DUNE-GEPARD Synergy: Framework and Simulator for Tailoring Privacy Assistant Solutions in Internet of Things

Edward Rochester

Abstract

In the last decade, there has been rapid growth in digital economies and ecosystems that rely heavily on data collection and sharing. However, the users have little to no control over the personal data collection, usage and sharing by the devices. Data privacy management has been further complicated by the rapid deployment of Internet of Things (IoT) devices. The sheer number of these devices and their ubiquitous nature go beyond what a person can observe, manually assess and consciously and reasonably make decisions upon in terms of data collection and sharing requests. Therefore, the design of privacy assistants (PAs) that can automatically negotiate privacy decisions on behalf of the users has been the focus of abundant research in recent years. However, the question of scalability and real-world applicability of the proposed PA designs in IoT environments remains to be answered. To address this gap, we develop a component-based framework called DUNE (**D**evice, **U**ser, **N**etwork and **E**nvironment) and implement the proposed framework in a **G**eneral **E**nvironment for **P**rivacy **A**ssistant **R**esearch and **D**esign simulator called GEPARD. To develop DUNE, we identified common structure components used in most PA designs. We then disassembled existing state-of-the-art PA designs following the DUNE framework component structure. The resulting components were implemented in the GEPARD repository, which allowed us to (i) study the behaviour and performance of the components, identify the best-performing components, and (ii) mix and match the proposed components to create new PA designs. We conducted tournament-style experiments, with our results highlighting the importance of different components and their effects on PA performance. Specifically, from the component impact analysis, we observe that network type greatly affects the average user consent percentage and average user power consumption, while negotiation algorithms affect the average user utility that a PA can achieve. As a result, we validate the analytical approach of the DUNE framework to optimize and design novel PA solutions and evaluate them using GEPARD.

CONTENTS

I. Introduction

Pervasive intelligent health care, advanced building management systems, smart city services, intelligent agriculture and other human activities have been affected by or felt the effects of the Internet of Things (IoT) [1]. The IoT interconnects billions of things in our environment and provides them with the ability to collect, store, process and communicate information about themselves, their environment and us [2]. While delivering enormous benefits, the fine-grained data acquisition of the IoT, *e.g.*, in the form of Big Data, gives rise to serious privacy concerns [3]. One of these concerns touches upon informed consent [4]. In general, to legitimize the collection, storage, and processing of any personal information, one requires the Data Subject (DS), *i.e.*, users, informed consent [5]. However, collecting informed consent in IoT environments is challenging due to the ubiquitous nature of IoT and the lack of direct communication/interaction channels between the IoT devices and DS [6]. Additionally, even if such channels were added, the sheer number of IoT devices requiring DS attention would be overwhelming. Consequently, as DSs are continuously asked to consent to the data collection by a multitude of IoT devices, they develop what is known as privacy fatigue [7]. To address these challenges, there has been a growing interest in designing systems capable of estimating DS's privacy preferences, assisting the DS with their privacy decisions and communicating these to the IoT devices. In the literature, these systems are referred to as Privacy Assistant (PA), and they come with varying communication protocols, levels of automation, etc. [6], [8], [9], [10]. However, each of these variations has a significant impact on the IoT system performance and PA user satisfaction. Thus, successfully developing and deploying PA systems provides significant advantages and has critical implications for DS privacy and IoT adoption.

With the ever-increasing proliferation of IoT devices into every aspect of human lives, PA designs have to account for a variety of environments with their respective limitations and requirements. Data collection context, communication network limitations, or DS preferences commonly set forward these limitations and requirements. With the wide variety of currently existing and an ever-growing number of PA designs, however, the proposed individual design of each PA is rarely set comparably against pre-existing PA designs. Additionally, even when the performance of the proposed PA design is tested using varying performance metrics, the inner components of the PA are rarely individually inspected. This makes it challenging to compare PA designs, let alone their underlying components, meaningfully. As a result, there is a lack of

a reliable way to pinpoint the most effective parts of the PA design. Even for a successful PA, it is almost impossible to determine the reasons behind its success or to attempt and provide an incremental improvement over an existing design. As such, we succinctly identify the following problem:

We lack a fundamental approach to incrementally building effective PAs that allows us to compare their effectiveness and understand how their underlying technologies influence their overall performance.

Our main contributions are: (i) PA framework design called DUNE, (ii) simulation environment development called GEPARD and (iii) existing PA solutions analysis.

In the PA framework design, we show common components used in the PA design and develop a component-based PA framework called Device, User, Network, Environment (DUNE). The framework distinguishes the following four components of a PA design:

1) **IoT device**. Given the data collection context, how can IoT devices estimate a DS 's privacy preferences? When should the device communicate data collection policies to the DS?
2) **User model**. What are the DS 's privacy preferences and Privacy Policies (PP) acceptance strategy?
3) **Communication networks**. What network types apply to the chosen communication context? What network technologies best support communication requirements within the communication environment limitations?
4) **Data collection environment**. Describes DS arrival and departure processes, the data collection context and the communication environment.

The advantage of developing the framework and distinguishing between the different components of PA design is that given the performance measure of individual components, we can then assemble, from the identified components, a new PA in a plug-and-play fashion, *e.g.*, by replacing the communication network technology and evaluating what difference it makes in PA's performance. Such an approach enables us to systematically combine individual components and explore the space of possible PA designs, which is critical because of the dependencies between the PA components. A successful PA needs components that work effectively individually and an architecture in which the components can be successfully combined and assembled.

The above motivated the second contribution of our work, *i.e.*, developing a custom-made

flexible simulations environment called General Environment for Privacy Assistant Research and Design (GEPARD). The simulator allows: (i) combining various PA components into a holistic solution, (ii) analyzing and comparing the solution's performance against the alternatives, and (iii) observing the contribution of individual PA components to the PA 's performance. The latter is also used for the final contribution of our work, *i.e.*, analyzing and evaluating different PA designs and their components under various conditions and environments. Currently, we analyze and evaluate the PAs with respect to power consumption, consent collection, and the individual component effect size.

Organization. In Section II we review the related works. Section III provides the necessary background on IoT network technologies. Section IV describes the proposed DUNE framework, while Section V presents the high-level architecture and implementation details of the proposed GEPARD simulator. Section VI presents the experimental evaluation of different PA designs using GEPARD. Finally, Sections VII and VIII discuss limitations and future directions of our work and conclude the paper, respectively.

II. Related Work

A. Automated Negotiations

Negotiations lie at the core of many activities in human society, *e.g.*, forming alliances, reaching trading agreements, and conflict resolution [11]. Consequently, negotiations are an important topic of research in various fields, *e.g.*, economics [12], artificial intelligence [13], game theory [14], and social psychology [15]. The last several decades have brought a growing interest to the automation of negotiations [16], [17]. This rising interest has been fuelled by the promise of automated agents that can negotiate on behalf of users and come to better outcomes than human negotiators [18], [19]. Within the field of data privacy, the negotiation agents promise to reduce efforts required from the DS during the privacy negotiation and make negotiation more manageable and comprehensible for an average user [20].

Automated negotiation research deals with two main topics [11]: (i) negotiation protocol and (ii) agent's decision-making model. The negotiation protocols are the rules governing the way negotiation takes place. This covers the number of participants, valid actions, and what valid negotiation states are. The agent's decision-making model reflects the reasoning modules and strategies the negotiating agent employs to make decisions. Within the field of data privacy, Baarslag *et al*. [21] presented an automated negotiation agent that relies on the learned DS privacy preferences to negotiate the data sharing permissions. The proposed agent learns the DS's privacy preferences by posing three privacy-related questions and, based on the replies, assigns the DS a privacy type. The authors evaluated the proposed agent in a lab study, showing that the agreements negotiated by agents are more accurate than the baseline (built upon Westin's work) and result in agreements better aligned with the DS's privacy preferences. However, the authors note that the agent's deployment did not result in less effort from the DS, since the DS's require time to build trust and concede more autonomy to the agent.

Baarslag *et al*. was extended by Filipczuk *et al*. [22], where the authors presented an agent-based negotiation framework that can operate over partial and complete offers. The authors also introduce a variant of DS privacy preference learning approach that can learn personalized privacy preferences of individual DSs and determine DS privacy type. Interestingly, the evaluation results show that the DSs are inclined to share personal data 2.5 times more often when they can negotiate the conditions for the data sharing. Moreover, the authors show that their approach to negotiations is less mentally demanding from the DSs than the currently used take-it-or-leave-it

approach and allows the DSs to align the environment to their privacy preferences.

Similarly, Mohammad *et al*. [23] propose extending the optimal privacy preference elicitation algorithms to practical usages through only partial uncertainty reduction in the utility functions. Such extension was achieved but with limitations, *e.g*., the algorithms only apply to cases with countable outcome spaces. Nonetheless, the proposed methods outperformed the chosen state-of-the-art algorithms in terms of higher agreement probability and lower elicitation costs.

Alternatively, Kökciyan *et al*. [24] propose an agent-based PA model that automatically determines the level of trust for a context. The designed PA model determines when it can decide on behalf of the DS or when it should abstain and delegate the decision to the DS. The model makes the decision based on the trust for a context, which is determined based on the DS's past positive and negative privacy experiences in that context. Such an approach attempts to address the problem of PA's limited experiences within the context. The proposed model was evaluated using an anonymized IoT dataset, with the results comparable to existing works' privacy decision accuracy.

Finally, Sanchez *et al*. [25] propose to utilize Semantic Web technology to manage DS's privacy preferences and negotiate data sharing permissions. In the proposed approach, the Semantic Web enables the communication between the Personal Data Manager and 3^{rd} party. Negotiations only occur when there has been an indication that privacy preferences are negotiable. The proposed solution was evaluated using an actual running application. The results show that existing permission settings on the DS phones did not align with their privacy preferences. Additionally, the results highlight the benefits of the proposed privacy preference settings and modelling compared to existing settings in their respective platforms, *e.g*. Android. However, the authors recognized some limitations of the proposed work. For example, their work does not account for changes in DS's privacy preferences over time, and there is a lack of control over the data once it has been disclosed. These, however, are common restrictions of other existing works.

B. Privacy Assistants

The development and proliferation of IoT technologies raise various privacy concerns, particularly regarding DS consent. The informed consent of a DS is a fundamental necessity to legitimize collecting, storing, and processing personal information. The existing notice and consent mechanisms, *e.g*., wall notice signs and End User License Agreements, have many

limitations and are neither easily nor effectively transferred to IoT. Additionally, with the increase in IoT device numbers, existing notice and consent collection approaches would overwhelm the DS. As a result, several works have proposed the use of PAs. A PA can be defined as an agent that interacts on behalf of the DS and, in some manner, communicates the DS 's privacy preferences to an environment, *e.g*., IoT devices.

Langheinrich [26] presented early privacy awareness Systems (pawS) for ubiquitous computing environments. The system's design consisted of the four core concepts: (i) notice, (ii) choice and consent, (iii) proximity and locality, and (iv) access and recourse. Although decades before the proliferation of IoT, the design ideas behind pawS are still used in various IoT PA architectures. In pawS, upon entering the ubicomp environment, a privacy beacon, using Wireless Sensor Network (WSN), announces the data collection. The PA, carried by the DS, delegates the information to the privacy proxy, residing in the Cloud, which contacts the corresponding service proxies and inquires about their PP. Following the inquiry, the user proxy decides whether to consent or decline the usage of the service. One can notice that while alleviating the direct communication energy penalty, the timeliness of proxy communication does not meet the real-time consent demands.

More recently, Neisse *et al*. [27] described a framework for informed consent in IoT, which followed the design presented by Langheinrich [26]. In particular, the framework provides a privacy-friendly way to associate personal data usage control policies with DSs and service providers. The framework uses pseudonyms to provide DS registration pseudonymity and builds upon existing technologies to provide informed consent PP specification and evaluation. However, the PP specification framework does not consider data usage purpose and follows the access control specification approach rather than PP specification.

Cha *et al*. [28] addressed the consent issues when data subjects access nearby IoT devices from their smartphones via the iBeacon functionality of the Bluetooth Low Power (BLE). Cha *et al*. proposed a privacy preference expression framework named PrivacyBat. DS privacy preferences are communicated through the Privacy Preference Expression Generic ATTribute Profile (GATT) services, which every personal data collecting BLE device must implement. Compared to Langheinrich [26] and Neisse *et al*. [27], the presented work addressed a specific IoT scenario and, as such, does not directly apply to general IoT deployment scenarios.

Das *et al*. [8] developed and deployed PAs supporting IoT infrastructure, closely following the ideas outlined by Langheinrich [26]. The proposed infrastructure consists of three major

components:

1) The IoT Resource Registry (IRR): allows IoT device owners to publish IoT resource descriptions and data collection practices.
2) The Personalized Privacy Assistant (PPA): is a mobile app that runs on the data subject's smartphone and interacts with IRR to discover nearby IoT devices.
3) The Policy Enforcement Points (PEPs): are responsible for enforcing DS 's privacy settings.

The authors developed three mobile applications and deployed them at two university campuses. The authors mentioned that IRR can be used for advertising purposes to motivate IoT device owners to register with the IRR. However, such communication overhead can lead to DS fatigue from the advertisement overflow from, potentially, hundreds of surrounding IoT devices.

In more recent work, Morel *et al*. [6] and Cunche *et al*. [10] presented high-level requirements for consent in IoT. Based on the requirements, the authors presented a set of technical requirements for implementing a consent framework in IoT that is protective of data subjects and controllers. The requirements were defined using formal semantics, ensuring clarity and proof capabilities. Following this, Morel *et al*. outlined technical options to implement the requirements. The underlying assumption for the presented technical options is a software tool located on the DS's device. Such design follows the same route as presented by the above-mentioned work [8], [26], [28]. To show the real-world applicability of the presented framework, the authors implemented a BLE-based prototype and showed that the prototype meets the requirements presented by the authors. However, similar to the previous works, there was no analysis of the non-functional elements of the infrastructure.

Finally, Alanezi *et al*. [29], [30] presented PA design that addresses privacy concerns of both individual DSs and group of DSs. We see this work as the closest to ours since the authors provided prototypes of the proposed solutions and showed the practicality of the proposed PA design. Nevertheless, the authors did not compare the proposed PA design against other existing solutions.

Consequently, we can observe an evident lack of works that consider both privacy negotiations and non-functional elements of privacy negotiation implementation, *e.g*., power consumption, sensing/communication range disparities, and scalability, when developing and evaluating newly proposed PA designs.

C. IoT Capability Evaluation

Provided the above discussed PA design variety, it also is essential to evaluate the capability of the IoT infrastructure to support the discussed negotiations and PA designs. Such evaluation helps to analyze the IoT infrastructure's readiness to accommodate the PA designs and negotiations. Factors such as energy consumption, network capacity, effective communication range, and others play a pivotal role in ensuring the robust and easy integration of PA within the IoT ecosystem, emphasizing the importance of thoroughly assessing the IoT environments.

In IoT there are many complicated scenarios and numerous network protocols utilized that possess unique and distinctive features that need to be implemented in a simulator. Consequently, many existing network simulators have been adapted to suit the simulation process of the IoT environments, *e.g.*, OMNeT++ [31] and NS family of simulators [32].

Most recently, Sharma *et al*. [33] provided a comprehensive overview of existing network simulators. The authors compared the simulators' advantages and weaknesses, as well as presented a case study highlighting the benefits of specific simulation solutions, *i.e*. MATLAB/Simulink. Although survey work focuses on the 2008-2020 time period, it is still highly applicable, and the majority of recent work still make use of one of the mentioned simulation tools [34], *e.g*., MATLAB [35], OMNeT++ [36], and NS-3 [37].

Overall, with regard to simulators that facilitate the actual design of PAs, to our knowledge there are no systems that are close to our line of suggested work. Most simulators and systems that could be related to the main focus of our work are either focused on IoT as an application and service provider or strictly network components of the system. GEPARD, however, offers an opportunity to quickly evaluate PA designs with respect to both privacy mechanisms and IoT infrastructure.

