value is 2 then the bias would be -1. The updated error range must be $e_L = -1$ and $e_H = 3$. If the next delta value is 5, it is needed to investigate the delta value range $5-1$ to $5+3$. When looking for the entry of $5-(-1) = 6$ in the SC table, as a matter of fact, it covers the delta value range of $6-2$ to $6+2$ (see Figure 5.11). Algorithm 3 describes how to use the SC table at runtime for the proposed approximate compression.

5.4.4 Table Size Reduction

The collected ECG signals of the same person can change from day to day, because of changes in person's activity, conductivity of electrodes, muscle contraction, etc. Figure 5.12 shows the histogram of delta values corresponding to the ECG signal of one person for two different days

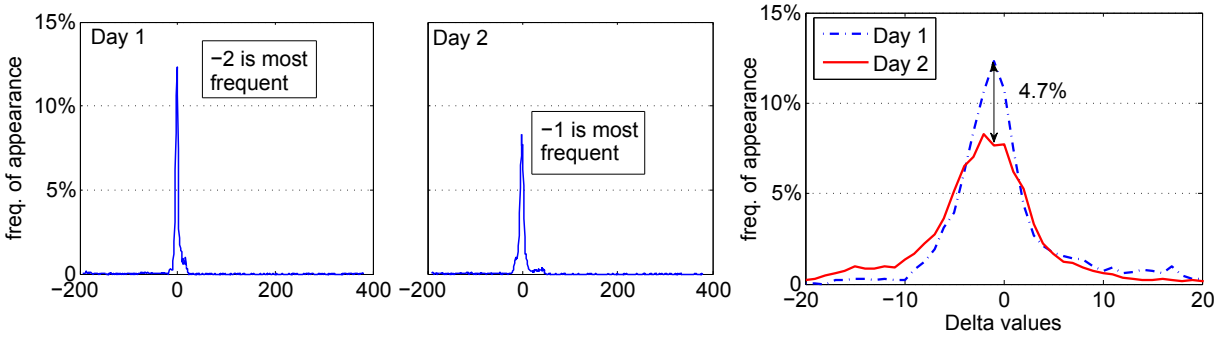


Figure 5.12: The appearance frequency of delta values of the ECG recordings of the same person for two different days

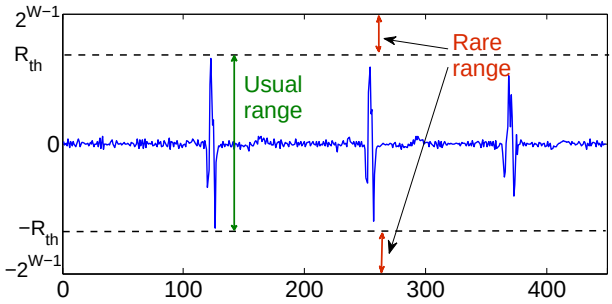


Figure 5.13: The whole range of delta values can be divided into two groups: rare range and usual range

measured on the ECG monitoring prototype that was developed in the scope of this thesis (see Section A.2). As shown in the figure, not only the occurrence frequency of delta value '-2' is reduced by 4.7%, but this delta value is not the most frequent value anymore. Changing occurrence frequencies of delta values can lead to a reduction in compression ratio, which demands updating the SC table (or generally the Huffman table). The SC table can be constructed (updated) either on the wearable device or on the Smartphone (server) and then transmitted to the wearable device. Due to the high resolution in recorded data, the range of ECG values, and consequently delta values, is large. It results in a large SC table (on average 1.47 MBits in the experiments). A simple but effective approach is used to reduce the size of the SC table (or generally Huffman table) while still retaining the same compression ratio.

Although the delta values can theoretically vary from -2^{W-1} to $+2^{W-1}-1$, in practice the appearance of large negative and large positive values is extremely rare, according to comprehensive observations. As shown in Figure 5.13, the delta values can be divided into two groups: 1) the *rare range* which includes large negative and large positive values that appear rarely, and 2) the *usual range* which includes the values that appear frequently. The delta values greater than a pre-defined threshold R_{th} or less than $-R_{th}$ are considered as rare values.

It is worth noting that Huffman codes are proved [Bur93] to have an upper bound that is shown in Equation (5.6), where p_1 and p_2 are the probabilities of the least and second least frequent values, respectively, and n is the number of different values to be compressed (i.e. alphabets). According to the probability distribution of delta values, p_1 and p_2 are too small and tend to zero. Hence, the upper limit approaches $\min\{\lfloor log_{\Phi}(1/p), n-1\}$, $p = p_2 \approx p_1$. As $p \to 0$, the upper limit of Huffman code approaches $(n-1)$.

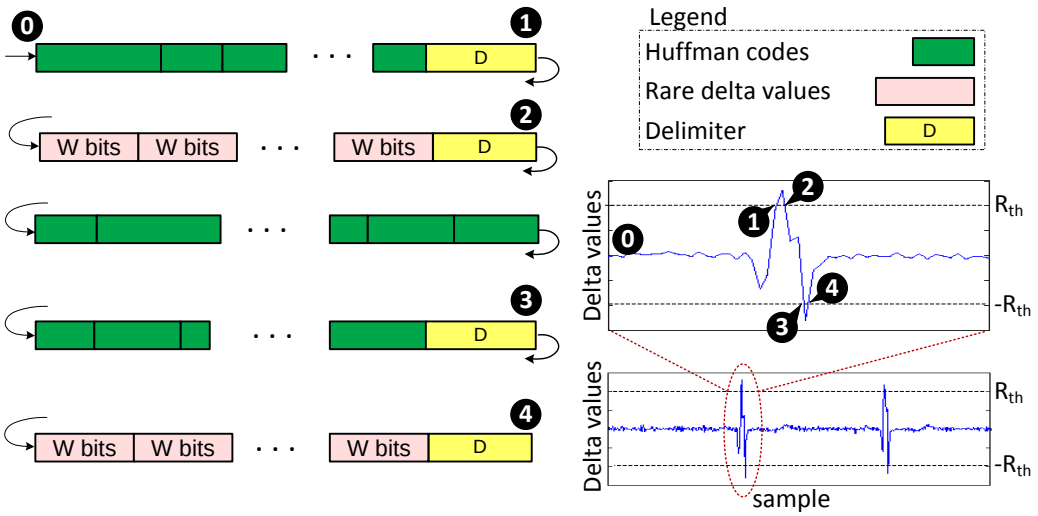


Figure 5.14: Rare delta values are not coded by Huffman, instead, a delimiter is used to distinguish the Huffman codes and W-bit-codes

$$\min\left\{\left\lfloor log_{\Phi}\left(\frac{\Phi+1}{\Phi\times p_1+p2}\right)\right\rfloor, n-1\right\}, \quad \Phi=\frac{1+\sqrt{5}}{2} \tag{5.6}$$

In order to reduce the size of the SC table (or Huffman table), only the usual range of delta values is considered. For the rare range of values, the Huffman coding is not used, but the raw W bits are stored. A specific code word of the Huffman table must be used as the delimiter to distinguish these two different ranges and codings. Figure 5.14 shows an example where some rare delta values appear. As these delta values do not have Huffman codes, they are presented in the original W-bit width codes. A delimiter is used to separate Huffman codes and raw data, such that the data can be decoded.

In this hybrid approach, the delta values in the rare range are represented by only W bits instead of a long Huffman code, however, they need a delimiter to separate them from Huffman codes. It should be noted that this hybrid coding approach will not necessarily increase or decrease the compression ratio. This is due to the fact that delta values in the rare range appear extremely infrequent, and therefore they do not contribute to the compression ratio significantly. The main benefit of this approach is a significant reduction in table size as evaluated in Section 5.4.5.

5.4.5 Evaluation and Results

Experimental Setup

To evaluate the proposed technique, the data from the MIT-BIH Long-Term ECG Database (ltdb) and MIT-BIH Arrhythmia Database (mitdb) [GAG+00] are used. The Long-Term ECG Database provides 7 sets of two-channel ECG signals sampled at 128 Hz with 12-bit resolution for almost one day. The Arrhythmia database provides 48 sets of two-channel ECG signals sampled at 360 Hz with 11-bit resolution for half an hour. The first 40,000 samples of each

ECG recording are used for training to construct the Huffman tables, while the rest are used for evaluating the compression rate of proposed technique and state-of-the-art.