III. Background

A. Network Technologies

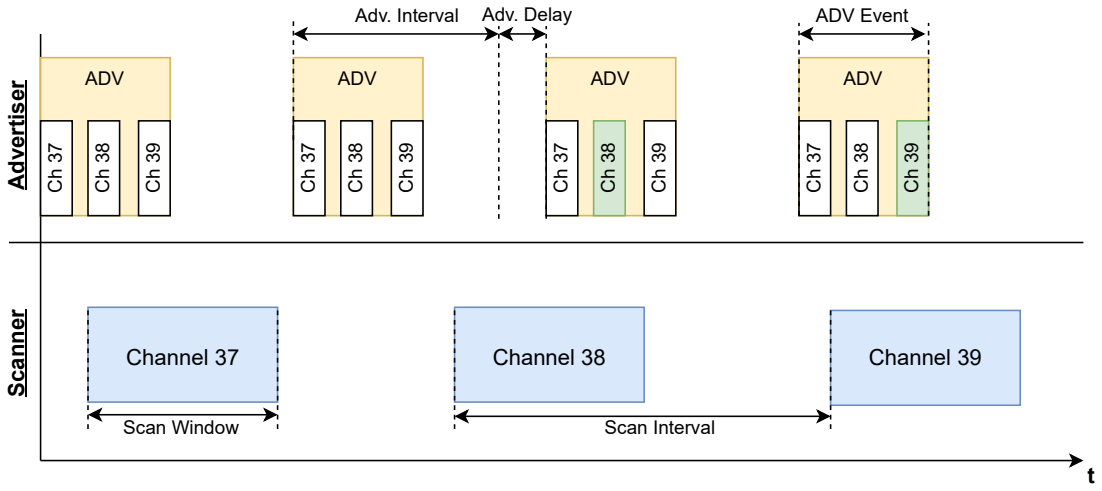
1) Bluetooth Low Energy (BLE): In this section, we summarize the main features of the BLE protocol that apply to our proposed simulator. For all the details of the BLE protocol, see BLE specifications [38]. BLE operates over the 2.4 GHz Industrial, Scientific, and Medical (ISM) spectrum and employs frequency hopping over 37 channels for (bidirectional) communication and 3 for (unidirectional) advertising with a bitrate of 1 Mbps. At a high-level, two BLE-enabled devices attempting to establish a connection will follow the following set of steps:

1) To discover each other, one of the devices periodically sends out Advertisement (ADV) packets, while the second device scans for these packets.
2) As soon as the scanning device receives ADV packet, it goes into *initiating* state to establish a connection with the advertising device. In this state, so-called handshake packets are exchanged between the devices.
3) Upon completion, both devices transition into *connected* state, in which the used-to-be advertiser devices now become the slave and the former scanner device becomes what is called the master device. In BLE connection, the master controls the timing of the communication, so both devices are aware of the time-slicing for data exchange.

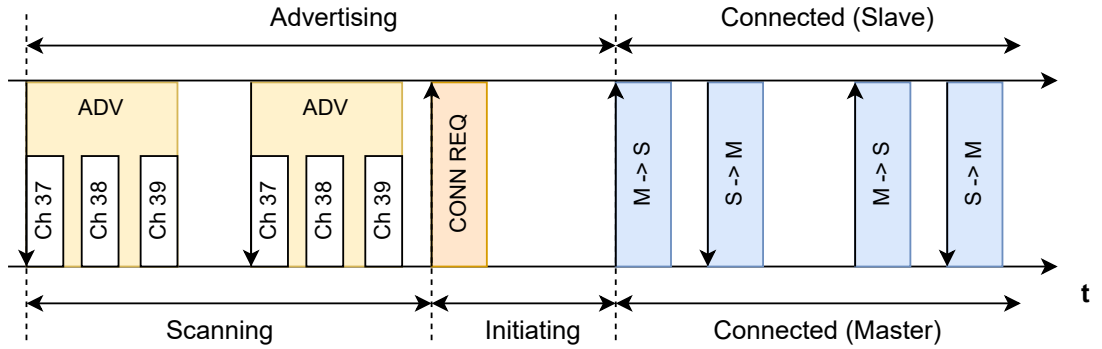
For the purpose of our simulator and to align the BLE protocol model with existing PA designs, we distinguish between the following three BLE modes: (i) scanning/advertising mode (to model BLE for user smartphone and IoT device, respectively), (ii) connection establishment mode, and (iii) connected mode. Without the loss of generality, forBLE modelling and simulator, we consider BLE networks with only two participating devices.

Note: We are aware that the simplification of the BLE model introduces some of the following limitations to our results: (i) in case the master device has a connection to multiple slave devices, then there is a potential for connection events from both slaves to overlap in time, leading to one of them being underserved, resulting in a higher than predicted energy consumption, and (ii) during the discovery phase with multiple devices, there is a probability for packet collision, which also results in increased energy consumption and longer connection setup times than predicted by the current version of the simulator.

Scanning/Advertising Mode. Scanning/Advertising mode, also known as non-connected communication mode, is mainly used in BLE for neighbor discovery or exchange of small amounts



(a) Advertising and scanning in BLE.



(b) BLE connection establishment packet flow.

Fig. 1: Packet flow in BLE protocol.

of data between the two devices without prior synchronization. As was discussed above, in this mode, one device acts as a scanner while the other is an advertiser. Figure 1(a) shows a simplified diagram of advertising and scanning operation in BLE. During this time, the advertiser periodically sends out ADV packets. The ADV packets are sent out on at least one of the three dedicated advertising channels, *i.e.*, channel 37, 38, or 39. The exact subset of the three channels to be used for advertising is determined by the application. Independently of the advertiser, the scanner device periodically, specified by the *scan interval*, listens to one channel at a time for a specified *scan window* time, and then hops on to the next advertising channel for the next period. When the scanner receives an ADV packet, it may respond to the advertiser within the same advertisement event, defined as a group of consecutive ADV packets.

Note: The BLE specifications require the scanner to use all three advertising channels. The

advertiser will expect a response on the same advertising channel for *interframe-space* interval or 150 μs after the end of the ADV packet.

Connection Establishment Mode. After the scanner has detected the advertiser, the scanner may send the connection request packet. Figure 1(b) presents the packet flow during the BLE connection establishment. The connection establishment packet will contain such information as the timing of the connection establishment procedure, *i.e.*, *transmit windows offset* and *transmit window*. By sending the connection establishment packet, the scanner enters the initiating state and, as such, is called an *initiator* of the connection. 1.25 ms $+$ *transmit windows offset* time units after the connection request packet, the transmit window starts, during which the scanner/initiator may schedule the first regular connection event. The 1.25 ms constant is defined by the BLE specifications [38]. The advertiser has to listen during the entirety of the transmission window until it successfully receives the first connection event. Afterwards, the BLE devices are connected, and the data exchange can begin.

Connected Mode. During the connection establishment phase, both devices will agree on a connection interval that determines the interval or period between the connection events. The connection interval must range between 7.5 ms and 4 s. To synchronize the devices, an anchor point t is used that both devices must be awake at, and during which the scanner, now master, will send a packet. The advertiser, now a slave, will acknowledge the packet by sending a response packet 150 μs later, where the response packet can either be an empty response, a payload packet, or a packet performing control functions. If more data is left to transmit, more pairs of packets will be exchanged in the same manner, separated by *interframe-space* interval. The flow of these packets can be seen in Figure 1(b).

2) LoRa: LoRa is an Low-Power Local Area Network (LPWAN) technology developed by Semtech Corporation[1] that offers low power long range communication. LoRa permits to trade-off achievable data rates for communication range, enabling it to be used for large-scale deployment of low power IoT devices over vast geographic areas [39]. LoRa networks operate in unlicensed ISM frequency band, which for North America is the frequency band $902 - 928$ MHz with a center frequency of 915 MHz. For this band, LoRa specifications define 64 non-overlapping uplink and 8 downlink channels. Moreover, LoRa networks have three major components that apply to our proposed simulator: (i) Physical (PHY) layer that implements the LoRa modulation,

[1]https://www.semtech.com/

(ii) link layer that implements LoRaWAN specifications, and (iii) the network architecture [40]. A brief overview of each component is provided below. For more details on LoRa see LoRa specification [41].

Network Architecture. LoRa networks rely on a star topology in which end devices directly communicate with a few gateways in a single-hop manner. The gateways are, in turn, connected to a central network server using TCP/IP to communicate with the server. Each end device may adjust its data rate manually or using Adaptive Data Rate (ADR) [40].

PHY Layer. At the physical layer, LoRa implements Chirp Spread Spectrum (CSS) with integrated Forward Error Correction (FEC). To provide flexibility to trade-off communication range and data rate, LoRa modulation supports multiple Spreading Factors (SFs) and Coding Rates (CRs) for end devices. LoRa's SFs are orthogonal, which allows multiple devices to transmit simultaneously with different spreading factors over the same channel. For North America, LoRa specifications define the following four SF values: $\{7, 8, 9, 10\}$. Different SFs result in different transmission rates, thus affecting the time it takes to transmit a message and the devices' time and power consumption. To achieve robustness in the presence of interference, LoRa also implements the following CRs: $\{4/5, 4/6, 4/7, 4/8\}$.

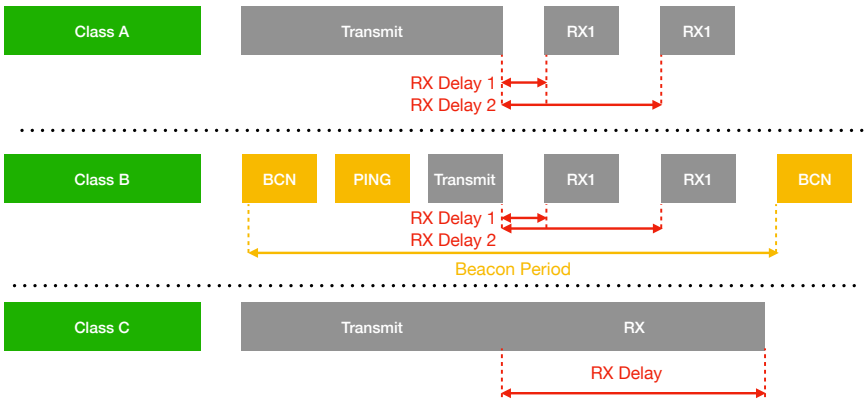


Fig. 2: Packet flow of different LoRa device classes.

Link Layer. LoRa MAC layer, called LoRaWAN, distinguishes between three end device Classes: A, B, and C. Class A devices are optimized for power consumption since the devices have their transceivers in deep sleep mode most of the time, waking up only for infrequent packet

transmissions. As this class of devices is unreachable most of the time, the opportunities for downlink messages are scarce. Consequently, the standard requires the device to open two short receive windows after each uplink message. In addition to the two receive windows defined for Class A devices, Class B devices open extra downlink receive windows at scheduled times, where time is synchronized with beacons transmitted by gateways. Class C devices, on the other hand, continuously keep the receive window open, only closing the window when transmitting. Figure 2 presents the packet flow of the different LoRa device classes. Our work focuses on Class A devices as it is the most power-efficient across all device classes.

Note: The specification requires Class B and C devices to be compatible with the Class A devices.

3) ZigBee: In this section, we summarize the main features of the ZigBee protocol that apply to our proposed simulator. For all the details of the ZigBee protocol, see ZigBee specifications [42]. Institute of Electrical and Electronics Engineers (IEEE) $802.15.4$ standard and ZigBee jointly establish a protocol stack for developing short-range and low-power communications for Wireless Personal Area Networks (WPANs). The IEEE $802.15.4$ standard specifies PHY and MAC layers, while ZigBee alliance specifies network and application layers and security protocols. A typical ZigBee network consists of several device types; a network coordinator which is a device that sets up the network. The coordinator is aware of all the nodes within the network and manages the information about each node and the information being transmitted/received within the network. In addition to the coordinator, there are many battery-powered sensor nodes called 'motes'. Motes are usually kept in a low-power/sleep state, with occasional 'wake up' to sense/transmit the data to reduce the power consumption in IEEE $802.15.4$/ZigBee [43].

Properties of IEEE 802.15.4. IEEE $802.15.4$ allows for $10-100$ m communication range with a maximum transfer rate of 250 kbps. However, the transfer rate can be decreased to 20 kbps to enable a lower power consumption of the motes. Table I presents some basic parameters of the IEEE $802.15.4$/ZigBee standard [44].

TABLE I: Main ZigBee Parameters.

Transmission Range (meters)	$10-100$
Network Size (# of nodes)	$64\,000$
Throughput (kbps)	$20-250$

Network Architecture. IEEE $802.15.4$ supports three types of topologies: star, mesh and tree.

A ZigBee coordinator is responsible for initializing, maintaining, and controlling the network. In the Star topology, a coordinator is surrounded by a group of either end devices or routers. While the topology is simple, if the coordinator stops operating, the entire network is functionless. For tree and mesh networks, devices communicate with each other in a multi-hop fashion. In these topologies, the network backbone is formed by one ZigBee coordinator and multiple ZigBee routers. RDFs join the network as end devices by associating with either the ZigBee coordinator or the routers [45].

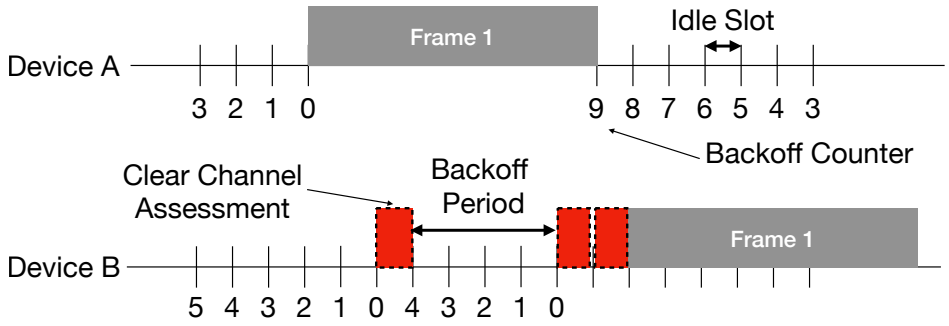


Fig. 3: Packet flow of point-to-point mode in ZigBee.

PHY Layer. The PHY layer defines the physical and electrical characteristics of the network. Its main task is data transmission and reception, which involves modulation and spreading techniques that map bits of information in such a way as to allow them to travel through the air. The PHY layer is responsible for turning the radio transceiver on/off, linking the quality indication for received packets, detecting energy within the current channel, and assessing the channel.

Link Layer. The MAC layer defines how multiple 802.15.4 radios operating in the same area share airwaves. This includes coordinating access to the shared radio link and scheduling and routing data frames. Additionally, network association and disassociation functions are embedded in the MAC layer. These functions support the self-configuration and peer-to-peer communication features of a ZigBee network. IEEE 802.15.4 standard identifies two classes of motes:(i) Full-Function Devices (FFDs) that can operate as network coordinators and can be connected to other motes. (ii) Reduced-Function Devices (RFDs) permits communication as 'end' motes. These motes have no coordinated functionalities to synchronize with the other motes in the network therefore, they connect to an FFD as the coordinator

The MAC layer of IEEE 802.15.4 enables two operational modes:

1) *Point-to-point or Non-beacon-enabled mode*: In this mode, the channel access control is governed by non-slotted Carrier Sense Multiple Access/Collision Avoidance (CSMA/CA). This means that motes sense the radio medium before any transmissions. If the mote senses the channel to be busy, it waits for a random time (*i.e.*, backoff period) before listening to the radio medium again. Otherwise, if the mote finds the channel idle, it transmits the packet and waits for the acknowledgement message. If the acknowledgement is not received in a pre-determined period, the mote retransmits the packet. If two motes transmit their packets simultaneously, a collision happens, which highly affects the performance of CSMA/CA. Note that for our work, we focus on the point-to-point mode. Figure 3 presents this mode's high-level packet flow overview.
2) *Beacon-enabled mode*: In this operational mode, the FFD announces its presence as the coordinator by broadcasting a particular frame called beacon. As the beacon helps synchronize the children's motes, the FFD broadcasts the beacon periodically. The time between two consecutive beacon frames is called Beacon Interval (BI), which varies from 15 ms to 252 s. BI includes two subframes: a Contention Access Period (CAP) and a Contention Free Period (CFP). During the CAP, the communications between the coordinator and children motes are governed by CSMA/CA, while in CFP, time slots are reserved for particular children's motes.

IV. DUNE: Privacy Assistant Architecture Framework

In the last decade, a wide range of PA designs have been introduced in the pursuit of a versatile and effective PA solution (for an overview, see Section II-B). Despite the diversity of the proposed designs, there are common structures to the overall design of the PAs. For example, the majority of the PAs decide whether the privacy policies are acceptable and, if not, whether to produce a counter-offer and what kind of counter-offer to produce. When these decisions are made, the PAs would commonly consider the privacy preferences of the user it represents in the environment and possibly the potential preferences of the PP offerer.

Current work often focus on optimizing the PA design as a whole with limited evaluation and comparison against the existing PA design's performance. Additionally, they commonly overlook the infrastructure required for introducing PAs and privacy negotiation into the IoT environment. An example of an overlooked infrastructure component is the communication network necessary to support the proposed PP negotiations. This oversight hinders a comprehensive understanding of the effectiveness and practicality of the proposed PA designs. Consequently, we propose a shift towards a component-based approach, taking into account the existing pool of works with mutually comparable PA designs. The proposed approach has the following advantages:

1) Given the measure of effectiveness of individual components of a PA design, we can pinpoint the most promising components, which provides insight into the factors behind its success;
2) Focusing on the most effective components helps to systematically search the space of PA designs by recombining them into new and improved designs.

We then amalgamate this approach in a DUNE framework. This section outlines the key components of the DUNE framework and its application to the existing PA designs.

A. DUNE Components

DUNE framework encompasses common components required to design and support PAs within an IoT environment. While focusing on PAs, the framework considers the various components that enable their functioning. It consists of the following four major components:

1) IoT **D**evice: This component represents the IoT device within the environment. It is responsible for sensing the environment, collecting data, and transmitting the collected data to applications. Prior to data collection, the IoT device communicates its data collection policies to the user and seeks their consent. Different PP communication strategies are employed, such as broadcasting PP using BLE.

2) **U**ser: This component represents the DS within the IoT environment. Each user has their privacy preferences and a PP acceptance strategy. These preferences and strategies may be communicated directly to the PA or estimated by the PA based on user behaviour and interaction.
3) **N**etwork: This component represents different network types and technologies the devices use. This includes the communication protocols and infrastructure enabling PP transmission between the IoT device and PAs.
4) **E**nvironment: This component represents the overall context and setting in which the IoT devices and users operate. It encompasses the physical environment, defines user arrival and departure behaviour and speeds, and affects user privacy preferences and other relevant factors that impact privacy considerations.

The elements of the DUNE framework are visualized in Figure 4.

DUNE Framework	
IoT **D**evice	**U**ser
• Represents the IoT device. • Senses the environment, collects data, communicates data collection policies to the user and seeks consent.	• Represents the data subject. • Includes privacy preferences and strategies for privacy policy acceptance.
Network	**E**nvironment
• Represents the network used in the IoT environment. • Includes communication protocols and infrastructure for privacy policy transmission.	• Represents the overall context and user behaviour within the IoT environment. • Influences privacy preferences and considerations.

Fig. 4: The DUNE framework.

B. Employing DUNE Framework

To successfully fit the existing PA designs into the DUNE framework, we should be able to distinguish these components within the proposed designs. To better understand how the DUNE framework operates within the context of the existing works, we decouple two existing PA agent designs into respective DUNE components. Specifically, we chose the PA designs proposed by Alanezi *et al*. [29] and Cunche *et al*. [10], discussed in Section II.

We begin with the PA design proposed by Alanezi *et al*. [29] and further discussed later [30]. To decouple the PA design, we determine the PA components that can be identified and mapped to each of the DUNE framework components:

- IoT **D**evice: In their work, Alanezi *et al*. [29] present a privacy model in which IoT users request access to the IoT data from the IoT owner, which may control several sensors in the environment. *Note:* As mentioned in Section III, in our work, we focus on a single sensing device in the IoT environment.
- **U**ser: No assumptions regarding user privacy preferences are provided by Alanezi *et al*.
- **N**etwork: Within the work, the authors emphasize BLE and Wi-Fi network technologies as the main IoT network solutions. The authors argue that Wi-Fi is more common for edge and cloud deployments, while BLE is more typical for user deployments. However, the authors admit that both BLE and Wi-Fi are viable options in the group context, where the privacy preferences of each individual are combined into a group privacy profile to be negotiated with the IoT owner.
- **E**nvironment: The authors consider multiple deployment environments, *e.g*., shopping mall and musical hall, and they emphasize that for such environments, one would consider group context. The authors also consider that the IoT owner represents the IoT deployment, and it is with the IoT owner that the users negotiate. In group contexts, authors apply the "least misery" approach and focus on the user with the most stringent requirements. In case the agreement is not achieved, the corresponding users will be notified with suggestions from the IoT owner to obtain their approval.

  However, we argue that the group context, as defined by Alanezi *et al*., is not appropriate for such environments as shopping malls since introducing groups on the fly as users come and go would be challenging even for non-IoT based systems, *i.e*., computers. Consequently, we focus on simulating individual negotiations, as presented by Alanezi *et al*.

There are several note-worthy observations to note. First, one can notice that some of the PA designs will either lack the details or any considerations for some of the components defined in the DUNE framework. In those cases, we will either attempt to extract as much information as possible from the provided description or if no description is provided, we borrow the implementations from the other PA designs. Secondly, some PA implementations may contain dependencies between different components. We make the best effort to resolve these

dependencies and separate the design into singular components. Finally, the main limitation of the solution proposed by Alanezi *et al*. is that the authors do not consider *when* to start the negotiations, which is a critical design component and only highlights the downside of non-holistic approaches.

Similarly, we can decouple the PA design proposed by Cunche *et al*. [6]. In their initial work, Morel *et al*. [6] propose a generic framework for consent in IoT. In the later work, the authors propose an implementation of a consent assistant in IoT named CoIoT [10]. In their prototype, the authors presented an Android application representing a PA, or as the authors called it, the Personal Data Custodian (PDC), communicating with IoT infrastructure through BLE. Below, we dissect the proposed implementation using the DUNE framework:

- IoT **D**evice: In their work, Cunche *et al*. [10] modelled IoT device as a Data Collector (DC), which broadcast their PP, taking advantage of the Advertising features of the BLE protocol. A major point to note is that DC PPs are retrieved between one and five seconds after the PDC enters the area.
- **U**ser: Similar to the Alanezi *et al*. [30] work, the authors did not list any user privacy preference assumptions or modelling approaches.
- **N**etwork: The authors used BLE technology as the preferred network technology for the prototype. It is worth mentioning that Morel *et al*. [6] have an in-depth discussion on the effects of the network technology choice on the PP communication types.
- **E**nvironment: While in the original work [6] authors discuss a variety of deployment environments, the prototype [10] was primarily discussed in the context of shopping mall, museum, and music festival areas.

Other PA designs can be decoupled similarly. It is worth noting that while decoupling the proposed PA designs, we can distinguish commonly used component types. For example, it is common to consider shopping malls as the primary PA operational environments and BLE as the main network technology. Consequently, this leads us also to notice potential gaps in PA component designs since different from proposed component types can be considered for improving PA designs, *e.g*., ZigBee.

To facilitate DUNE-based design and evaluation of PAs and showcase the use of the DUNE framework, we created a generalized simulation environment named General Environment for Privacy Assistant Research and Design (GEPARD), presented in Section V.

V. GEPARD

The GEPARD environment was developed with two primary goals in mind: (i) to practically showcase the use of the DUNE framework and (ii) to facilitate the design and evaluation of PAs as a research tool. It provides a flexible and easy-to-use environment for implementing and evaluating PA designs in various scenarios. GEPARD also aids the development of PAs as an analytical toolbox, providing various tools to analyze the PA performance based on the outcome and dynamics in IoT environments. The DUNE framework has been seamlessly integrated into the GEPARD by combining pre-existing components into a repository and templates, allowing users to rapidly create, apply and evaluate newly developed components.

A. *GEPARD Architecture*

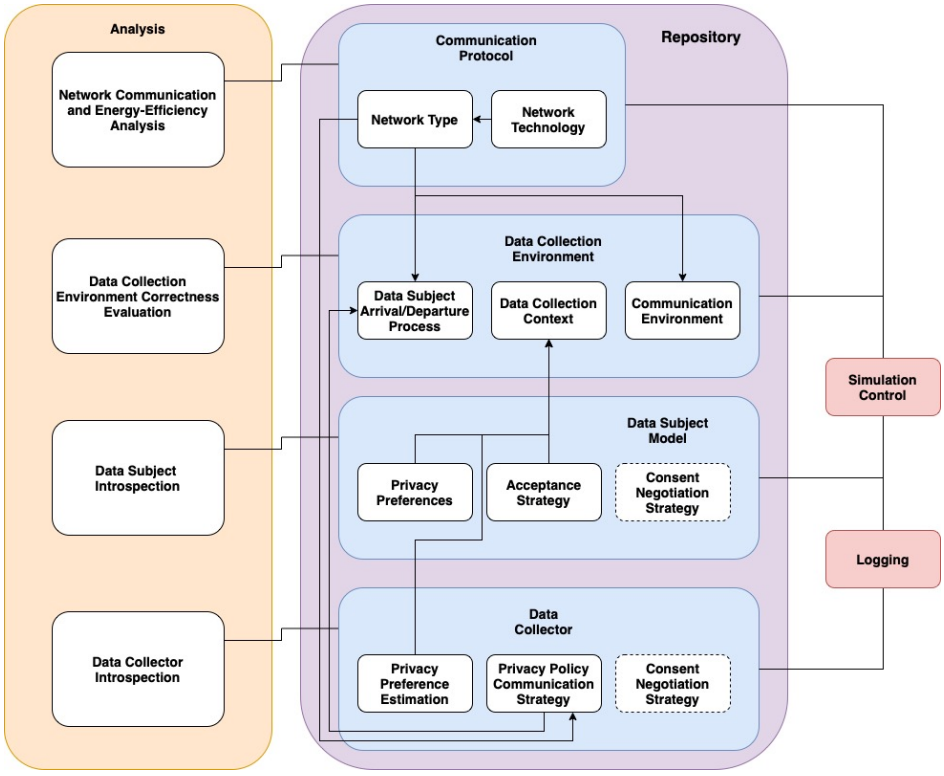


Fig. 5: GEPARD high-level architecture.

The high-level architecture overview of GEPARD is presented in Figure 5. It consists of 4 major modules:

1) Analysis
2) Repository

3) Simulation Controls
4) Logging

The mapping between the high-level architecture, the DUNE framework and the low-level code implementation is presented and discussed in more detail in Section V-C.

The Repository module consists of the following four sub-modules:

1) *Communication Protocol:* Defines which network technology is used by the IoT device and the PA, *e.g.*, BLE, ZigBee or LoRa.
2) *Data Collection Environment:* Defines DS arrival and departure processes, the data collection context, and the communication environment. The environment is a major determinant of DS's privacy preferences and available network types.
3) *Data Subject Model:* Defines the DS' privacy preferences, consent strategies, and other parameters. This module represents the DS model in the environment and depends on the data collection environment.
4) *Data Collector:* Represents the IoT device and defines its location, achieved utility and other parameters.

Each Repository module has a matching Analysis module that offers the respective module output analysis in terms of its performance. We discuss the Analysis implementation in more detail below.

Note: As can be seen from Figure 5 and will be shown and discussed in Section V-C, some of the modules are currently shared between the IoT device and the users. For example, negotiation and network protocols are shared between the two since the simulator relies on the same protocols being used by both parties and currently, we do not plan on supporting cross-technology simulations.

The rest of the simulator modules, *i.e.*, the simulation control and logging modules, allow us to control and debug the simulations.

B. Code Setup

Figure 6 shows the file and folder structure of the GEPARD project. GEPARD is implemented in Python 3.9.6.

The "requirements.txt" lists all the libraries used to implement the simulator and that are required to run it. Outside of the built-in libraries, we make use of the following 5 libraries:

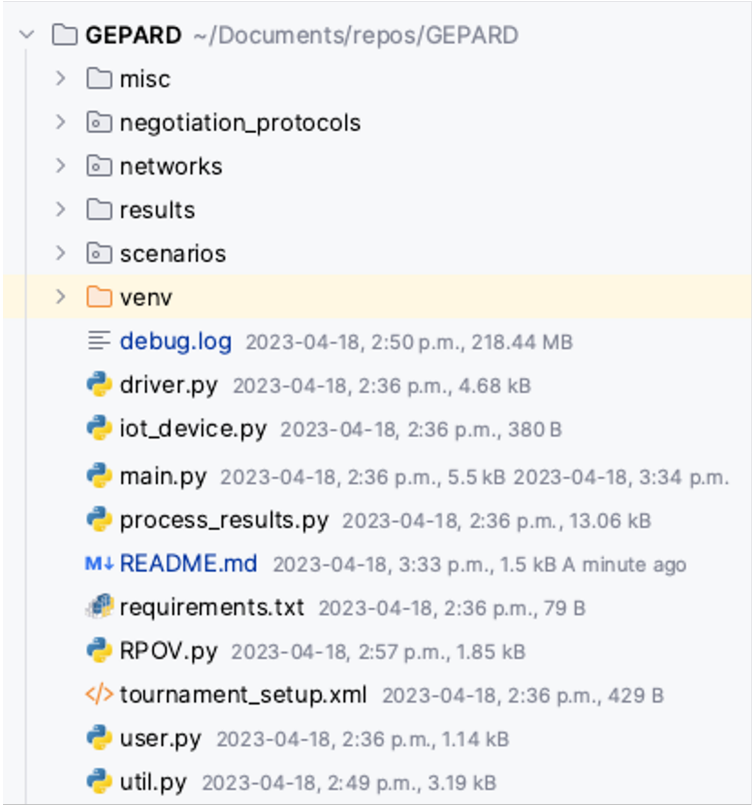


Fig. 6: GEPARD file and folder structure.

1) *matplotlib*: used to produce plots and figures, *e.g.*, user arrival and departure locations and trajectories.
2) *numpy* - used to support large, multi-dimensional arrays and matrices, along with mathematical functions to operate on these data structures, *e.g.*, calculating the mean values over all the runs and metrics.
3) *pandas* - used to store data from the "results.csv" file, and analyze and manipulate it.
4) *statsmodels* - used to implement Relative Proportion of Variation (RPOV).
5) *tqdm* - used to implement a progress bar and visualize the progress of iterables and loops in a readable and appealing way.

We now provide the description of what is included in each folder and the usage of each file:

- "networks" folder (Figure 7) includes the following network implementation and definition files:
  - "network.py" is a script that defines the "Network" class from which other network implementations are initialized and run. Any new network implementation have to be

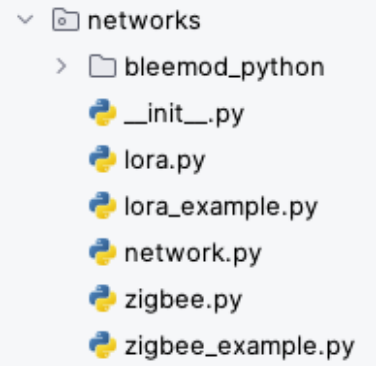


Fig. 7: "*networks*" folder structure.

defined in this class, *e.g.*, Listing 1 for ZigBee definition in the "Network" class.

Listing 1: ZigBee definition example.

```
self.network_type == "zigbee":
    self.network_impl = ZigBee()
```

– "lora.py", and "zigbee.py" are scripts that implement LoRa and ZigBee network technologies, respectively. Specifically, they implement methods to calculate energy consumption and timing information for different protocol states, such as scanning, connection establishment, and sending and receiving payloads. In our case, the payloads are PPs and consent.