Comparison with state-of-the-art

The proposed approximate compression approach for the Huffman technique can also be applied to other existing compression techniques, like [CHY04, SHLL14, MD06, PBM13], to further improve their compression ratio. In other words, these techniques and the proposed approximate compression are complementary solutions.

To show the effectiveness of the proposed technique, it is compared with two state-of-the-art compression method, i.e. QLV [KYM+10] and [PBM13]. Despite many other existing compression techniques, QLV [KYM+10] and the proposed approximate compression are mutually exclusive. The reason is that QLV uses Huffman codes as separators to distinguish different levels, and therefore, it can not exploit approximation. Thus, the proposed approach is compared against QLV. Instead, [PBM13] and approximate compression technique are orthogonal and can be used at the same time. Therefore, the approximate compression is applied on top of the technique proposed in [PBM13] to show the further compression ratio that can be achieved by the proposed approximation technique.

To quantify the compression performance, the compression ratio (CR) metric is employed which is defined as:

$$CR = \frac{S_{orig} - S_{comp}}{S_{orig}} \times 100 \tag{5.7}$$

where S_{orig} and S_{comp} represent the size of original and compressed data, respectively. An acceptable error range of up to ± 4 is considered, which has a quite negligible impact on the quality of the reconstructed signal.

Figure 5.15 shows the compression ratio achieved using an exact Huffman technique and the proposed approximate compression with different tolerable error ranges. As expected, the compression ratio is improved as the tolerable error range extends. Some ECG recordings, like '1', '2' and '4', show a large potential for approximate compression. For some other recordings, like '3', the compression ratio does not show large sensitivity to the tolerable error range (while still is beneficial). The reason behind this behavior is that the ECG recording '3' belongs to a person whose heart beats are perfectly normal and regular. On the contrary, the ECG recording '1' has some anomalies inducing irregularities in the data as a result of arrhythmia or unfiltered motion artifact [KKVH+14].

The achieved compression ratio of the proposed technique and QLV [KYM+10] are shown in Figure 5.16. For all the ECG recordings, the approximate compressor outperforms the state-of-the-art approach [KYM+10]. For ECG recording '10' the compression ratio of the proposed technique is about 75%, while QLV [KYM+10] achieves only 33%. The approximate compressor achieves up to 60% data reduction (corresponding to ECG recording '10') compared to

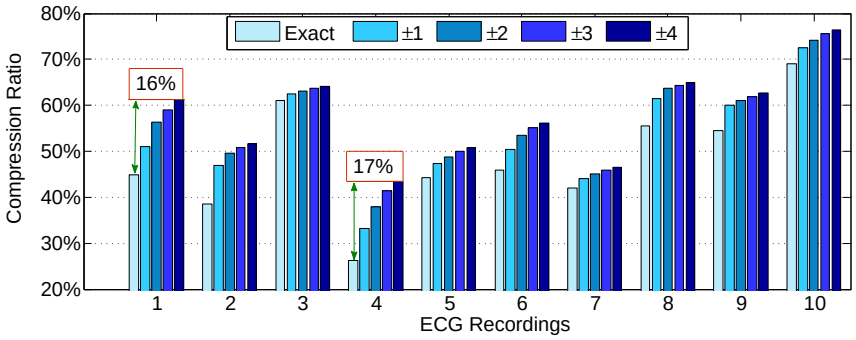


Figure 5.15: Compression ratio of the approximate compressor (for different acceptable error ranges) and exact Huffman compared to the uncompressed baseline

QLV, i.e. the data volume to be transmitted is 60% smaller when using the proposed approximate compression rather than using QLV.

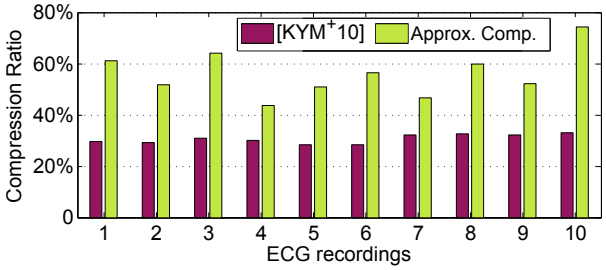


Figure 5.16: Compression ratio of the proposed approximate compressor and state-of-the-art [KYM+10] compared to the uncompressed baseline

Figure 5.17 illustrates the compression ratio improvement achieved by applying the approximate compression on top of the frame-based compression presented in [PBM13]. The effectiveness of [PBM13] highly depends on the accuracy of the R-peak detection, which makes it more vulnerable to motion artifacts and noise.

Table Size Analysis

A threshold of $R_{th} = 512$ is used to identify the delta values in the *usual range* (see Section 5.4.4). For the considered ECG recordings, more than 99.7% of delta values fall in the usual range. Using the proposed hybrid coding approach, it succeeds to reduce the table size significantly. Considering the full range of delta values for creating the SC table leads to a table size of up to 1.74 Mbits (1.47 Mbits on average). By exploiting the proposed hybrid approach, the size of the SC table decreases to at most 500 KBits (359 KBits on average).

5.4.6 Summary of Approximate Compression Technique

This section presented an approximate compression technique for biomedical signals (e.g. ECG) to reduce the energy consumption of data transmission in IoT health monitoring systems. This approximate compressor takes advantage of error tolerance in biomedical signals and finds the shortest Huffman code for each delta value in the user-specified acceptable error range. Then,

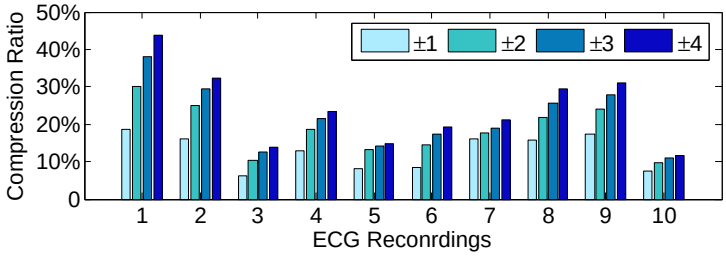


Figure 5.17: Compression ratio improvement of the proposed approximate compressor applied on top of [PBM13] compared to [PBM13] as baseline

it proposed a method to reduce the computational cost for finding the approximate result that is based on a pre-computed table and a mechanism to keep the accumulated error in the acceptable error range. Experimental results showed that the proposed technique achieved more than 60% reduction in wireless communication (i.e. data size that is to be transmitted) compared to the state-of-the-art approach [KYM+10], and more than 40% improvement when applied on top of the state-of-the-art approach [PBM13].

5.5 Summary of Application-Specific Optimizations

This chapter has presented application-specific optimizations for health monitoring domain. It first investigated some IoT applications in the healthcare domain which process bio-signal data. These applications include EEG processing for seizure detection, ECG processing for anomaly detection, and physical activity monitoring. It has studied the software structure of those implemented applications and shown that they have a similar pipeline structure. The general application model that was presented in Section 4.1 is based on these implemented applications. Then, a novel technique has been proposed to compress the bio-signal input data on wearable IoT devices. The proposed technique is for IoT applications that can tolerate a small amount of error in their input data (i.e. most of applications that process bio-signal). The proposed approximate compressor reduces the amount of data that is transmitted from IoT devices. These application-specific optimizations can be applied to the individual IoT devices to reduce their resource usage (i.e. local resources like storage and energy consumption) as well as the shared resources on the gateway (e.g. communication bandwidth).

6 Conclusion and Future Work

6.1 Conclusion

This thesis studied the efficient resource management for IoT devices to process the IoT-generated data at the edge of the network. First, it provided an overview of IoT technologies required from an embedded design perspective and specific properties associated with IoT in embedded systems' landscape. It investigated essential technologies for the development of IoT systems, existing trends, and its distinguishing properties. By discussing the key characteristics, main application domains, and major research issues in IoT, this thesis provided a comprehensive IoT perspective for embedded system design.

The thesis considered the operation modes of the IoT devices as a control parameter for managing the constrained resources on the edge of the network. These include the individual resources on the IoT devices such as battery or energy source and the shareable resources on the gateway such as communication bandwidth, processing power, and memory. The thesis presented some fast and efficient techniques for selecting the operation modes of IoT devices at runtime to achieve the system optimization goals while meeting the device and application requirements and respecting the shared resources.

In the multi-gateway IoT systems, the complexity of the binding and allocation problems does not allow an efficient optimal solution. This thesis addressed the problem by providing a trade-based management scheme that reaches sub-optimal solutions by improving the state of the system step-by-step.

To present application-specific optimizations, several IoT applications from healthcare domain have been studied. The selected applications represent the main applications in IoT domain whose input data is in form of signal. Based on the pipeline structure of such applications, a model has been presented that describes the processing stages that the input data passes through in order to retrieve the output result. Moreover, a novel approximate compression technique has been proposed for bio-signal data to reduce the amount of transmitted data. The approximate compressor can be used for wearable IoT devices in health monitoring application without requiring any hardware extension. The application-specific optimizations have been proposed for the individual IoT devices but they reduce the usage of resources both on the device and on the shared gateway.

The evaluation of the applications was done by using real-world input signals and implementation on the IoT platforms. The execution time, power consumption and resource usage of applications have been profiled. The data transmission under interference was also studied to

obtain the properties of communication bandwidth including the throughput. The overhead of operation mode selection techniques has been measured on the hardware platforms including Intel Quark SoC and Cortex-M4 microcontrollers. The measured parameters have been then used in a simulation environment to capture the system-wide behavior and interaction of devices with the gateway.

6.2 Future Work

The focus of this thesis was on modeling the IoT applications in the healthcare domain, defining operation modes to provide flexibility to manage the constrained resources on the devices as well as shareable resources on the gateway. Several promising research directions can be pursued as the follow-up work in the future. These directions include two major areas: (i) improving the efficiency of IoT devices, both in hardware design and application software design using the characteristics of applications, and (ii) enhancing the management schemes by exploiting learning techniques at the edge of the network.

– Hardware Optimization The IoT platforms can be improved in terms of energy efficiency by leveraging the properties of their applications. The opportunities are two-fold:

- Approximation: Most of the IoT devices operate in a noisy environment with noisy input data. Even the data processing of such application, inherently, involves a trade-off between the accuracy and computational effort. For instance, the classification stage may tolerate a level of inaccuracy and still deliver the same results. Exploiting the approximate computing concept in IoT applications, especially when dealing with input signals and classification tasks, can open opportunities to further increase the efficiency of IoT devices in terms of power and energy.
- Accelerators for classifiers: Many of IoT applications use machine learning techniques for classification of their input data. These classifiers often can benefit from parallelism, especially when using Boosting Trees where several independent trees must be traversed to obtain one score from each. To traverse each tree, several comparisons must be done which can be accelerated by parallel hardware implementation.

– Learning-based management of IoT devices Another research direction for management of IoT devices and mode selection is using machine learning techniques, and particularly, reinforcement learning. To make the IoT devices more self-dependent, their ability to make decisions should increase by making them smarter. Pushing the intelligence to the IoT end-nodes requires exploiting the light and efficient learning techniques on the edge devices.

A Appendix

In order to support the experiments of this thesis, several practical setups were devised. Section A.1 describes the practical setup to study the properties of Bluetooth Low Energy including the data throughput and effect of WiFi interference on it. Section A.2 describes and presents IoT prototypes that were designed and developed to study some IoT applications in the healthcare monitoring domain.

A.1 Wireless Transmission

The massive number of IoT devices will bring several issues. One of these issues is dense deployment of IoT devices which communicate by means of wireless radios. Transmission of data over the same frequency channel may lead to collision and wireless interference. Consequently, the transmitted packets will be corrupted and dropped. In this case, either the packet is lost (i.e. loss of data) or it will be re-transmitted which costs power and energy for the IoT device. Avoiding wireless interference is critical for IoT wireless technologies [SBH16]. Bluetooth Low Energy exploits a frequency hopping scheme to minimize the interference and collision with other surrounding BLE devices. Each two connected BLE devices use a random sub-channel for each interval. Even though the likelihood of interference between BLE devices is very low, the effect of WiFi devices needs to be studied as WiFi is using the same frequency band (i.e. 2.5 GHz).

To study the effect of wireless interference between WiFi and BLE, some experiments are conducted using Microchip RN4677 BLE modules. One module was configured as the master and the other one as the slave. Different parameters including *connection interval*, *slave latency*, and *supervision timeout* are explored. The maximum achievable throughput using these modules is 36 Kbps which is reached when the slave latency is 0, the supervision timeout is set to 160 ms and the connection interval is set to 12.5 ms. The WiFi access point was configured to operate in channel 11 (range of 2.452-2.472 GHz) which overlaps the BLE range for 20 MHz. The access-point was set to transmit data continuously while the BLE modules were sending and receiving the data at the configuration with the highest throughput. The experiment shows that with heavy WiFi interference, the throughput degraded slightly from 36 Kbps to 34 Kbps which is less than 6% degradation.

A.2 IoT prototypes

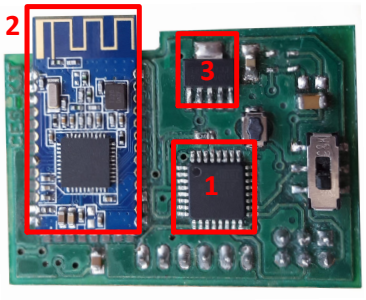
In order to study the IoT applications in the healthcare domain, several in-house IoT prototypes were design and developed. The main goals of these prototypes were to (i) investigate the research problems and challenges in development of wearable and portable IoT devices, (ii) create realistic scenarios and obtain real-world input data for the experiments and (iii) test and evaluate the applications on hardware platforms.

A printed circuit board (PCB) has been design and manufactured in the scope of this thesis for an IoT prototype that can support a few different applications. The features the main and necessary components of an IoT devices for wireless communication, controlling and processing, interfacing with physical world to capture the data.

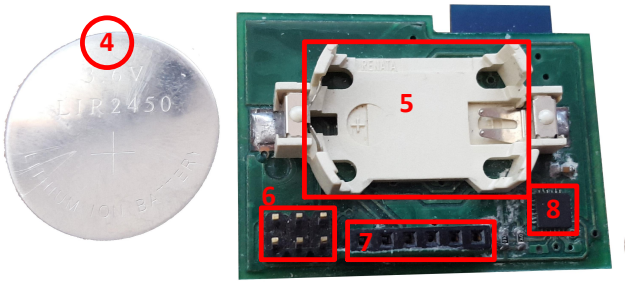
Figure A.1 shows the manufactured board and annotates the main components:

1. Microcontroller: a low-power ATMega328P chip
2. Wireless Radio: an HM-10 Bluetooth Low Energy module
3. The voltage regulator
4. Coin cell battery: a 3.6 V Lithium-Ion battery with the capacity of 120 mAh
5. Battery placeholder
6. Programming interface: Serial Peripheral Interface (SPI)
7. Analog Front End: the interface to connect the sensors (and actuators)
8. On-board accelerometer and gyroscope chip: MPU-6000 integrated 6-axis motion tracking device

Figure A.2 shows the layout of two layers of the PCB and Figure A.3 shows the schematic of the board. Note that on the schematic in Figure A.3 the Bluetooth module is HC-5 which is a classic Bluetooth module. But in the later version, an HM-10 BLE module was used with no need to change the PCB because both modules had the same layout, size and and interface.