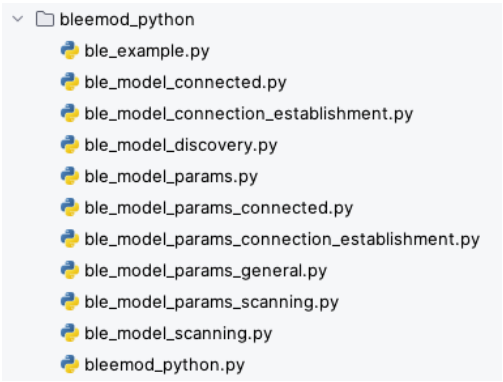


Fig. 8: "*bleemod_python*" folder structure.

– The "bleemod_python" folder contains code of the C-library presented by Kindt *et al*. [46] for precise BLE energy modelling and converted to Python (see Figure 8). We also made some minor modifications to the library source code to account for the specific needs of

the simulated PA designs. For example, we incorporated PP transmission during device discovery to enable Alanezi's PA design in the library.

– The network example files, *e.g.*, "zigbee_example.py", contain scripts that provide examples of the network class implementation and were used to validate the implementation of the network class usage (for more details, see Section V-D)

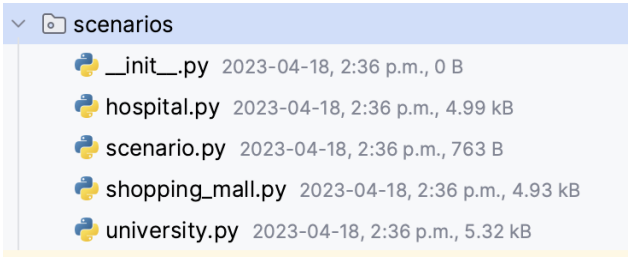


Fig. 9: "*scenarios*" folder structure.

- "scenarios" folder (Figure 9) includes various scenario implementation and definition files:
  – "scenario.py" is a script that defines the "Scenario" class from which other scenario implementations are initialized and run. Any new scenario implementation have to be defined in this class, *e.g.*, Listing 2 for the University scenario definition in the "Scenario" class.

Listing 2: University scenario definition example.

```
elif scenario == "university":
        self.scenario = University(list_of_users, iot_device)
```

  – "hospital.py", "shopping_mall.py", and "university.py" implement Hospital, Shopping Mall, and University scenarios, respectively. Specifically, they implement "generate_scenario()" and "plot_scenario()" methods. The "generate_scenario()" method generates users within the environment and populates the variables in the "User" class (see "user.py" discussion below) that represent users' speed, privacy preferences, arrival/departure times and trajectories in the environment. Currently, all users are generated using the Poisson arrival process in all environments. The "plot_scenario()" method produces a figure that shows the users' arrival and departure locations and their trajectory within the environment. Since we assume users traverse the environment in a straight line, the plot draws lines between two points (*i.e.*, arrival and departure) in space (see Figure 10).
- "negotiation_protocols" folder (Figure 11) includes negotiation protocol implementation and

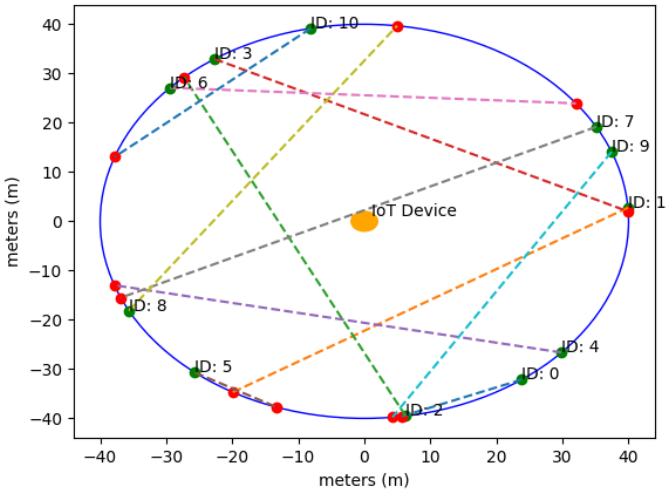


Fig. 10: Example of user arrival and departure locations shown in green and red, respectively, and the trajectory through the environment (Scenario: Hospital).

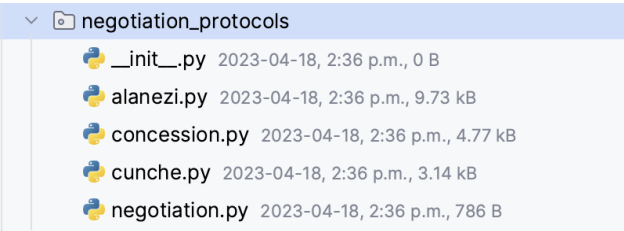


Fig. 11: "*negotiation_protocols*" folder structure.

definition files:

– "negotiation.py" is a script that defines the "NegotiationProtocol" class from which other negotiation protocols implementations are initialized and run. Any new negotiation protocol have to be defined in this class, *e.g.*, Listing 3 for Cunche *et al.* [10].

Listing 3: Cunche et al. negotiation protocol definition example.

```
elif self.algo == "cunche":
        cunche = Cunche(self.network)
        return cunche.run(list_of_users, iot_device)
```

– "alanezi.py", "cunche.py", and "concession.py" are scripts that implement Alanezi *et al.* [30], Cunche *et al.* [10], and a Concession agent classes, respectively. At a high level, the class structure for all the agents is the same. Each class includes the "run()" method,

which is the main method that drives the negotiation within the provided environment (which includes the list of User objects) and with the given network technology. The classes also implement the "consumption_for_user()" method that calls the respective negotiation protocol implementation utilizing a specific network technology and is called within the "ProcessPoolExecutor()" method used to parallelize the negotiations. For further discussions on parallelization in GEPARD, see Section VII.

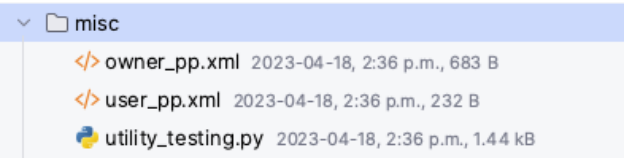


Fig. 12: "*misc*" folder structure.

- "misc" folder (Figure 12) includes the following miscellaneous files:
  - "owner_pp.xml" and "user_pp.xml" are an example of the owner and user PPs in an XML format, as defined by Alanezi *et al*. [29], [30]. These are used to determine the payload size when calculating the packet number and size for various network technologies.
  - "utility_testing.py" is a script that produces plots showing the assigned privacy coefficient for the first 100 users. It is used to verify that the privacy preference assignment algorithm works as expected.

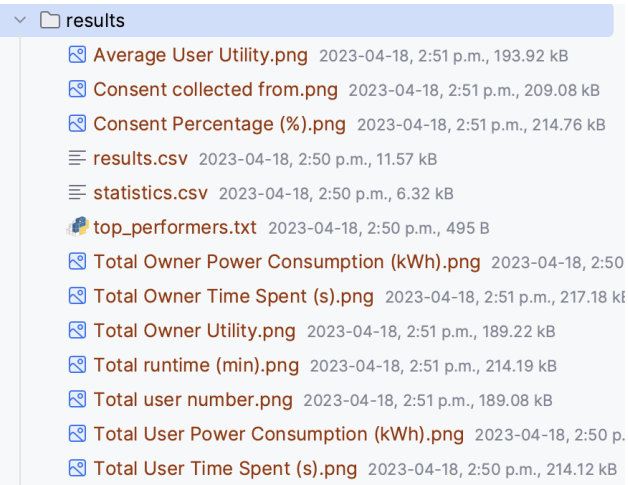


Fig. 13: "*results*" folder structure.

- "results" folder (Figure 13) contains the output results files:

Algorithm	Network	Scenario	Total User Current Consumption (W)	Total Owner Current Consumption (W)	Total User Time Spent (s)
alanezi	ble	university	0.25	0.43	68.75
alanezi	ble	hospital	0.76	1.32	210.29
alanezi	ble	shoppingmall	1.12	1.94	309.96
alanezi	zigbee	university	47.76	48.11	560.68
alanezi	zigbee	hospital	56.66	57.06	661.58
alanezi	zigbee	shoppingmall	203.89	205.32	2378.6
alanezi	lora	university	0.47	0.2	18137.19

Fig. 14: Example of "results.csv" file contents.

– "results.csv" (Figure 14) records the simulation results, *e.g.*, Total Power Consumption, Consent (%), *etc.*, of each run for each combination of the negotiation algorithm, network technology and scenario.

Algorithm	Network	Scenario	Avg User Power	Min User Power	Max User Power
alanezi	ble	university	0.24	0.21	0.26
alanezi	ble	hospital	0.74	0.69	0.82
alanezi	ble	shoppingmall	1.05	0.91	1.12
alanezi	zigbee	university	44.72	39.51	47.76
alanezi	zigbee	hospital	55.09	51.14	62.24
alanezi	zigbee	shoppingmall	199.28	187.89	206.57
alanezi	lora	university	0.49	0.43	0.55
alanezi	lora	hospital	0.62	0.57	0.69

Fig. 15: Example of "statistics.csv" file contents.

– "stastistics.csv" (Figure 15) records the calculated average, minimum, maximum, and standard deviation values per each unique combination of variables, *i.e.*, negotiation algorithm, network, and scenario.
– "top_performers.txt" (Figure 16) shows the top-performing combination of negotiation algorithm, network and scenario for each of the provided metrics.

Additionally, we have other files included in the root directory of the simulator:

- "main.py" is the starting point of the simulator. It initializes the required arguments and classes, implements the logic behind the tournament and single simulator runs, and initializes and calls the result processor class.
- "process_results.py" processes the "results.csv" file, calculates the minimum, maximum, average and standard deviation values for each variable combination, and saves the calculation

```
The combination with the least current consumption is:
Algorithm: cunche
Network: ble
Scenario: university
Current Consumption: 0.08

The combination with the highest user consent is:
Algorithm: cunche
Network: lora
Scenario: university
User Consent: 100.00

The combination with the least time taken is:
Algorithm: alanezi
Network: ble
Scenario: university
Time Taken: 66.17

The combination with the highest user utility is:
Algorithm: cunche
Network: zigbee
Scenario: university
User Utility: 70.27
```

Fig. 16: Example of "top_performers.txt" file contents.

results in "statistics.csv". It also computes the top-performing combinations for different metrics and saves them in "top_performers.txt".

- "iot_device.py" implements the IoT Device class. It stores the IoT device's achieved utility, its location in the environment and a list of weights used in utility calculation, *i.e.*, emphasis or weight on energy consumption or data.
- "user.py" implements the User class. It stores the user's id, speed, various major locations (*e.g.*, arrival location) and timing (*e.g.*, arrival time) information, as well as the user's privacy preference, utility and weights. It also implements methods to update the user's utility, location, and consent data.
- "driver.py" is the class used to "drive" the simulation. This is where we step through the simulation time, determine current users in the location and call the "run()" method implemented in the "NegotiationProtocol" class to compute the various metrics.
- "util.py" implements various utility methods used in the simulator. It includes methods for utility, distance and timing calculations. It also implements a method to write data to the file.
- "tournament_setup.xml" defines the tournament configuration. It includes information on how many runs are to be executed, for which network types, negotiation algorithms, and scenarios to run the simulator. Listing 4 presents an example of the tournament setup XML file.

Listing 4: Example XML of tournament setup.

```
<simulation>
      <runs>5</runs>
      <networks>
            <network>ble</network>
            <network>lora</network>
            <network>zigbee</network>
      </networks>
      <scenarios>
            <scenario>university</scenario>
            <scenario>hospital</scenario>
            <scenario>shoppingmall</scenario>
      </scenarios>
      <algorithms>
            <algorithm>alanezi</algorithm>
            <algorithm>cunche</algorithm>
            <algorithm>concession</algorithm>
      </algorithms>
</simulation>
```

- "RPOV.py" implements the RPOV and needs to be called separately from the simulator code, as it requires several tournament style runs and is of interest only in certain types of analysis, *e.g.*, to determine which components contributed the most to the variance of the system's performance with respect to the chosen metric.
- "debug.log" stores the simulator's debug information. It includes such data as the number of users, each time step details, *e.g.*, number of users in the environment, number of consented users, *etc*.

C. Architecture to Code Mapping

Figure 17 shows the mapping between the DUNE framework, the GEPARD architecture and implementation. From the figure, it is possible to see how each of the DUNE framework components maps to the corresponding component in the GEPARD repository, which in turn directly maps to the code implementation of each component.

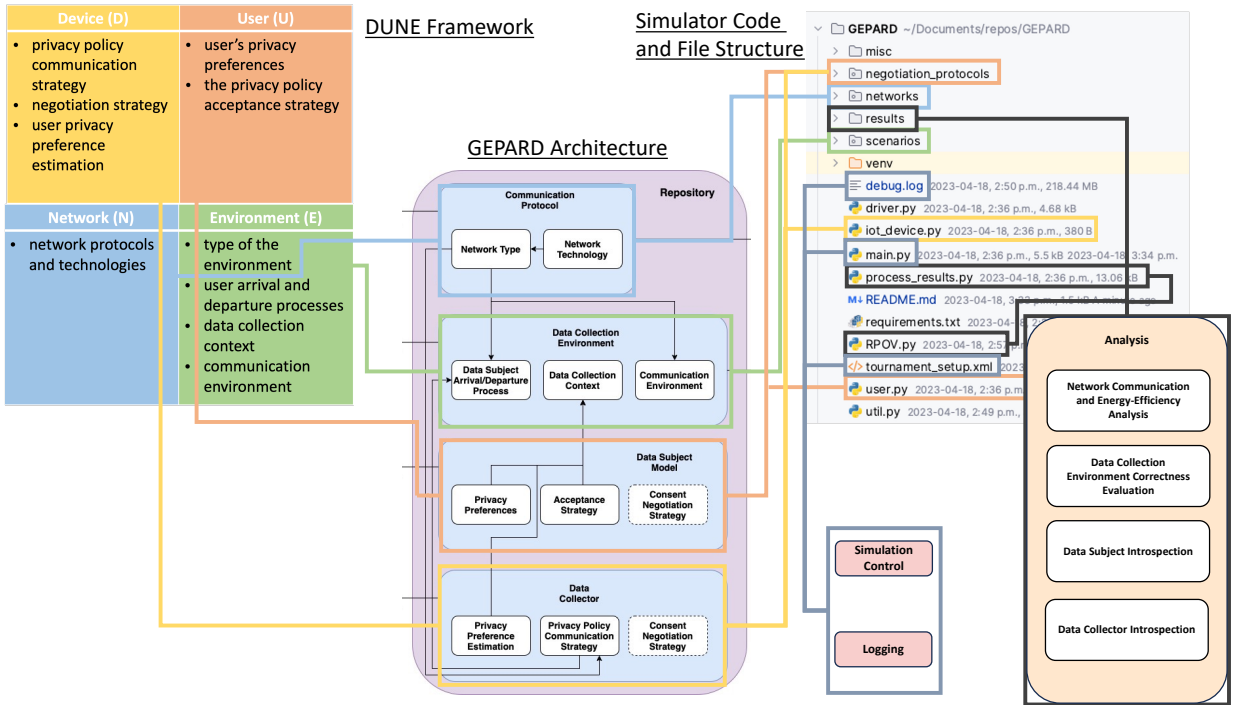


Fig. 17: DUNE framework to high-level GEPARD architecture to code mapping.

For example, one can trace how the **N**etwork component in the DUNE framework links to the *Communication Protocol* module in the *Repository* of the GEPARD's architecture, which then links to the "networks" folder in the source code. Figure 7 then presents the contents of the "networks" folder, which includes the classes implementing BLE, LoRa and ZigBee network technologies (for more details, see Section V-B).

Figure 17 also highlights some of the DUNE and GEPARD design decisions. Specifically, negotiation protocols implementation in Simulator Code maps to the Devices and Users in DUNE and Data Subject Model and Data Collector model GEPARD architecture, respectively, since we rely on both to use the same protocol to negotiate. Consequently, when driving the simulator, we rely on negotiation protocol classes to drive some parts of the simulation, *i.e.*, which packets are sent, between which parties and when, while utilizing the surrounding infrastructure, *i.e.*, network and the scenario (we discuss more details in Section V-D3). Currently, the network technologies between the IoT and the user devices similarly should match, as we do not support Cross-Technology Communications (CTC) as was proposed in existing literature, *e.g.*, Chen *et al.* [47] present a survey of CTC from the hardware perspective.