(a) Front



(b) Back

Figure A.1: Picture of design board with peripherals, interfaces and main components

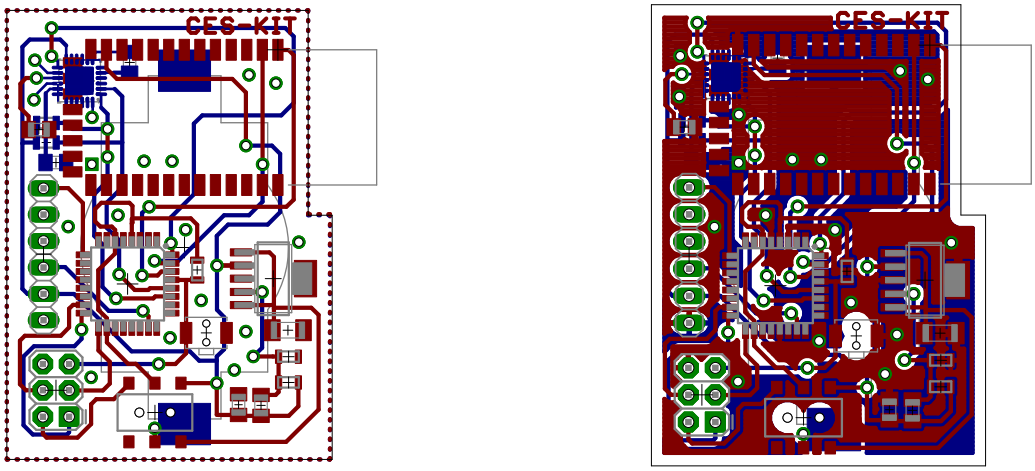


Figure A.2: Layout of the two layers for the designed PCB

The designed PCB is able to support different sensors. It featured an on-board accelerometer and gyroscope, but the analog front end can be connected to other sensors. It was used for different IoT applications including ECG monitoring, physical activity monitoring and smart shoe. Figure A.4a shows the ECG monitoring device which features a Sparkfun AD8232 (a single lead heart rate monitor board) to measure the electrical activity of the heart. It receives the input from electrodes and performs signal conditioning (amplification, filter noise, etc.). It is attached to the PCB board from analog front end interface. Figure A.4b shows the designed PCB being used for physical activity monitoring application by means of Force-Sensitive Resistor (FSR). The FSR is a flexible sensor that can measure the force. It can be placed under the heel while the PCB is worn on the ankle using a strap. The force on the heel is measured and

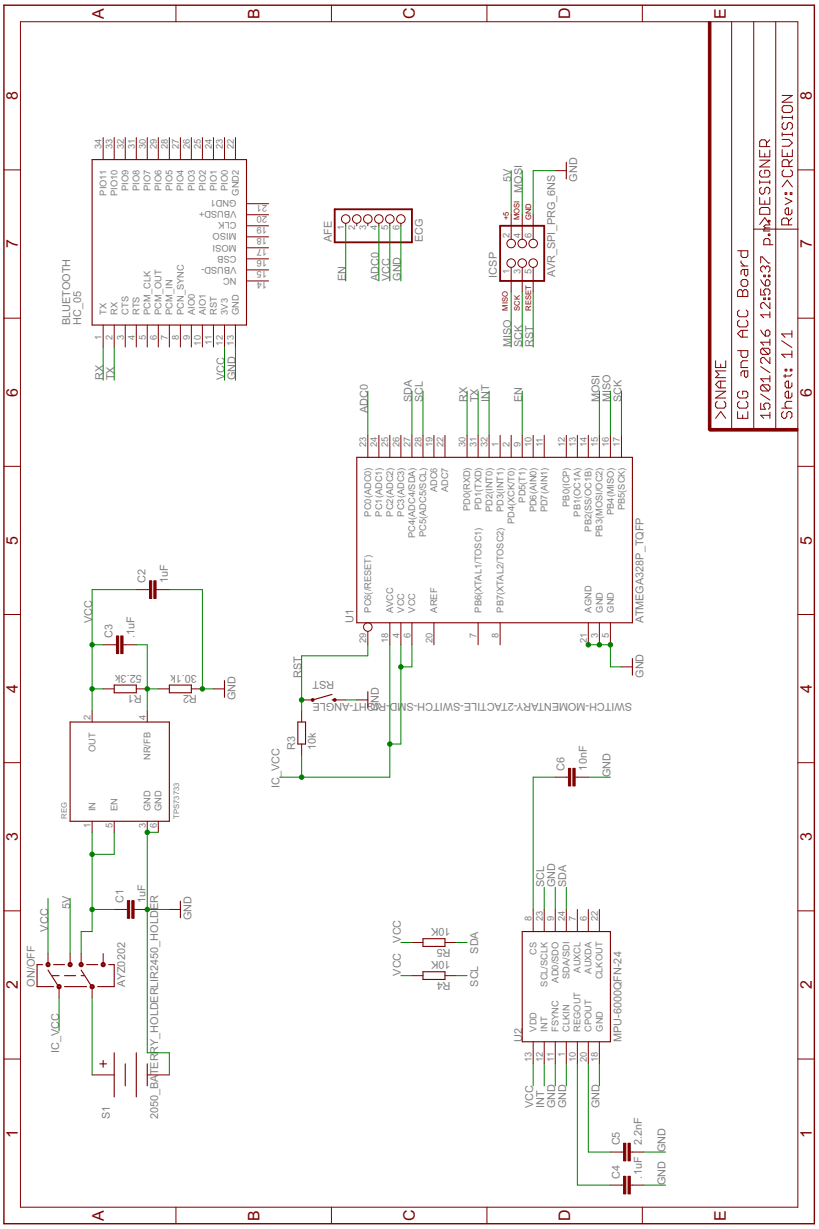
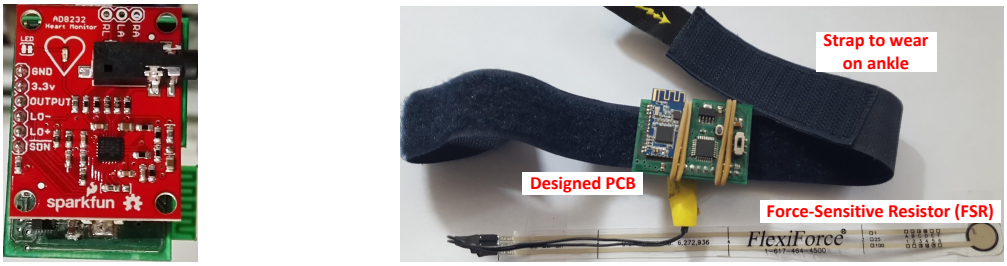


Figure A.3: Schematic of the two layers for the designed PCB

collected continuously to classify the the person's status: walking, running, standing, siting, etc. The data from FSR can be fuse with the accelerometer and gyroscope data to increase the accuracy.



(a) (b)

Figure A.4: Picture of prototypes using designed PCB: (a) ECG monitoring device using Sparkfun analog front end attached to the analog interface of PCB, (b) physical activity monitoring using the force-sensitive resistor to detect the pressure on the heel.

Bibliography

[AB15] Lisa Avila and Mike Bailey. The wearable revolution. *IEEE Computer Graphics and Applications*, 35(2):104–104, 2015.

[ABC+15] Mobyen Uddin Ahmed, Mats Björkman, Aida Causevic, Hossein Fotouhi, and Maria Lindén. An overview on the internet of things for health monitoring systems. In *IoT Technologies for HealthCare*, 2015.

[AFGM+15] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. Internet of Things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys & Tutorials*, 17(4):2347–2376, 2015.

[AIM10] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The Internet of Things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

[AMJ15] Shahab Ardalan, Siavash Moghadami, and Samira Jaafari. Motion noise cancelation in heartbeat sensing using accelerometer and adaptive filter. *IEEE Embedded Systems Letters*, 7(4):101–104, 2015.

[ANRS17] Elie Bou Assi, Dang K Nguyen, Sandy Rihana, and Mohamad Sawan. Towards accurate prediction of epileptic seizures: A review. *Biomedical Signal Processing and Control*, 34:144–157, 2017.

[AP13] Jong Hoon Ahnn and Miodrag Potkonjak. mhealthmon: Toward energy-efficient and distributed mobile health monitoring using parallel offloading. *Journal of medical systems*, 37(5):1–11, 2013.

[BAAR12] Rubén Braojos, Giovanni Ansaloni, David Atienza, and Francisco J Rincón. Embedded real-time ECG delineation methods: A comparative evaluation. In *International Conference on Bioinformatics & Bioengineering (BIBE)*, pages 99–104, 2012.

[BBC+14] Rubén Braojos, Ivan Beretta, Jeremy Constantin, Andreas Burg, and David Atienza. A wireless body sensor network for activity monitoring with low transmission overhead. In *IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, pages 265–272, 2014.