D. GEPARD Parameterization and Validation

In this section, we discuss the specifics of GEPARD parameterization and the steps we took to ensure that the implemented simulation model is a faithful representation of the real IoT environment and PA system. We split the discussion across the three main driving components of the GEPARD: Scenarios, Network Technologies and Algorithms.

1) Scenarios: To simulate realistic IoT scenarios in terms of the user arrival process, we relied on the Poisson arrival process, common in modelling user interarrival times. Table II presents the parameters defined in each scenario. In the Poisson process, λ is a fundamental parameter that represents the average arrival rate per unit of time and directly influences the shape of the Poisson distribution, governing the average number of events in a fixed interval. To this extent, we use the arrival rate values presented in Table II. In the table, you can also observe that we vary the space size across the scenarios. This is done to: i) showcase the importance of careful consideration of PA component choice for different environments, and ii) account for the common differences in space sizes across the environments. Consequently, the arrival and departure locations are generated uniformly at random in the specified space sizes. We also vary the total simulation times between the scenarios since the operating hours of the businesses represented in each scenario would also change.

TABLE II: Simulation Parameters for Different Scenarios

Scenario	**Arrival Rate (users/min)**	**Space Size (m)**	**Total Simulation Time (h)**	**Reference**
Hospital	0.1584	40	24	[48]
University	14.18	80	8	[49]
Shopping Mall	2.55	120	10	[50]

The Poisson process was implemented using Python built-in library “random”. To verify that the users are generated in the correct space constraints, we used a sample of users to plot their arrival and departure locations and trajectory across the space (as shown in Figure 10).

It is important to consider that users walk at different speeds, so we generate user speeds ranging from 0.27 to 1.5 meters per second according to Fitzpatrick *et al*. [51]. These speeds are used in the Shopping Mall scenario. However, to highlight the differences across the scenarios, we assume that in the University scenario, users walk 10% faster, while in the Hospital scenario, they walk 10% slower.

Ken: What about them stopping for coffee?

The user privacy preference categories were taken from Westin's work as presented by Kumaraguru and Cranor [52]. Westin categorized individuals into three privacy categories: (i) privacy fundamentalists that strongly value privacy and tend to be protective of their personal information, (ii) privacy pragmatists that adopt a more balanced approach to privacy and are willing to trade some level of privacy for convenience, and (iii) privacy unconcerned that place a lower value on privacy and often are willing to share personal data freely. The privacy preferences affect the number of negotiation rounds and consent probabilities. Specifically, when not mentioned otherwise, we follow the results of the questionnaire written by Phelps *et al.* [53] as follows:

- 25% privacy fundamentalists, which, are split into 20.4% that will consent and 79.6% that will not consent, resulting in 5.1% consenting and 19.9% not consent out of the total population. Consenting parties, however, will not always consent within the 1st round of the negotiation; as such, we use the 25/75 split, resulting in 1.28%/3.82% for the 1st and 2nd round consent, respectively.
- 55% pragmatists, which will result in split of 73.55% consent and 26.45% not consent, or 40.45% consenting and 14.55% not consenting out of the total population. For pragmatists, we follow a similar trend as for fundamentalists; however, with reversed percentages, i.e., 75/25 split, or 30.34%/10.11% for the 1st and 2nd round consent, respectively.
- 20% unconcerned, which always consent in the 1st round of the negotiation.

2) Network Technologies: In GEPARD, we've implemented three network technologies, namely BLE, ZigBee, and LoRa. We chose these network technologies as they are common choices for IoT environments, offer diverse communication ranges and are known for low power consumption.

For BLE, we used the "BleeMod" library presented by Kindt *et al*. [46]. We translated the implementation to Python and included some minor modifications to account for the specific needs of the simulated PA designs. To verify the correctness of our translation and implementation, we ran the example codes provided by the authors in the original library and compared their output with our translation output, which perfectly aligned.

For ZigBee, we utilized the work presented by Casilari *et al*. [43]. While the authors provide both the analytical model and real-world current consumption measurements for ZigBee, they did not include any exact way for us to compare our implementation eg, there were no example codes or software implementation provided. Consequently, we utilized the estimated values

collected from the presented figures within their work and plotted against them the results of our implementation in GEPARD (see Figure 18). As can be seen from the figure, our implementation results align closely with the model presented by Casilari *et al.*, and any inconsistencies can be explained by estimation errors of the figure values.

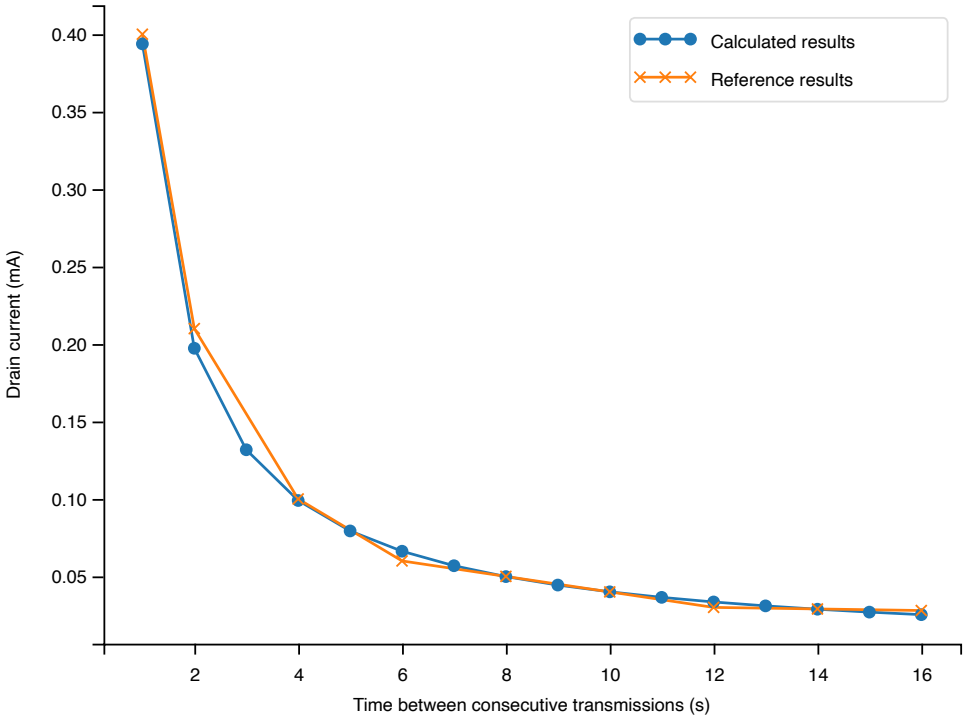


Fig. 18: Implementation vs. Analytical model estimation of Zigbee drain current

Finally, for LoRa, we made use of the model presented by Bouguera *et al.* [54]. To verify the correctness of our implementation of the model, we opted to make use of existing LoRa energy consumption and airtime calculators[2] and compared the results provided by our implementation and the calculators to ensure that they align.

It is worth noting that we assumed a specific effective communication range value for each network technology outside of which the PP negotiations cannot occur (see Table III). The provided values were chosen with the following goals in mind: i) highlight the effects of networking technologies on the PA performance in various scenarios, while ii) keeping the values realistic [55]. To this end, we chose BLE to have a lower range than ZigBee (within

[2]We made use of the Semtech LoRa calculator in https://www.semtech.com/design-support/lora-calculator, where Semtech is silicon developer of LoRa and founding member of LoRa Alliance, a non-profit technology alliance that defined LoRaWAN.

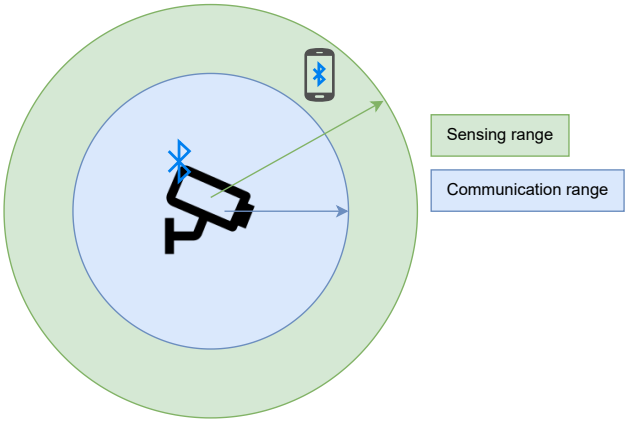


Fig. 19: Example of sensing and communication range mismatch in IoT.

the known effective communication ranges), although we recognize that the actual range can vary based on environmental factors, interference, and specific implementations. Consequently, not all generated users will be in the range of communication technology. There will be cases when the user is within the data collection range, *i.e.*, the sensing range, but not within the PP negotiation range, *i.e.*, communication range. For example, if the sensing range of the video camera is assumed to be 100 m, but the IoT device uses BLE to communicate the PP, then we will still observe and collect data about people outside of the communication and consent range (see Figure 19).

TABLE III: Network technologies and communication ranges.

Network Technology	**Effective Communication Range (m)**
BLE	50
ZigBee	100
LoRa	10000

3) Algorithms: In GEPARD, we consider three negotiation algorithms, namely Alanezi [30], Cunche [10], and Concession [11].

(1) Alanezi.

Alanezi *et al*. proposed the following negotiation exchange [29]:

1) PA on a user device broadcasts its data request information (*i.e.*, PP) within the IoT environment

2) the IoT device is scanning for the request, and, when received, calculates the utility of the request using the following utility-privacy function [29]:

$$U = -\gamma.P_e(t, r, s, i) + B(t, r, s, i), \tag{1}$$

where U denotes the total utility achieved from pursuing the information exchange, B is the benefit from the data exchange as seen from the IoT device perspective, γ is an overall privacy sensitivity perception factor, P_e is the degree in privacy exposure for the selected privacy policy and (t, r, s, i) are the data type, retention, shared, and inferred elements specified in the privacy policy that affect the benefit and privacy exposure values.

3) if the utility value of the request is equal or higher than the utility from the IoT PP, then the IoT device accepts the request, and the negotiation is done. Otherwise, IoT device sends its own PP as a counter-offer to the user's PA, which performs the utility checks and either accepts or rejects the offer.

Overall, Alanezi specifies a maximum 2-phase negotiation, following which, if the parties do not come to an agreement, the user will get a notification that their privacy requirements cannot be met and detailed information about the data collection that will take place. The users can then choose to stay in the environment or leave the location.

In their work, Alanezi *et al*. do not provide specific values for the above-mentioned variables or details on calculating them. However, they reference the work of Prebusch [56] as the main inspiration for the utility function. In their work, Prebusch defined γ to be near 1 for privacy fundamentalists, near 0.5 for privacy pragmatists, and near 0 for privacy unconcerned. However, we had to look somewhere else for benefit and cost values. Specifically, we need to reflect how values of t, r, s, i change between alternative PP. We also need to account for the effect of γ on the calculated utility. To this extent, we assume the following:

- Privacy unconcerned will always agree to consent even to the first offered PP.
- Privacy pragmatists will agree to the second (alternative) PP in 23.55% of cases, assumed from the 73.55% mentioned above, alongside the questionnaire answers within the previously mentioned work written by Phelps *et al*. [53]. They will also consent to a third offered PP in 50% of cases. However, there will be the remaining 23.45% of pragmatists that will not agree to any PP and who, due to only having a 2-phase negotiation option, will leave the area.

- Finally, Privacy fundamentalists will consent to the strictest PP in 20.4% of cases (again following the above-mentioned distributions) and will not consent to anything else, leaving the area.

Note: For simplicity of the simulations, we assume that the IoT owner will always communicate the correct PPs to the correct individuals, and they will consent or not consent following the above-described behaviour.

(2) Cunche.

Cunche *et al*. proposed the following negotiation exchange [10]:

1) the IoT device broadcasts its PP into the IoT environment.
2) user device scans the environment for the PP broadcasts, and, when received, compares its own PP with the received PP.
3) the user PA can agree to the PP or begin the negotiation process.
4) during the negotiation process, the user PA will forward their PP to the IoT device, which will compute the terms on which the two policies agree and communicates them back to the user if agreeable PP has been found.

It is worth noting that in their work, Cunche *et al*. assume that the range of the collecting devices can be tuned to fit the range of the PAs to ensure that the data collection is aligned with the PPs. Additionally, the authors assume that the PPs are retrieved between 1 and 5 seconds after the user enters the IoT area. In our work, we argue that these are critical assumptions when developing PAs as they greatly affect the performance and feasibility of the PA design.

Note: Cunche *et al*. specify a gateway device between the IoT and user devices; however, in the current GEPARD design, we simulate a direct communication between the IoT and user devices, which the authors also accounted for in the initial paper [6]. Additionally, the current version of GEPARD does not fully simulate the described negotiation process, *i.e*., we do not simulate PP computation and agreement. Instead, we assume that some percentage of negotiations succeed, where the percentages are assumed following the privacy preference distributions described in Section V-D1.

(3) Concession.

We use the Concession algorithm as a baseline against which to compare the rest of the algorithms. The Concession algorithm follows a simplified version of a time-dependent concession-making agent, commonly used in automated agent-based negotiations [11]:

1) for each user in the IoT environment, the IoT device estimates a utility depending on the estimated time t the user will be in the environment:

$$U = e^{-t}, \tag{2}$$

The idea is that the less time is left for collecting the user's data, the more valuable it becomes, *i.e.*, we rely on quantity and variety of data instead of quality. For simplicity, it is assumed that we know precisely how long the user will be in the environment.

2) the IoT device sends it's PP to the n higher utility users within the environment (we set $n = 5$ for our experiments).
3) the user has to reply with consent or leave the environment.

4) Privacy Policies: Each of the presented three algorithms also specifies its PP format, outlining the possible negotiation space and, ultimately, affecting network resource and power consumption.

(1) Alanezi.

The format and contents of the PPs determine the payload that will need to be transmitted through the network and will be a significant factor in determining the energy consumption of the devices. Alanezi *et al*. specified PPs to be stored in an XML file with <data-in> and <data-out> tags defining the type of data that one party wants to acquire from another party [30]. Currently, we make use of previous work [57] and [58] to argue that it is safe to assume only 1 deployed sensor in the environment, *e.g.*, for occupancy tracking and interior analytics to understand shoppers' paths, actions and behaviour. It is common to use video sensors for such data collection and analytics; therefore, we focus the PPs on that specific type of sensor (see Listings 5 and 6). Nonetheless, it is worth mentioning that occupancy counting can also be achieved through other sensors, such as Infrared (IR) and ultrasonic [59], and, as such, the PPs can be adjusted to reflect the alternatives.

Note: While Alanezi *et al*. [30] do specify policy priority and ability to clarify an alternative policy, we assume that most users will not specify such policies. Specifying alternative policies would require users to go through the existing policies and manually adjust them, which the users are not ready to do with much less time-consuming privacy-related aspects, *e.g.*, as can be seen from the results in user privacy paradox and attitude research literature [60].

Listing 5: IoT owner PP XML file example.

```
<?xml version="1.0" encoding="utf-8"?>
<privacy-policy>
      <data-in>
            <data-in type="video" priority="1"></data-in>
            <retention>1-year</retention>
            <shared>yes</shared>
            <inferred>yes</inferred>
      </data-in>
      <data-in>
            <data-in type="video" priority="2"></data-in>
            <retention>1-year</retention>
            <shared>no</shared>
            <inferred>yes</inferred>
      </data-in>
      <data-in>
            <data-in type="video" priority="3"></data-in>
            <retention>30-days</retention>
            <shared>no</shared>
            <inferred>yes</inferred>
      </data-in>
      <data-in>
            <data-in type="video" priority="4"></data-in>
            <retention>30-days</retention>
            <shared>no</shared>
            <inferred>no</inferred>
      </data-in>
</privacy-policy>
```

Listing 6: IoT user PP XML file example.

```
<?xml version="1.0" encoding="utf-8"?>
<privacy-policy>
      <data-out>
            <data-out type="video" priority="1">
            <retention>30-days</retention>
            <shared>no</shared>
            <inferred>no</inferred>
            </data-out>
      </data-out>
</privacy-policy>
```

To further elaborate on the XML files, from Listing 5, we can see that the IoT owner specifies multiple alternative policies, *i.e.*, multiple <data-in> clauses, as the owner is interested in getting as much data as possible from the device and specifying a larger number of alternatives allows collecting small bits of data even from privacy fundamentalists. Consequently, specifying alternative policies would allow the IoT owner to achieve more "profitable" data points, as the data points for users, less likely to share their information, would also be more valuable. In general, in the case of IoT owner PPs, we try and collect the data for as long as possible and use it for sharing and inference. However, if that does not work, we attempt to: (i) go to lower priority policies, (ii) remove the sharing (we still want to store the data and infer from it), (iii) reduce the number of days we store the data (since it is better to have the data for less time but have the use of it for more purposes), and, (iv) we remove the inference, so that we at least can keep the recording, *e.g.*, security purposes.

From Listing 6, we can see that the user wants video data to stay in the system for a maximum of 30 days, not to be used for inference or any 3^{rd} party sharing. Note, this only shows an example policy, and users that fall under privacy unconcerned would have more lenient PP. Consequently, we assume IoT users and owners use the defined PPs, resulting in policy sizes of 217 bytes and 639 bytes, respectively.

(2) Cunche.

The authors define that the IoT broadcasts 86 bytes PP (using *Advertising* capabilities of the BLE). The consent is sent using the Attribute Protocol. In the Cunche *et al*. prototype case, the consent message consists of the MAC address of the smartwatch and the hash of the owner's

PP. The user's PP is retrieved between 1-5 seconds after a user enters the area. Authors specify that the users can adjust the broadcasted PP in any way, shape and form and then consent (or not). As such, close to 100% of the people will consent (except for maybe the fundamentalist, as they may be too restrictive and make it impossible to consent). The authors do not specify which hash function was used for hashing the PP, so we assume a SHA-256 cryptographic hash function. Its output is always 256 bits long. MAC address is 48 bits. As such, the user consent reply is 304 bits or 38 bytes long.

Note: Due to the setup of the negotiations, we assume that most people will consent to some degree of observation, except for the subset of privacy fundamentalists who would never agree to anything. There is also no negotiation phase, as the IoT owner will accept anything the user offers. Generally, we assume that only 19.9% of the total number of users will not consent. For example, if the user is in the privacy fundamentalists category, they have a 79.6% probability of not consenting.

(3) Concession. We assume the same PPs as in Cunche.

VI. Utilizing GEPARD: Experimental Evaluation

In this section, we delve into the practical application of our developed simulator, GEPARD, and present the results of utilizing GEPARD for different PA design evaluations.

A. Experiment Setup

Table IV presents the setup we used in our experiments. As can be seen from the table, we ran each algorithm 10 times, utilizing different network technologies and scenarios.

TABLE IV: Experimental setup.

Description	Values
Runs	10
Scenarios	Hospital, University, Shopping Mall
Network Technologies	BLE, ZigBee, LoRa
Algorithms	Alanezi, Cunche, Concession

B. Results

As was discussed in Section V-B, GEPARD provides detailed performance results and statistics of the simulated PA designs using a variety of metrics. Given that GEPARD is intended for public availability and reproducibility, we refrain from presenting an exhaustive compilation of all experiment results and statistics. Instead, we selectively showcase a subset of particularly noteworthy findings to provide insight into the capabilities and outcomes of our experiments. Other results can be found in Section C. Specifically, we discuss average user consent and power consumption of different PA designs.

1) Average User Consent: We measure the average user consent as a percentage of consent collected by an IoT device averaged over all the runs.

Figure 20 presents the average user consent percentage when PA negotiation component is designed following Alanezi *et al*.

Note: Similar trends were observed for other algorithms (detailed in Appendix C-1). We only feature a representative figure in the main text to conserve space.

From the figure, we can observe the expected impact of the networking technology on the quantity of collected consents across different scenarios. As the coverage of the networking technology diminishes within the simulated scenario spaces, *e.g*., the Hospital scenario has a

smaller space than the Shopping Mall scenario, the IoT device cannot reach a growing number of users. This result shows the criticality of careful design choices that need to be taken when designing PA, especially in such information-sensitive environments as hospitals. We can also observe that, for example, in the Shopping Mall scenario, BLE based PA design would only be able to collect consent from a fairly small proportion of users, compared to ZigBee based PA. Specifically, BLE based PA collects 37.91% less user consent than ZigBee based PA. At the same time, LoRa, known for its long communication range, would provide expected, close to 100%, user consent collection. Notably, these results are accompanied by trade-offs in terms of power consumption and network capacity, as discussed below.

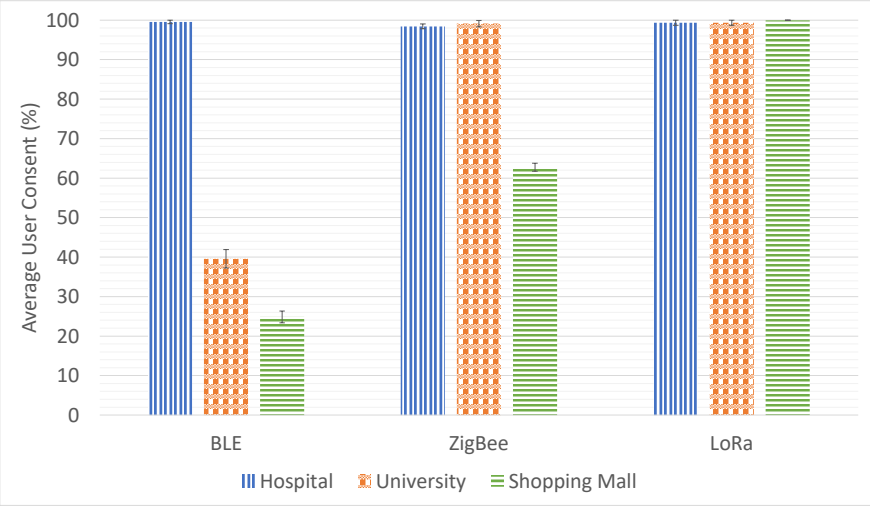
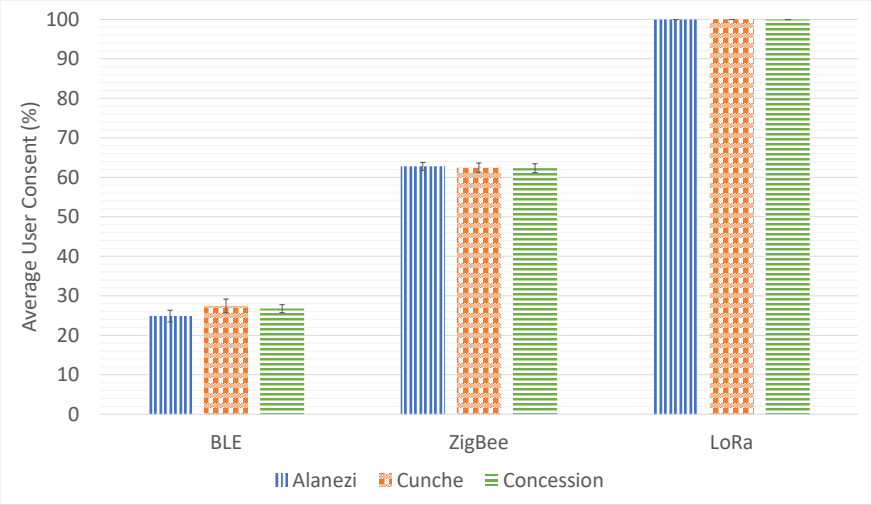


Fig. 20: Average user consent percentage for different network technologies and scenarios under Alanezi.

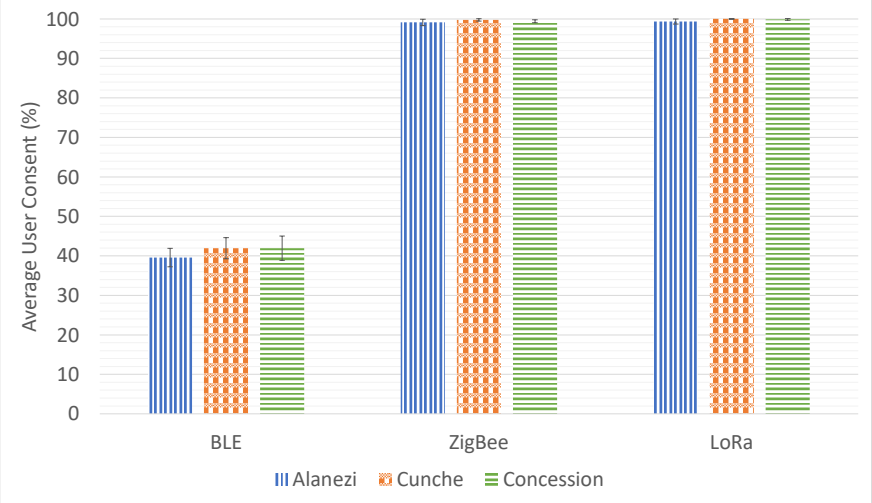
Figure 21 presents the average user consent percentage with a focus on the effect of the algorithm choice, *i.e.* moving from Alanezi to Cunche and Concession. From the figure, we can observe that in each scenario the chosen algorithms perform similarly. Specifically, in the Shopping Mall scenario (see Figure 21(a)), the biggest difference in terms of consent is 2.57%, between Alanezi and Cunche based PA.

2) Average Power Consumption (Users): In this metric, we measured the combined average power consumption of all user devices, *e.g.*, smartphones, and averaged it over the total number of runs.

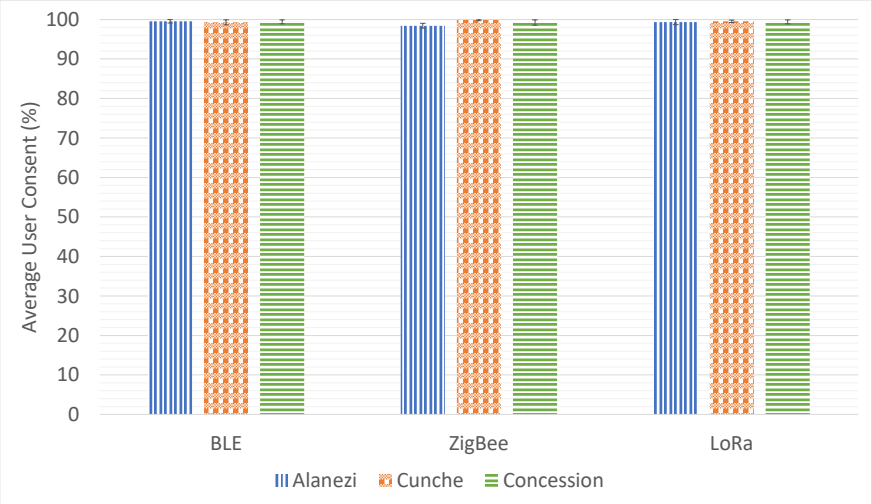
Figure 22 presents the average power consumption of all user devices in the Hospital scenario. We chose the Hospital scenario as a representative scenario because only in this scenario devices



(a) Average user consent percentage for different network technologies and negotiation algorithms in Shopping Mall scenario.



(b) Average user consent percentage for different network technologies and negotiation algorithms in University scenario.



(c) Average user consent percentage for different network technologies and negotiation algorithms in Hospital scenario.

Fig. 21: Average user consent percentage for varying scenarios.

have 100% of the space and, as such, will attempt to negotiate with all the users in the space (close to 100% average user consent as shown in Figure 21(c)).

Note: Similar to the previous set of results, due to similar trends being observed for other scenarios (as detailed in Appendix C-2) and to conserve space, we only feature a representative figure in the main text.

From the figure, we can observe that, at best, ZigBee based PA results in close to 7 and 75 times more energy consumption than LoRa and BLE based PAs using Alanezi and Concession, respectively. Albeit surprising, since ZigBee is expected to have energy efficiency on par with BLE, such results can be explained by the unique communication pattern observed in each PA algorithm. Specifically, we identify the following as some of the reasons for ZigBee based PA to consume more power than BLE based PA:

1) In BLE based PA the PP is sent alongside device discovery, while in ZigBee based PA the first PP exchange only occurs after the association has been concluded, thus consuming more energy, in ZigBee case, on extra transmission.
2) Although BLE and ZigBee have somewhat comparable transmission power consumption (*i.e*. commonly there is only around 10 mA larger consumption in ZigBee), the achievable data rate of BLE is 4 times larger than ZigBee's. Such discrepancy allows BLE to have shorter transmission times and, consequently, less power consumption.

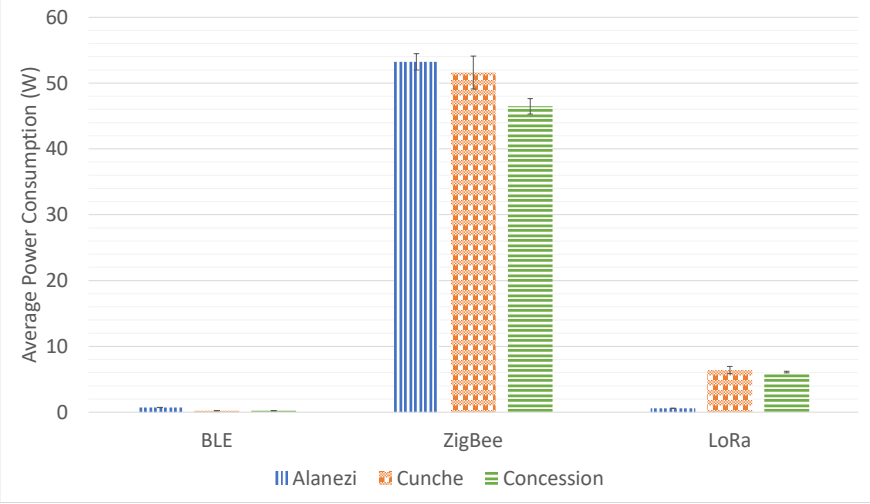


Fig. 22: Average user power consumption for different network technologies and negotiation algorithms in the Hospital scenario.

To showcase the implications of the calculated average power consumption, we use the results

to calculate how many hours an average 4000 mAh smartphone battery would last if it was only performing the negotiation. We chose the 4000 mAh value as it is a common battery size used by Android users [3]. Assuming the 3.7 V nominal charge, we can calculate the battery life in hours as follows:

$$\text{Battery Life (hours)} = \frac{\text{Battery Capacity (Wh)}}{\frac{\text{Average Power Consumption (W)}}{\text{Total Simulation Duration (hours)} \times \text{Average Number of Users}}} \tag{3}$$

Table V presents the estimated smartphone battery life in each scenario under Cunche. These results emphasize the importance of holistically looking at the PA designs as part of an IoT environment and considering the technical and infrastructure limitations when devising privacy negotiations. From the results, we can observe that in some scenarios, *e.g.*, using Cunche with ZigBee in University, would result in 19.29 days of operation for the device, while alternative network technologies, *e.g.*, BLE, in the same environment, provides 10672.24 days or 29.24 years of operation.

TABLE V: Smartphone Battery Life Estimates for each Scenario.

Scenario	**Battery Life (hours)**	**Battery Life (days)**
Cunche, BLE, University	256133.67	10672.24
Cunche, BLE, Hospital	352293.82	14678.91
Cunche, BLE, Shopping Mall	533399.05	22224.96
Cunche, ZigBee, University	462.85	19.29
Cunche, ZigBee, Hospital	1501.74	62.57
Cunche, ZigBee, Shopping Mall	924.32	38.51
Cunche, LoRa, University	3744.02	156.00
Cunche, LoRa, Hospital	12110.10	504.59
Cunche, LoRa, Shopping Mall	4686.77	195.28

3) Average Power Consumption (IoT Device): In this metric, we measured the power consumption of an IoT device averaged over the total number of runs.

Note: Similar to the previous set of results, due to similar trends being observed for other scenarios (as detailed in Appendix C-3) and to conserve space, we only feature a representative figure in the main text.