[BBM+15] Daniele Bortolotti, Andrea Bartolini, Mauro Mangia, Riccardo Rovatti, Gianluca Setti, and Luca Benini. Energy-aware bio-signal compressed sensing reconstruction: Focuss on the wbsn-gateway. In *IEEE 9th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, pages 120–126, 2015.

[BCM+15] S Benatti, F Casamassima, B Milosevic, E Farella, P Schonle, S Fateh, T Burger, Q Huang, and L Benini. A versatile embedded platform for EMG acquisition and gesture recognition. *IEEE trans. on biomedical circuits and systems*, 2015.

[BGC13] Mirza Mansoor Baig, Hamid Gholamhosseini, and Martin J Connolly. A comprehensive survey of wearable and wireless ECG monitoring systems for older adults. *Medical & biological engineering & computing*, 51(5):485–495, 2013.

[BI] Here's how the internet of things will explode by 2020. Online: `http://www.businessinsider.de/iot-ecosystem-internet-of-things-forecasts-and-business-opportunities-2016-2`. Visited on March 5, 2017.

[BMB+14] Daniele Bortolotti, Hossein Mamaghanian, Andrea Bartolini, Maryam Ashouei, Jan Stuijt, David Atienza, Pierre Vandergheynst, and Luca Benini. Approximate compressed sensing: ultra-low power biosignal processing via aggressive voltage scaling on a hybrid memory multi-core processor. In *ISLPED*, pages 45–50, 2014.

[BMB+15] Daniele Bortolotti, Mauro Mangia, Andrea Bartolini, Riccardo Rovatti, Gianluca Setti, and Luca Benini. An ultra-low power dual-mode ECG monitor for healthcare and wellness. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1611–1616, 2015.

[BMB+16] Daniele Bortolotti, Mauro Mangia, Andrea Bartolini, Riccardo Rovatti, Gianluca Setti, and Luca Benini. Energy-aware bio-signal compressed sensing reconstruction on the WBSN-gateway. *IEEE Transactions on Emerging Topics in Computing*, 2016.

[BMZA12] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the Internet of Things. In *workshop on Mobile cloud computing (MCC)*, pages 13–16, 2012.

[BSKH07] Lars Bauer, Muhammad Shafique, Simon Kramer, and Jörg Henkel. Rispp: Rotating instruction set processing platform. In *Proceedings of the 44th annual Design Automation Conference (DAC)*, pages 791–796. ACM, 2007.

[BSLR10] Sangeeta Bhattacharya, Abusayeed Saifullah, Chenyang Lu, and Gruia-Catalin Roman. Multi-application deployment in shared sensor networks based on quality of monitoring. In *16th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 259–268, 2010.

[Bur93] Michael Buro. On the maximum length of Huffman codes. *Information processing letters*, 45(5):219–223, 1993.

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[BWA+16] Benjamin H. Brinkmann, Joost Wagenaar, Drew Abbot, Phillip Adkins, Simone C. Bosshard, Min Chen, Quang M. Tieng, Jialune He, F. J. Muñoz-Almaraz, Paloma Botella-Rocamora, Juan Pardo, Francisco Zamora-Martinez, Michael Hills, Wei Wu, Iryna Korshunova, Will Cukierski, Charles Vite, Edward E. Patterson, Brian Litt, and

Gregory A. Worrell. Crowdsourcing reproducible seizure forecasting in human and canine epilepsy. *Brain*, 139, 2016.

[CCGT14] Jean-Philippe Chaput, Valerie Carson, Casey E Gray, and Mark S Tremblay. Importance of all movement behaviors in a 24 hour period for overall health. *International journal of environmental research and public health*, 11(12):12575–12581, 2014.

[CCRR13] Vinay K Chippa, Srimat T Chakradhar, Kaushik Roy, and Anand Raghunathan. Analysis and characterization of inherent application resilience for approximate computing. In *DAC*, 2013.

[CCT+15] Shao-Yi Chien, Wei-Kai Chan, Yu-Hsiang Tseng, Chia-Han Lee, V. Srinivasa Somayazulu, and Yen-Kuang Chen. Distributed computing in IoT: System-on-a-chip for smart cameras as an example. In *ASP-DAC*, pages 130–135, 2015.

[CDDM+12] Luca Catarinucci, Danilo De Donno, Luca Mainetti, Luca Palano, Luigi Patrono, Maria Stefanizzi, and Luciano Tarricone. An iot-aware architecture for smart healthcare systems. *IEEE Internet of Things Journal*, 2012.

[CG06] Sung-woo Cho and Ashish Goel. Pricing for fairness: distributed resource allocation for multiple objectives. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, pages 197–204, 2006.

[CH05] Lei Chen and Wendi B Heinzelman. Qos-aware routing based on bandwidth estimation for mobile ad hoc networks. *IEEE Journal on selected areas in communications*, 23(3):561–572, 2005.

[CH09] Nawal Cherfi and Mhand Hifi. Hybrid algorithms for the multiple-choice multi-dimensional knapsack problem. *International Journal of Operational Research*, 5(1):89–109, 2009.

[Che15] Xu Chen. Decentralized computation offloading game for mobile cloud computing. *IEEE Transactions on Parallel and Distributed Systems*, 26(4):974–983, 2015.

[CHY04] Wen-Shiung Chen, Lili Hsieh, and Shang-Yuan Yuan. High performance data compression method with pattern matching for biomedical ECG and arterial pulse waveforms. *Computer methods and programs in Biomedicine*, 74(1):11–27, 2004.

[cit] CityPulse: Real-time IoT stream processing and large-scale data analytics for smart city applications. Online `www.ict-citypulse.eu`.

[CLM+10] Shirley Coyle, King-Tong Lau, Niall Moyna, Donal O Gorman, Dermot Diamond, Fabio Di Francesco, Daniele Costanzo, Pietro Salvo, Maria Giovanna Trivella, Danilo Emilio De Rossi, et al. BIOTEX-Biosensing textiles for personalised healthcare management. *IEEE Trans. on Info. Tech. in Biomedicine*, 14(2):364–370, 2010.

[Col79] Jules L Coleman. Efficiency, utility, and wealth maximization. *Hofstra L. Rev.*, 8:509, 1979.

[Con03] Mary Boudreau Conover. *Understanding electrocardiography*. Elsevier Health Sciences, 2003.

[COO13] Vedat Coskun, Busra Ozdenizci, and Kerem Ok. A survey on near field communication (NFC) technology. *Wireless personal communications*, 71(3):2259–2294, 2013.

[CRMS09] Delphine Christin, Andreas Reinhardt, Parag Mogre, and Ralf Steinmetz. *Wireless Sensor Networks and the Internet of Things: Selected Challenges*, 2009.

[DBN14] Soumya Kanti Datta, Christian Bonnet, and Navid Nikaein. An IoT gateway centric architecture to provide novel m2m services. In *World Forum on Internet of Things (WF-IoT)*, pages 514–519, 2014.

[DMHL12] Thomas Dittrich, Chen Menachem, Y Herzel, and A Lou. Lithium batteries for wireless sensor networks. Technical report, Tadiran Batteries, 2012.

[DMSS15] Alessandro Dionisi, Daniele Marioli, Emilio Sardini, and Mauro Serpelloni. Low power wearable system for vital signs measurement in all day long applications. In *International Symposium on Medical Measurements and Applications (MeMeA)*, pages 537–542, 2015.

[Dom12] Mari Carmen Domingo. An overview of the internet of things for people with disabilities. *Journal of Network and Computer Applications*, 35(2):584–596, 2012.

[DXHL14] Li Da Xu, Wu He, and Shancang Li. Internet of things in industries: A survey. *IEEE Transactions on Industrial Informatics*, 10(4):2233–2243, 2014.

[EEDA14] Mohamed Elgendi, Björn Eskofier, Socrates Dokos, and Derek Abbott. Revisiting QRS detection methodologies for portable, wearable, battery-operated, and wireless ECG systems. *PloS one*, 9(1):e84018, 2014.

[FG08] Catalina Monica Fira and Liviu Goras. An ECG signals compression method and its validation using NNs. *IEEE Transactions on Biomedical Engineering*, 55(4):1319–1326, 2008.

[Fol15a] Joe Folkens. Building a gateway to the Internet of Things. Technical report, Texas Instruments, 2015.

[Fol15b] Joe Folkens. IoT gateways: Behind the scenes of smart cities. Online: `http://e2e.ti.com/blogs_/b/connecting_wirelessly/archive/2015/04/22/iot-gateways-behind-the-scenes-of-smart-cities`, 2015. Published on April 22, 2015, visited on December 14, 2016.

[GAG+00] Ary L Goldberger, Luis AN Amaral, Leon Glass, Jeffrey M Hausdorff, Plamen Ch Ivanov, Roger G Mark, Joseph E Mietus, George B Moody, Chung-Kang Peng, and H Eugene Stanley. Physiobank, physiotoolkit, and physionet components of a new research resource for complex physiologic signals. *Circulation*, 101(23):e215–e220, 2000.

[gar] Gartner says 8.4 billion connected "things" will be in use in 2017. Online: `http://www.gartner.com/newsroom/id/3165317`. Published: February 7, 2017, visited on August 1, 2017.

[GBMP13] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660, 2013.

[GH15] Mohamed T Ghoneim and Muhammad M Hussain. Review on physically flexible nonvolatile memory for internet of everything electronics. *Electronics*, 4(3):424–479, 2015.

[GKN+11] Alexander Gluhak, Srdjan Krco, Michele Nati, Dennis Pfisterer, Nathalie Mitton, and Tahiry Razafindralambo. A survey on facilities for experimental internet of things research. *IEEE Communications Magazine*, 49(11):58–67, 2011.

[GN14] Ashish Goel and Hamid Nazerzadeh. Price-based protocols for fair resource allocation: Convergence time analysis and extension to leontief utilities. *ACM Transactions on Algorithms (TALG)*, 10(2), 2014.

[GRW+15] Tuan Nguyen Gia, Mingzhe Jiang1 Amir-Mohammad Rahmani, Tomi Westerlund, Pasi Liljeberg, and Hannu Tenhunen. Fog computing in healthcare Internet-of-Things: A case study on ECG feature extraction. In *Int'l Conf. on Computer and Information Technology (CIT)*, pages 356–363, 2015.

[GZY+13] Bin Guo, Daqing Zhang, Zhiwen Yu, Yunji Liang, Zhu Wang, and Xingshe Zhou. From the internet of things to embedded intelligence. *World Wide Web*, 16(4):399–420, 2013.

[Han13] Matthew W. Hann. Ultra low power, 18 bit precision ecg data acquisition system, June 2013.

[HBB+11] Jörg Henkel, Lars Bauer, Joachim Becker, Oliver Bringmann, Uwe Brinkschulte, Samarjit Chakraborty, Michael Engel, Rolf Ernst, Hermann Härtig, Lars Hedrich, et al. Design and architectures for dependable embedded systems. In *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*, pages 69–78. IEEE, 2011.

[HHB+12] Jörg Henkel, Andreas Herkersdorf, Lars Bauer, Thomas Wild, Michael Hübner, Ravi Kumar Pujari, Artjom Grudnitsky, Jan Heisswolf, Aurang Zaib, Benjamin Vogel, et al. Invasive manycore architectures. In *Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 193–200. IEEE, 2012.

[HMCP04] Wendi B Heinzelman, Amy L Murphy, Hervaldo S Carvalho, and Mark A Perillo. Middleware to support sensor network applications. *IEEE Network*, 18(1):6–14, 2004.

[HO13] Jie Han and Michael Orshansky. Approximate computing: An emerging paradigm for energy-efficient design. In *European Test Symposium (ETS)*, pages 1–6, 2013.

[HPA+17] Jörg Henkel, Santiago Pagani, Hussam Amrouch, Lars Bauer, and Farzad Samie. Ultra-low power and dependability for iot devices (invited paper for iot technologies). In *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 954–959, 2017.

[HPS+15] Moeen Hassanalieragh, Alex Page, Tolga Soyata, Gaurav Sharma, Mehmet Aktas, Gonzalo Mateos, Burak Kantarci, and Silvana Andreescu. Health monitoring and management using Internet-of-Things (IoT) sensing with cloud-based processing: Opportunities and challenges. In *IEEE International Conference on Services Computing (SCC)*, pages 285–292, 2015.

[HWN12] Dong Huang, Ping Wang, and Dusit Niyato. A dynamic offloading algorithm for mobile computing. *IEEE Transactions on Wireless Communications*, 11(6):1991–1995, 2012.

[Ias03] Leon D Iasemidis. Epileptic seizure prediction and control. *IEEE Transactions on Biomedical Engineering*, 50(5):549–558, 2003.

[int] Intel IoT Gateway. Online: `http://www.intel.de/content/dam/www/public/us/en/documents/product-briefs/gateway-solutions-iot-brief.pdf`. Visited on April 3, 2016.

[KC11] Joyce Kwong and Anantha P Chandrakasan. An energy-efficient biomedical signal processing platform. *IEEE Journal of Solid-State Circuits*, 46(7):1742–1753, 2011.

[Kel14] Dave Kelf. IoT: A Return to Our Favorite EDA Requirements. Online: `www.eetimes.com/author.asp?doc_id=1322009`, 2014. Published: April 18, 2014, visited on December 14, 2015.

[Kha15] Minhaj Ahmad Khan. A survey of computation offloading strategies for performance improvement of applications running on mobile devices. *Journal of Network and Computer Applications*, 56:28–40, 2015.

[Kim05] Sukun Kim. Wireless sensor networks for structural health monitoring. Msc, University of California at Berkeley, 2005.

[Kim15] Sungwook Kim. Nested game-based computation offloading scheme for mobile cloud IoT systems. *Journal on Wireless Communications and Networking*, 2015(1):1–11, 2015.

[KK15] Andreas Kliem and Odej Kao. The internet of things resource management challenge. In *IEEE International Conference on Data Science and Data Intensive Systems*, pages 483–490, 2015.

[KKVH+14] Hyejung Kim, Sunyoung Kim, Nick Van Helleputte, Antonio Artes, Mario Konijnenburg, Jos Huisken, Chris Van Hoof, and Refet Firat Yazicioglu. A configurable and low-power mixed signal SoC for portable ECG monitoring applications. *IEEE Transactions on Biomedical Circuits and Systems*, 8(2):257–267, 2014.

[KS15] Navroop Kaur and Sandeep K Sood. An energy-efficient architecture for the Internet of Things (IoT). *IEEE Systems Journal*, 2015.

[KYM+10] Hyejung Kim, Refet Firat Yazicioglu, Patrick Merken, Chris Van Hoof, and Hoi-Jun Yoo. ECG signal compression and classification algorithm with quad level vector for ECG holter system. *IEEE Transactions on Information Technology in Biomedicine*, 14(1):93–100, 2010.

[LCM10] Gregorio López, Víctor Custodio, and José Ignacio Moreno. Lobin: E-textile and wireless-sensor-network-based platform for healthcare monitoring in future hospital environments. *IEEE Transactions on Information Technology in Biomedicine*, 14(6):1446–1458, 2010.

[LDXW13] Shancang Li, Li Da Xu, and Xinheng Wang. Compressed sensing signal and data acquisition in wireless sensor networks and internet of things. *IEEE Transactions on Industrial Informatics*, 9(4):2177–2186, 2013.

[LE02] Brian Litt and Javier Echauz. Prediction of epileptic seizures. *The Lancet Neurology*, 1(1):22–30, 2002.

[LEMC12] Ilias Leontiadis, Christos Efstratiou, Cecilia Mascolo, and Jon Crowcroft. Senshare: transforming sensor networks into multi-application sensing infrastructures. In *Wireless Sensor Networks*, 2012.

[LL15] In Lee and Kyoochun Lee. The Internet of Things (IoT): Applications, investments, and challenges for enterprises. *Business Horizons*, 2015.

[LL16] . Link Labs. A comprehensive look at low power, wide area networks. Online: `http://info.link-labs.com/lpwan`, 2016.

[LWX01] Zhiyuan Li, Cheng Wang, and Rong Xu. Computation offloading to save energy on handheld devices: a partition scheme. In *Proceedings of the international conference on Compilers, architecture, and synthesis for embedded systems*, pages 238–246, 2001.