Figure 23 presents the average power consumption of IoT device in the Hospital scenario. From the figure, we observe similar behaviour to Figure 22. These results are expected since

[3]https://www.androidauthority.com/smartphone-battery-size-poll-results-1221015/

the simulated PA designs have symmetrical negotiation algorithms between the user and IoT device, *i.e.*, in each negotiation phase, each device will receive and transmit a similar number of packets.

Note: These results arise due to some of our simulation assumptions, i.e., the user's consent is the user's PP, meaning the consent packet size is larger than it needs to be because we could have represented consent with a single binary value. Additionally, we use the same energy consumption models for IoT and user devices, while in practice, the IoT device would be optimized for privacy negotiations.

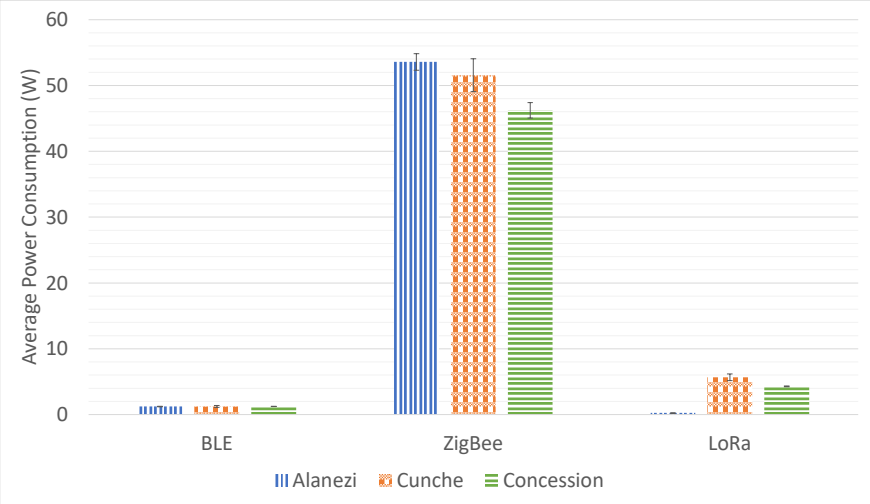


Fig. 23: Average IoT device power consumption for different network technologies and negotiation algorithms in the Hospital scenario.

4) Component Impact Analysis: To evaluate the impact each of the components has on the resulting performance of PA, we performed a relative proportion of variation analysis. To this extent, we used *smf.ols* method from the *statsmodels.formula.api* module that fits a linear regression model using the Ordinary Least Squares (OLS) method on the "results.csv" file contents. We use the OLS model to compute the Analysis of Variance (ANOVA) table, which allows us to analyze the variance between different data groups, *i.e.*, PA components. The OLS model is used to compute the ANOVA table because it provides a way to partition the sum of squares into components associated with each variation source. The ANOVA table is then constructed using these components.

Note: We could also use other methods for ANOVA analysis, such as Generalized Linear Model (GLM) and Mixed Effects Model (MEM). GLMs generalize linear regression models

allowing non-normal distributions of the response variable and non-constant variance. MEMs are used when there is a need to account for random effects in the data. Currently, we focus on using OLS as it is a simple and widely used method for linear regression analysis and is computationally efficient and easy to interpret.

TABLE VI: Relative proportion of variation analysis results.

	Average User Consent Percentage (%)	**Average User Utility**	**Average User Power Consumption (W)**	**Average User Time Spent (s)**
Network	44.20	17.09	52.88	34.85
Scenario	28.44	11.56	19.87	28.59
Network:Scenario	27.14	8.42	25.55	14.53
Algorithm	1.58×10^{-4}	38.98	8.50×10^{-6}	8.05
Network:Algorithm	8.55×10^{-5}	9.90	0.9	4.02
Algorithm:Scenario	3.20×10^{-5}	1.89	8.63×10^{-5}	3.30

Table VI provides the analysis of the factors influencing average user consent percentage, average user utility, average user power consumption, and average user time spent. Each row corresponds to a specific factor, while the columns depict the proportion of variation attributed to each factor. Simply put, the component that exhibits a higher relative proportion of variation contributes more to the system's overall performance in terms of the performance metric. For instance, if Network scores higher than Scenario, then Network has a greater impact on the system's behaviour compared to Scenario.

In terms of **Average User Consent Percentage**, the dominant factor is the Network, accounting for 44.20% of the variation. However, both Scenario (28.44%) and the interaction between the Network and Scenario (27.14%) play substantial roles. These results align with expectations since the environment and the context will determine user privacy decisions, while the network technology will affect what is possible to collect, from how many users and how often. However, the main interest lies in the observation that the Algorithm component has a lower-than-expected effect on the consent percentage. Such behaviour can be attributed to all the chosen algorithms achieving similar consent percentages, meaning that as a PA designer, one can be inclined to go with the simplest and easiest choice.

Moving to **Average User Utility**, the Algorithm becomes the most influential factor, contributing 38.98% to the variation. Network and Scenario also play significant roles, with percentages of 17.09% and 11.56%, respectively. The joint impact of Network and Algorithm is noteworthy

at 9.90%. These results align with our expectations. Firstly, the Algorithm has the most influence on average utility since, across the proposed algorithms, each calculates utility in its own way and prioritizes different sets of values in its calculations. At the same time, all utilities are affected by the context (defined by Scenario) and the network technology (the primary consent collection medium, which affects how much utility can be achieved).

In the realm of **Average User Power Consumption (W)**, the Network stands out as the primary contributor, responsible for 52.88% of the variation. The joint effect of Network and Scenario follows closely at 25.55%, and the Scenario alone contributes 19.87%. These results align with expectations since the two primary contributors to the variation, Network and Scenario, are the primary determinants of power consumption per connection (Network) and the number of connections that have to be established (Scenario). Similar to Average User Consent Percentage, one can observe that the Algorithm is not a dominant factor affecting power consumption. This can be attributed to the limited number of negotiation rounds and the PP sizes that the algorithms define.

Concerning **Average User Time Spent (s)**, the Network plays a pivotal role, accounting for 34.85% of the variation. The joint impact of Network and Scenario is 28.59%, and the Scenario alone contributes 14.53%. Similar to the power consumption, the Network is expected to be the primary factor, with Scenario following in second place, determining the amount of time spent in the communication/negotiation.

Overall, we can observe that no single component reliably determines the PA's performance across different metrics. Depending on the PA design goals, some PA designs may be preferred over others, *e.g.*, if the power consumption is of no concern (the devices are plugged into a power source), but the goal is to collect as much data as possible for the users (*i.e.* device needs to collect as much consent as possible), then the choice of network technology has to be carefully considered.

Consequently, the importance of these results lies in their potential to guide PA design and optimization efforts. Understanding the dominant factors provides valuable insights for prioritizing improvements. For example, enhancing network-related aspects may have a pronounced impact on consent, power consumption, and time efficiency. Similarly, addressing algorithm-related issues could significantly improve average user utility.

VII. Discussions

In this section, we discuss the results, design principles, and future directions for our work.

A. GEPARD Design and Implementation

We see GEPARD as a crucial tool in designing PAs for IoT environments. However, the current implementation of GEPARD has limitations that may be critical in designing some of the PAs. One such limitation is that the simulator does not account for the network capacity, *i.e.*, number of supported end devices, when processing the negotiations between the devices. This limits the applicability of results provided by GEPARD with respect to the scalability of the PA solution. For example, the simulator will still simulate the negotiations even if, in reality, there is no capacity left in the network to support the negotiations. Another limitation of the GEPARD implementation is its runtime. The experiments discussed in Section VI-A took a little less than 4 days to run on a 2017 Intel Core $\mathrm{i}5$ Macbook Pro with 16 GB RAM. We attempted to reduce the runtime by utilizing the "ProcessPoolExecutor" class from the "concurrent.futures module" in Python for power and time consumption calculations within the negotiation class. We used the "ProcessPoolExecutor" class instead of the "ThreadPoolExecutor" class as it provides better results for CPU-bound tasks, *e.g.*, computing estimated power spent by slave and master in BLE. However, there is still a clear need for further optimization, *e.g.*, in tournament-style runs, we run each scenario sequentially, although we could also utilize multiprocessing to optimize the runtime.

In addition to the above-mentioned points, as our future work, we also could improve the fidelity of GEPARD, for example, by: (i) replacing the current static Poisson arrival process with a time-varying Poisson arrival process, to better represent the changes in the user arrival process throughout the day, and (ii) consider group context and group privacy negotiations as introduced by Alanezi *et al*. [29].

B. Results Discussion

The results presented in Section VI-B showcase the importance of carefully considering different components in the PA design. The existing works, discussed in Section II, commonly ignore the evaluation of the proposed PA design and the impact the proposed designs would have on the overall IoT infrastructure, focusing solely on performance in terms of the collected consent, if any. The results we present allow us to see that, although it is important for a PA to

have an effective negotiation algorithm, the underlying choice of network technologies and the deployment scenarios also play a critical role in the PA performance results. The effects of the latter two components greatly determine not only the longevity of operation of the resulting PA (which one expects) but also affect the capability for consent collection of the PA (which would be less obvious in some instances). These observations, consequently, led us to do a component impact analysis presented in Section VI-B4. From the results, we can observe that all of the components proposed in DUNE (see Section IV) affect the performance of the PA and therefore require careful consideration when looking for real-world deployment of a PA. When considering real-world deployment of a PA, the designers and researchers would have to consider the trade-off of various negotiation algorithms, network technologies and the deployment scenario.

C. Future Works

Throughout this work, we mention several points that can be extended as part of our future works. Firstly, in Section V-D1, we mention that the user privacy preference categories were taken from the work presented by Kumaraguru and Cranor [52]. However, in the future, we also could extend the PP using Dupree's work [61]. The main difference between these works is that they extend the user privacy types to five with two dimensions instead of one: Fundamentalists, Lazy Experts, Technicians, Amateurs, and Marginally Concerned. The two dimensions are knowledge and motivation, *e.g.*, a Lazy Expert is knowledgeable but low on motivation, while a Technician type would be average knowledge but high on motivation. Secondly, we assume that the IoT device always communicates the correct PP to the DS. This assumption, however, is unrealistic since, at best, the IoT device can attempt to estimate the user privacy preferences using opponent modelling techniques from the automated negotiations field or utilizing Machine Learning (ML)-based techniques. Thirdly, when discussing Equation 2 in Section V-D3, we mention the data quantity versus quality trade-off we rely on when calculating the utility. In the future, we can analyze the effects of this assumption. We also assume perfect knowledge of how long the user will be in the environment, which is unrealistic and can be replaced with the above-mentioned techniques, *e.g.*, ML-based techniques. Finally, in the Concession algorithm (see Section V-D3), we did not analyze the effects of value n on the PA performance, so it would be interesting to consider different values of n in various scenarios.

VIII. Conclusion

In this work, we presented a novel PA architecture framework called DUNE and showcased the benefits and usability of the DUNE with the help of a GEPARD simulator. We argue that the approach of the existing works to PA design doesn't account for critical components, such as network type and IoT deployment environment, that greatly affect the real-world performance of the proposed PAs. To this extent, we introduce a component-based way of designing PAs, where four main components are considered: (i) IoT Device, (ii) User, (iii) Network, and (iv) Environment. We show how PA designs proposed in existing works can be disassembled into the proposed components. We then implement the resulting components in our simulation environment called GEPARD. GEPARD is then used to run tournament-style experiments, where different combinations of components are combined into a PA, simulated in an IoT environment and evaluated. We present the results that show the effects that different components have on the overall PA performance. Specifically, from the component impact analysis, we observe that network type greatly affects the average user consent percentage and average user power consumption, while negotiation algorithms affect the average user utility that a PA can achieve. Finally, we conclude by discussing some of the limitations of GEPARD design and implementation, our results and future directions.

APPENDIX A
REAL-WORLD MEASUREMENTS

Incomplete. Omitted from the main text because of existing works that provide finer grained measurements and better analytical models for power and time consumption, compared to the measurements discussed in this section.

The process of validation in simulators, as defined by Sargent, R. in [62], is concerned with checking the accuracy of the simulated model's representation of the real system. In our work, to validate our proposed simulator, we followed the following steps:

1) Data Collection
2) Real-World Measurements
3) Simulation Execution
4) Comparison and Analysis
5) Calibration
6) Re-validation

We now discuss each step in detail.

A. Data Collection

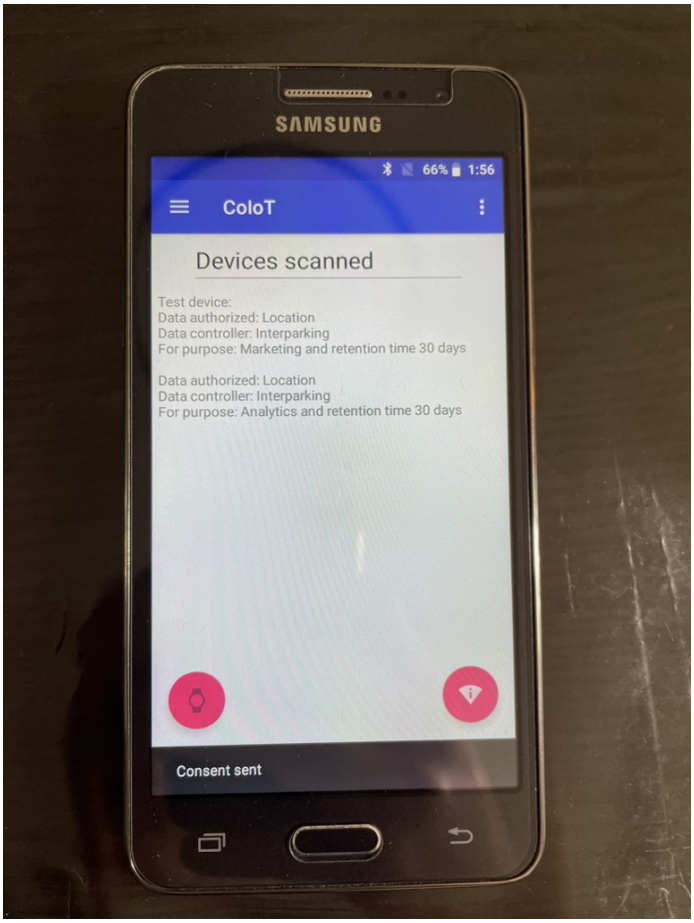
As discussed in Section V there is a variety of metrics that allow us to evaluate PA performance, which, consequently, we implemented as part of our proposed simulator (see Section **??**). One such metric is power consumption. We chose power consumption as the main means of simulator validation since (i) it is comparably straightforward to measure and compare against, (ii) it allows for scalable measurements and validation (as opposed to such metrics as timing, which would require prolonged study periods), and (iii) it has been done before, and there is a rich pool of data we can rely on to assure us of validity of our measurements, *e.g.*, [46] and [63]. Specifically, we gather current and voltage consumption data from real-world devices, *i.e.*, ESP32 and Samsung Galaxy Grand Prime, that run the implementations presented by the original authors of the simulated PA (see Section **??**). ESP32 was chosen as it is the device used by Cunche *et al.* in [10]. The main difference between our work and the Cunche *et al.* is the specific version of the board we've used. In particular, we used ESP32-DevKitM-1[4], while the Cunche *et al.* used

[4]https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/esp32/user-guide-devkitm-1.html

Espressif ESP32[5]. However, we didn't find any difference between the boards that would affect our conclusions. The usage of the same microcontroller unit allowed us to easily import the implementation codes provided by the authors[6] and run them on our device. Samsung Galaxy Grand Prime (Model: SM-G530W) was chosen as it runs the Android operating system and, similarly to ESP32, allowed us to import the application[7] developed by Cunche *et al*. in [10]. Figure 24(a) presents the screenshot of the CoIoT application on Samsung Galaxy Grand Prime. From the figure one can observe the data collection policies received from the ESP32 and, since, in this case, the policies received match the data subject's policies/preferences, the Personal Data Custodian issues consent (as per [10]). Figure 24(b) presents the privacy policies advertised by and consent reception on the Privacy Beacon.