[LYZ+17] Jie Lin, Wei Yu, Nan Zhang, Xinyu Yang, Hanlin Zhang, and Wei Zhao. A survey on internet of things: architecture, enabling technologies, security and privacy, and applications. *IEEE Internet of Things Journal*, 2017.

[LZP+12] Xin Liu, Yuanjin Zheng, Myint Wai Phyu, FN Endru, V Navaneethan, and Bin Zhao. An ultra-low power ECG acquisition and monitoring ASIC system for WBAN applications. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2(1):60–70, 2012.

[MAM13] Roshan Joy Martis, U Rajendra Acharya, and Lim Choo Min. ECG beat classification using PCA, LDA, ICA and discrete wavelet transform. *Biomedical Signal Processing and Control*, 8(5):437–448, 2013.

[MBF17] Bojan Milosevic, Simone Benatti, and Elisabetta Farella. Design challenges for wearable emg applications. In *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1432–1437. IEEE, 2017.

[MD06] M Sabarimalai Manikandan and S Dandapat. Wavelet threshold based ECG compression using USZZQ and Huffman coding of DSM. *Biomedical Signal Processing and Control*, 1(4):261–270, 2006.

[MFL+16] Patricia Milz, Pascal L Faber, Dietrich Lehmann, Thomas Koenig, Kieko Kochi, and Roberto D Pascual-Marqui. The functional significance of eeg microstates associations with modalities of thinking. *Neuroimage*, 125:643–656, 2016.

[MH03] Amy Murphy and Wendi Heinzelman. Milan: Middleware linking applications and networks. Technical report, University of Rochester, Tech. Rep. TR-795, Jan. 2003.

[MJS97] Martin Moser, Dusan P Jokanovic, and Norio Shiratori. An algorithm for the multidimensional multiple-choice knapsack problem. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 80(3):582–589, 1997.

[MM01] George B Moody and Roger G Mark. The impact of the MIT-BIH arrhythmia database. *IEEE Engineering in Medicine and Biology Magazine*, 20(3):45–50, 2001.

[MM+16] Mahmoud Shuker Mahmoud, Auday AH Mohamad, et al. A study of efficient power consumption wireless communication techniques/modules for internet of things (IoT) applications. *Advances in Internet of Things*, 2016.

[MNMKSK+15] A. Mohsen Nia, M. Mozaffari-Kermani, S. Sur-Kolay, A. Raghunathan, and N.K. Jha. Energy-efficient long-term continuous personal health monitoring. *IEEE Transactions on Multi-Scale Computing Systems*, 1(2):85–98, 2015.

[MPV11] Luca Mainetti, Luigi Patrono, and Antonio Vilei. Evolution of wireless sensor networks towards the internet of things: A survey. In *International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, pages 1–6, 2011.

[MPW07] Qicheng Ma, David C Parkes, and Matthew D Welsh. A utility-based approach to bandwidth allocation and link scheduling in wireless networks. In *International Workshop on Agent Technology for Sensor Networks (ATSN-07)*, 2007.

[MSDPC12] Daniele Miorandi, Sabrina Sicari, Francesco De Pellegrini, and Imrich Chlamtac. Internet of things: Vision, applications and research challenges. *Ad Hoc Networks*, 10(7):1497–1516, 2012.

[Mula] Geoff Mulligan. Keynote speech at wf-iot 2016.

[Mulb] Multitech. Introduction to LoRa. Online: `http://www.multitech.net/developer/software/lora/introduction-to-lora/`. Visited on June 23, 2016.

[MZ16] Yuyi Mao and Jun Zhang. Dynamic computation offloading for mobile-edge computing with energy harvesting devices. *IEEE Journal of Solid-State Circuits*, 51(3):712–723, 2016.

[MZL+15] Kaisheng Ma, Yang Zheng, Shuangchen Li, Karthik Swaminathan, Xueqing Li, Yongpan Liu, Jack Sampson, Yuan Xie, and Vijaykrishnan Narayanan. Architecture exploration for ambient energy harvesting nonvolatile processors. In *HPCA*, pages 526–537, 2015.

[OPP+15] Guido Oddi, Antonio Pietrabissa, Francesco Delli Priscoli, Francisco Facchinei, Laura Palagi, and Andrea Lanna. A QoE-aware dynamic bandwidth allocation algorithm based on game theory. In *Mediterranean Conference on Control and Automation (MED)*, pages 979–985, 2015.

[PBCA15] Filippo Palumbo, Paolo Barsocchi, Stefano Chessa, and Juan Carlos Augusto. A stigmergic approach to indoor localization using bluetooth low energy beacons. In *Advanced Video and Signal Based Surveillance (AVSS)*, pages 1–6, 2015.

[PBM13] Amit Pande, Eilwoo Baik, and Prasant Mohapatra. Efficient health data compression on mobile devices. In *ACM MobiHoc workshop on Pervasive wireless healthcare*, pages 25–30, 2013.

[Pis95] David Pisinger. *Algorithms for knapsack problems*. PhD thesis, University of Copenhagen, 1995.

[PJZ+14] Charith Perera, Prem Prakash Jayaraman, Arkady Zaslavsky, Dimitrios Georgakopoulos, and Peter Christen. Mosden: An Internet of Things middleware for resource constrained mobile devices. In *HICSS*, pages 1053–1062, 2014.

[PPB+12] Shyamal Patel, Hyung Park, Paolo Bonato, Leighton Chan, and Mary Rodgers. A review of wearable sensors and systems with application in rehabilitation. *Journal of neuroengineering and rehabilitation*, 9(1):21, 2012.

[PZCG14] Charith Perera, Arkady Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. Context aware computing for the Internet of Things: A survey. *IEEE Communications Surveys & Tutorials*, 16(1):414–454, 2014.

[Rap16] Chris Raphael. Why edge computing is crucial for the IoT. Online: `http://www.rtinsights.com/why-edge-computing-and-analytics-is-crucial-for-the-iot/`, 2016. Published on November 12, 2015, visited on July 6, 2016.

[RT09] Robert Rieger and John T Taylor. An adaptive sampling system for sensor nodes in body area networks. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 17(2):183–189, 2009.

[S+14] John Stankovic et al. Research directions for the Internet of Things. *IEEE Internet of Things Journal*, 1(1):3–9, 2014.

[SBH15] Farzad Samie, Lars Bauer, and Jörg Henkel. An approximate compressor for wearable biomedical healthcare monitoring systems. In *CODES+ISSS*, pages 133–142, 2015.

[SBH16] Farzad Samie, Lars Bauer, and Jörg Henkel. IoT Technologies for Embedded Computing: A Survey. In *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. IEEE Press, 2016.

[SBHH15] Farzad Samie, Lars Bauer, Chih-Ming Hsieh, and Jörg Henkel. Online binding of applications to multiple clock domains in shared fpga-based systems. In *DATE*, pages 25–30, 2015.

[Sch17] Patrick Schaumont. Security in the internet of things: A challenge of scale. In *Design, Automation & Test in Europe Conference (DATE)*, pages 674–679, 2017.

[SCL+05] Victor Shnayder, Bor-rong Chen, Konrad Lorincz, Thaddeus RF Fulford Jones, and Matt Welsh. Sensor networks for medical care. In *SenSys*, volume 5, pages 314–314, 2005.

[SCZ+16] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 2016.

[SEKC15] Ola Salman, Imad Elhajj, Ayman Kayssi, and Ali Chehab. Edge computing enabling the Internet of Things. In *IEEE World Forum on Internet of Things (WF-IoT)*, pages 603–608, 2015.

[Sha] Patricia O. Shafer. About epilepsy: The basics. Online: `http://www.epilepsy.com/learn/about-epilepsy-basics`. Published on January 2014, visited on April 14, 2017.

[SHLL14] Shun-Ren Siao, Chih-Cheng Hsu, Mark Po-Hung Lin, and Shuenn-Yuh Lee. A novel approach for ECG data compression in healthcare monitoring system. In *International Symposium on Bioelectronics and Bioinformatics (ISBB)*, pages 1–4, 2014.

[SHNN12] Matti Siekkinen, Markus Hiienkari, Jukka K Nurminen, and Johanna Nieminen. How low energy is bluetooth low energy? comparative measurements with ZigBee/802.15.4. In *IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, pages 232–237, 2012.

[sma] Smartcitizen. Online: `https://smartcitizen.me`.

[Smi11] Phil Smith. Comparing low-power wireless technologies. Tech Zone, Digikey Online Magazine, Digi-Key Corporation, 2011.

[SML+15] Z. Sheng, C. Mahapatra, V. Leung, M. Chen, and P. Sahu. Energy efficient cooperative computing in mobile wireless sensor networks. *IEEE Transactions on Cloud Computing*, 2015.

[Smo00] Brent A Smolinski. Approximating the 0-1 Multiple Knapsack Problem with Agent Decomposition and Market Negotiation. In *Intelligent Problem Solving. Methodologies and Approaches*, pages 296–306. Springer, 2000.

[SPBH18] Farzad Samie, Sebastian Paul, Lars Bauer, and Jörg Henkel. Highly Efficient and Accurate Seizure Prediction on Constrained IoT Systems. In *IEEE/ACM Design, Automation and Test in Europe Conference (DATE'18)*. IEEE Press, 2018.

[SRVMAA15] Grégoire Surrel, Francisco Javier Rincon Vallejos, Srinivasan Murali, and David Atienza Alonso. Real-time probabilistic heart beat classification and correction for embedded systems. In *Computing in Cardiology*, 2015.

[SSB15] Stefania Sardellitti, Gesualdo Scutari, and Sergio Barbarossa. Joint optimization of radio and computational resources for multicell mobile-edge computing. *IEEE Transactions on Signal and Information Processing over Networks*, 1(2):89–103, 2015.

[SSGS15] Giorgos Siantikos, Dimitris Sgouropoulos, Theodoros Giannakopoulos, and Evaggelos Spyrou. Fusing multiple audio sensors for acoustic event detection. In *ISPA*, 2015.

[STB+17a] Farzad Samie, Vasileios Tsoutsouras, Lars Bauer, Sotirios Xydis, Dimitrios Soudris, and Jörg Henkel. Distributed Trade-based Edge Device Management in Multi-gateway IoT. *ACM Transactions on Cyber-Physical Systems (TCPS), Special Issue on Internet of Things (IoT) (accepted to appear)*, 2017.

[STB+17b] Farzad Samie, Vasileios Tsoutsouras, Lars Bauer, Sotirios Xydis, Dimitrios Soudris, and Jörg Henkel. Fast Operation Mode Selection for Highly Efficient IoT Edge Devices. *ACM Transactions on Cyber-Physical Systems (TCPS), (under review)*, 2017.

[STX+16a] Farzad Samie, Vasileios Tsoutsouras, Sotirios Xydis, Lars Bauer, Dimitrios Soudris, and Jörg Henkel. Computation Offloading and Resource Allocation for Low-power IoT Edge Devices. In *IEEE 3rd World Forum on Internet of Things (WF-IoT)*, 2016.

[STX+16b] Farzad Samie, Vasileios Tsoutsouras, Sotirios Xydis, Lars Bauer, Dimitrios Soudris, and Jörg Henkel. Distributed QoS Management for Internet of Things under Resource Constraints. In *International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2016.

[TBA+14] Ralf Tönjes, P Barnaghi, MI Ali, A Mileo, M Hauswirth, F Ganz, S Ganea, B Kjærgaard, D Kuemper, Septimiu Nechifor, et al. Real time iot stream processing and large-scale data analytics for smart city applications. In *European Conference on Networks and Communications (EUCNC)*, 2014.

[TCD+14] Kevin Townsend, Carles Cufí, Robert Davidson, et al. *Getting started with Bluetooth low energy: Tools and techniques for low-power networking*. " O'Reilly Media, Inc.", 2014.

[TSE+15] Andreas Tobola, Franz J Streit, Chris Espig, Oliver Korpok, Christian Sauter, Nadine Lang, Björn Schmitz, Christian Hofmann, Matthias Struck, Christian Weigand, et al. Sampling rate impact on energy consumption of biomedical signal processing systems. In *International Conference on Wearable and Implantable Body Sensor Networks (BSN)*, pages 1–6, 2015.

[TZGX15] Liansheng Tan, Zhongxun Zhu, Fei Ge, and Naixue Xiong. Utility maximization resource allocation in wireless networks: Methods and algorithms. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 45(7):1018–1034, 2015.

[UAY+15] M Ueki, KT Akeuchi, T Yamamoto, A Tanabe, N Ikarashi, M Saitoh, T Nagumo, H Sunamura, M Narihiro, K Uejima, et al. Low-power embedded ReRAM technology for IoT applications. In *Symposium on VLSI Circuits*, pages 108–109, 2015.

[vid13] Understanding bitrates in video files. Online: `help.encoding.com/knowledge-base/article/understanding-bitrates-in-video-files`, 2013.

[VQA14] Shahin Vakilinia, Dongyu Qiu, and Mustafa Mehmet Ali. Optimal multi-dimensional dynamic resource allocation in mobile cloud computing. *Journal on Wireless Communications and Networking*, 2014(1):201, 2014.

[WJSX15] Aosen Wang, Zhanpeng Jin, Chen Song, and Wenyao Xu. Adaptive compressed sensing architecture in wireless brain-computer interface. In *Proceedings of the 52nd Annual Design Automation Conference*, page 173, 2015.

[WLS+15] Zhibo Wang, Yongpan Liu, Yinan Sun, Yang Li, Daming Zhang, and Huazhong Yang. An energy-efficient heterogeneous dual-core processor for Internet of Things. In *ISCAS*, pages 2301–2304, 2015.

[WPW30] Louis Wolff, John Parkinson, and Paul D White. Bundle-branch block with short PR interval in healthy young people prone to paroxysmal tachycardia. *American Heart Journal*, 5(6):685–704, 1930.

[WSJ15] Roy Want, Bill N Schilit, and Scott Jenson. Enabling the internet of things. *IEEE Computer*, 48(1):28–35, 2015.

[WTJ+11] Geng Wu, Shilpa Talwar, Kerstin Johnsson, Nageen Himayat, and Kevin D Johnson. M2M: From mobile to embedded internet. *IEEE Communications Magazine*, 49(4):36–43, 2011.

[XLL07] Changjiu Xian, Yung-Hsiang Lu, and Zhiyuan Li. Adaptive computation offloading for energy conservation on battery-powered systems. In *International Conference on Parallel and Distributed Systems*, volume 2, pages 1–8, 2007.

[XZST07] Feng Xia, Wenhong Zhao, Youxian Sun, and Yu-Chu Tian. Fuzzy logic control based qos management in wireless sensor/actuator networks. *Sensors*, 7(12):3179–3191, 2007.

[YCY+13] Lei Yang, Jiannong Cao, Yin Yuan, Tao Li, Andy Han, and Alvin Chan. A framework for partitioning and execution of data stream applications in mobile cloud computing. *ACM SIGMETRICS Performance Evaluation Review*, 40(4):23–32, 2013.

[YVH11] Hoi-Jun Yoo and Chris Van Hoof. Introduction to Bio-Medical CMOS IC. In *Bio-Medical CMOS ICs*, pages 1–9. Springer, 2011.

[ZKC+15] Thomas Zachariah, Noah Klugman, Bradford Campbell, Joshua Adkins, Neal Jackson, and Prabal Dutta. The internet of things has a gateway problem. In *Mobile Computing Systems and Applications (HotMobile)*, pages 27–32, 2015.

[ZLSX13] Cong Zhu, Xinghu Li, Lingjun Song, and Liming Xiang. Development of a theoretically based thermal model for lithium ion battery pack. *Journal of Power Sources*, 223:155–164, 2013.

[ZMK+15] Ben Zhang, Nitesh Mor, John Kolb, Douglas S Chan, Nikhil Goyal, Ken Lutz, Eric Allman, John Wawrzynek, Edward Lee, and John Kubiatowicz. The cloud is not enough: saving IoT from the cloud. In *USENIX Conf. on Hot Topics in Cloud Computing*, pages 21–21, 2015.

[ZWC+10] Qian Zhu, Ruicong Wang, Qi Chen, Yan Liu, and Weijun Qin. IoT gateway: Bridging wireless sensor networks into internet of things. In *Embedded and Ubiquitous Computing (EUC)*, pages 347–352, 2010.