There are several important things to note:

1) Cunche *et al*. implemented the proposed PA design using only BLE. As such, for measuring the effects of other network protocols and technologies, we will rely on our custom simplified implementations. Nonetheless, we argue that even such measurements will show the simulator's validity.
2) To make the Cunche *et al*. Android application code operational on our device, we had to do the following 3 major changes: (i) migrate the code to the ArduinoX[8], (ii) modify the code for higher SDK version (28), and (iii) modify the code to higher Gradle version (6.1.1).
3) There are other metrics used to evaluate PA designs in the simulation environment; however, as we will discuss in more detail below, for validating those metrics, we relied on the combination of one or more of the following approaches:
   - manufacturer's datasheets
   - software-based measurements
   - existing works
   - existing implementations



(a) CoIoT application on Samsung Galaxy Grand Prime.

```
Ready to receive consents!
Privacy Policy:
)
Location Interparking
        Marketing
)
Location Interparking
        Analytics
*********
Received a new consent:
Length:64
Value: ::Consent::{84:CF:BF:8A:99:21},9
```

(b) ESP32 privacy policy advertisement and consent reception.

Fig. 24: CoIoT privacy negotiation setup example.

B. Real-World Measurements

For real-world device power consumption measurements, we ran the simplified test scenarios on real devices and recorded their timing, current and voltage consumption data. We opted to measure the current and voltage consumption since, given that the time the devices operate is either known, can be measured or closely estimated, it is then straightforward to calculate the power and energy consumption of the device, *i.e.*, $Amps \times Volts = Watts$ and $Watts \times Hours = Watt\_Hours$, which then can be converted to either kWh or another metric of preference. The

[5]https://www.espressif.com/en/products/socs/esp32

[6]https://github.com/cunchem/BLE_Privacy_Beacon

[7]https://gitlab.inria.fr/vmorel/coiot

[8]https://developer.android.com/jetpack/androidx/migrate

test scenarios were defined based on the two major network technologies implemented in the simulator, *i.e.*, BLE and WiFi.

BLE. The BLE current and voltage consumption were measured for the following three modes:

- *Scanning mode*: In this mode, the link layer listens for advertising packets from other advertising devices. To implement this test scenario, we used *BLE_scan* code from the ESP32 sample code library[9].
- *Advertising mode*: In this mode, the device transmits advertising packets and listens to and responds to responses triggered due to advertising packets. To implement this test scenario, we used ESP32 library sample codes[10].
- *Connected mode*: This mode can be reached either from the "Initiating mode" or "Advertising mode". In this state, there are two roles the BLE device can assume: master or slave. When it enters this state from the "Initiating mode", it will assume the master role, but

[9]https://randomnerdtutorials.com/esp32-bluetooth-low-energy-ble-arduino-ide/

[10]https://medium.com/@jalltechlab/bluetooth-ble-advertising-with-arduino-esp32-sample-code-no-coding-part-2-972deb23b1c3

(a) ESP32-DevKitM-1 current consumption measurement setup. (b) Samsung smartphone current consumption measurement setup.

Fig. 25: Current consumption measurement setup.

when it enters from the “Advertising mode” (as is in our case), it will assume the slave role. In the master role, the link layer will communicate with the device in the slave role and defines the timings of transmissions. To implement this mode, we again make use of the ESP32 sample codes, specifically, BLE server implementation.

These state correspond to the BLE modes discussed in Section III-A1.

Note: For the measurement purposes we split scanning and advertising, since, for measurement purposes they behave differently. Additionally, we include the initial steps of the “Connection-Establishment mode”, *i.e.*, CONN_REQ message, in the “Connected mode”.

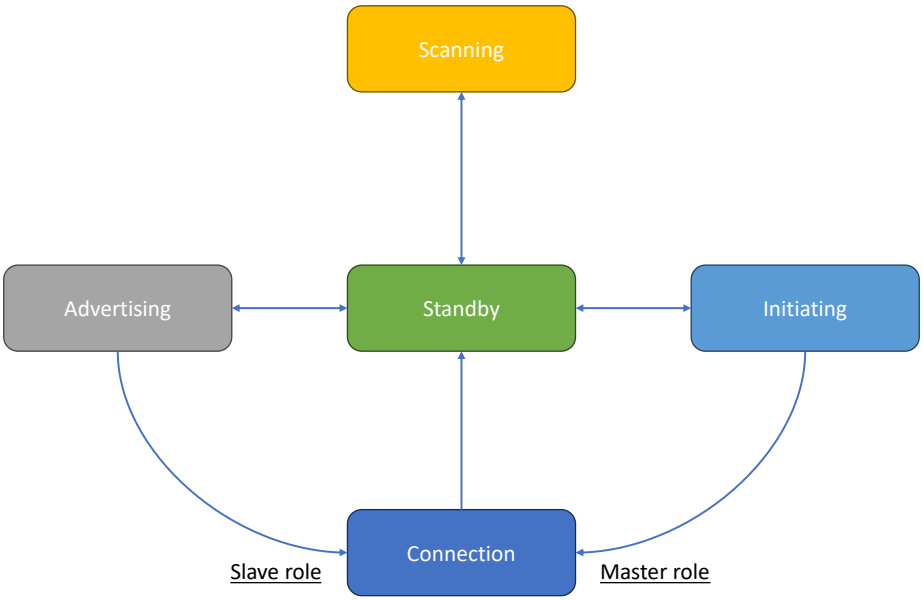


Fig. 26: BLE state diagram.

Figure 26 presents the BLE state diagram for the related states. It is important to note that different vendors may define new or different from the mentioned-above states, so it is important to account for these differences when implementing BLE across a variety of devices. For example, the BM70[11] radio defines an additional state (Shutdown), which is not defined in the BLE specification.

[11]https://microchipdeveloper.com/ble:bm70-overview

These specific states were chosen following the PA operation description provided by Cunche *et al*. in [10]. Specifically, at a high level, the following occurs on the IoT device and smartphone with respect to BLE link layer states:

1) *Scanning*: The device operates in the *Scanning mode*, actively listening for BLE advertisements from nearby devices, including the IoT device. When the smartphone receives the advertising packet from the IoT device, it can extract and process the privacy policies included in the packet.
2) *Advertising*: The IoT device operates in the *Advertising mode*, where it broadcasts its privacy policies using BLE advertising packets containing its DC privacy policy.
3) *Connected mode*: This mode can be reached directly from the Advertising mode. If the device determines that the advertised privacy policies are relevant or require further interaction, it transitions to the *Connected mode*. To transition, the device sends a connection request to the IoT device, initiating a communication link. Once the IoT device accepts the connection request, both devices enter the *Connected mode*. In this state, they exchange data, *e.g*., consent, using the Attribute Protocol (ATT).

   We split this mode into transmission (Tx) and reception (Rx) to allow for a fine grained device interaction implementation within the simulator. Note that for reception we expect only the values of the current to differ, since it is safe to assume that the time to transmit will equal the time to receive and the voltage will not differ.

TABLE VII: Real-world measurement results for ESP32 and Samsung Galaxy Grand Prime in different BLE modes.

BLE Mode	ESP32			Galaxy Grand Prime		
	Amperage (A)	Voltage (V)	Time (ms)	Amperage (A)	Voltage (V)	Time (ms)
Scanning	0.132	5.09	512	0.057	4.33	512
Advertising	0.128	5.09	20.184	0.047	4.33	1000.184
Connected (Tx)	0.128	5.09	41.706	0.05	4.33	9.206
Connected (Rx)	0.092	5.09	41.706	0.048	4.33	9.206

Table VII presents the measurement results for each mode. The current and voltage values were measured using a combination of a multimeter[12], USB digital tester[13], and, in the case

[12]https://www.etekcity.com/products/digital-multimeter-msr-r500

[13]https://www.pishop.ca/product/12-in-1-usb-digital-tester-adapter/

of Android smartphone, also Google's Battery Historian[14] logs. The provided measured values are averaged over 10 runs for each of the two measured devices, *i.e.*, multimeter and USB digital tester. For the timing data, we relied on manufacturers data sheets[15] and software-based measurements, *i.e.*, for Android measurements we rely on the Android Debug Bridge (adb) batterystats[16]. For reference, Figure 27 presents an example output of Battery Historian for a single scan and consent run of the CoIoT application, while Figure 28 presents an example output of the adb batterystats for a sample BLE application[17].

Application	fr.citi_lab.victor_morel.coiot
Version Name	1.0
Version Code	1
UID	10079
Device estimated power use	0.03%
Foreground	1 times over 10s 883ms
Total number of wakeup alarms	0

− Network Information:

Bluetooth idle time	0s
Bluetooth transfer time	50s 443ms total (49s 125ms receiving, 1s 318ms transmitting)
Bluetooth scanning	1 times for 59s 953ms total

Fig. 27: Example Battery Historian output for the CoIoT application.

```
Bluetooth total received: 0B, sent: 0B
Bluetooth scan time: 0ms
   Bluetooth Sleep time:  -1ms (-0.0%)
   Bluetooth Idle time:   22s 123ms (98.4%)
   Bluetooth Rx time:     361ms (1.6%)
   Bluetooth Tx time:     0ms (0.0%)
   Bluetooth Battery drain: 0.0487mAh
```

Fig. 28: Example batterystats output for a single BLE scan.

Scanning Mode Measurements. For BLE, in the Scanning mode, current and voltage consumption measurements on ESP32, we used 100 ms as the scan interval and 99 ms as the scan

[14]https://github.com/google/battery-historian

[15]https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/esp32/user-guide-devkitm-1.html

[16]https://developer.android.com/topic/performance/power/setup-battery-historian

[17]https://github.com/android/connectivity-samples?search=1

windows, as defined in the ESP32 library example code[18]. The scanning interval is the duration of time between two consecutive times that the scanner wakes up to receive the advertising messages. The scan window parameter defines how long to scan at each interval. For further explanation on the BLE scanning mode see Section III-A1. The total scan duration was set to 5 seconds, which gave us enough time to consistently measure the stationary current and voltage consumption over multiple scan intervals. The ESP32 CPU was set to the default 240 MHz frequency (alternative values were 160, 80 and 40 MHz, which would result in less power consumption, but also lower device performance). It is important to note that ESP32 MCUs are known to have higher than expected current consumption for BLE when compared to similar devices[19].

Additionally, BLE has two scanning modes: Active and Passive scanning. In passive scanning, the BLE radio just listens to other devices' advertisements. When one of these advertisements is detected, the radio reports to Bluetooth Framework the discovered device and all the advertisement packets. The advertisement contains information like discoverability and connectivity modes, TX power level, MAC address of the device and/or application data. In active scanning, the radio will request more information once an advertisement is received, and the advertiser will answer with information like friendly names and supported profiles. In our measurements, for both devices, active scanning mode was used.

For Samsung smartphone measurements, since we follow the CoIoT implementation, we used the default BLE scan settings, which, based on the Android documentation[20] and source code[21] found online result in 5120 ms interval and 512 ms window. *Note*: the Android 9.0.0 version is used since it matches the SDK 28 version used by our modernized version of CoIoT project.

To calculate the BLE scanning mode power consumption on the Samsung smartphone using Battery Historian and the adb batterystats, we ran Google's provided Android BluetoothLeGatt Sample project[22] and scanned the surrounding devices. In this way, we measured 0.057 A current

[18]Obtainable from the following Arduino board manager URLs: http://arduino.esp8266.com/stable/package_esp8266com_index.json, https://espressif.github.io/arduino-esp32/package_esp32_index.json and https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json

[19]https://electronics.stackexchange.com/questions/615765/why-is-esp32-ble-so-power-hungry

[20]https://developer.android.com/reference/android/bluetooth/le/ScanSettings

[21]http://androidxref.com/9.0.0_r3/xref/packages/apps/Bluetooth/src/com/android/bluetooth/gatt/ScanManager.java

[22]https://github.com/android/connectivity-samples?search=1

consumption at 4.33 V for BLE.

In addition, we also leveraged the adb tool to confirm our measurements and extract power profile information from the Samsung smartphone system files. Specifically, we retrieved the power profile data from the device using the following steps:

- We utilized the `adb` utility to pull the */system/vendor/overlay/framework-res_auto_generated_vendor.apk* file from the device to our local machine.
- The extracted APK was subsequently decompiled using an online Java decompiler[23]. Through this process, we gained access to the *power_profile.xml* file located within the APK's resources. This XML file contains valuable power consumption values associated with various device functionalities, including BLE (Bluetooth Low Energy) and WiFi. Refer to Figure 29 for a graphical representation of the extracted power profile data.

The power profile data within the *power_profile.xml* file expresses current consumption values in milliamps (mA) of the current draw at a nominal voltage[24].

framework-res__auto_generated_rro.apk › resources › res › xml › power_profile.xml

```
<?xml version="1.0" encoding="utf-8"?>
<device xmlns:android="http://schemas.android.com/apk/res/android" name="Android">
    <item name="none">0
    </item>
    <item name="screen.on">87
    </item>
    <item name="screen.full">187
    </item>
    <item name="bluetooth.controller.rx">57
    </item>
    <item name="bluetooth.controller.tx">57
    </item>
    <item name="bluetooth.controller.idle">2
    </item>
```

Fig. 29: Samsung Galaxy Grand Prime power profile.

As we can observe, the 0.057 A current consumption for the BLE controller receive operation matches with what we measured.

[23]http://www.javadecompilers.com

[24]https://source.android.com/docs/core/power

For ESP32, since there is no timing information provided by Cunche *et al*. and there is no global consensus on what the default values should be, we matched the values specified for the Android OS, *i.e*., 5120 ms interval and 512 ms window.

Note: We recognize the limitations of our measurement approaches, *i.e*., the precision and accuracy of such measurements; however, we argue that these are sufficient for the purposes of creating a realistic PA simulation baseline, and, if required, can be adjusted to the needs of the PA designer or other simulator users. One major limitation of our work is the lack of accurate real-world measurements on the Android smartphone, past what is possible to observe with a multimeter, USB digital tester, software-based approaches and manufacturer-provided power profile. This limitation results in closely positioned voltage and current values for single BLE packet transmission between different BLE states.

Advertising Mode Measurements. We employ similar methods to measure the BLE Advertising current and voltage consumption values. Specifically, on ESP32, we ran the example library code modified to send out advertisement packets with the minimum allowable value of 0x20 units (each unit is 0.625 ms) or 20 ms[25].

To calculate the full timing information of the BLE advertising, *i.e*., including packet transfer, on the ESP32, we calculated the packet sizes for the advertising and scan response data based on information extracted from the provided code and the BLE library source code[26].

The assigned device name in the example code is “Long name works now” with a length of 19 bytes.

The BLE advertising data includes the following fields:

- Flags field: 2 bytes (ESP_BLE_ADV_FLAG_GEN_DISC and ESP_BLE_ADV_FLAG_BREDR_NOT_SPT)
- Appearance field: 2 bytes
- Device Name field: 19 bytes
- Manufacturer data: 0 bytes
- Service UUIDs: 0 bytes

The resulting total advertising data size is:

$$2 + 2 + 19 + 0 + 0 = 23 \text{ bytes}$$

[25]https://www.bluetooth.com/specifications/specs/core-specification-5-3/

[26]https://github.com/espressif/arduino-esp32/blob/master/libraries/BLE/src/BLEAdvertising.cpp

From this information, we proceed to estimate the duration of a single advertisement event for ESP32 BLE. We also account for the advertising interval since, in the most common scenarios, the device will advertise multiple times. The advertising interval is governed by the `adv_int_min` parameter, which is set to 0x20 (32 in decimal) units in the example code. This interval determines how frequently the advertising event is triggered.

To estimate the duration of a single advertisement event, we use the following formula:

$$\text{Duration} = \frac{\text{Packet Size} \times 8 \times \text{Number of Packets}}{\text{Data Rate}} + \text{Advertising Interval Units} \times \text{Advertising Interval} \tag{4}$$

where:

- Packet Size = 23 bytes (total advertising data size)
- Data Rate = BLE advertising data rate (specified in bps)
- Number of Packets = 1 (single advertisement event)
- Advertising Interval = 32 units
- Advertising Interval Units = 0.625 ms (BLE advertising interval unit)

Assuming a typical BLE advertising data rate of 1 Mbps (megabits per second):

Calculating the duration:

$$\text{Duration} = \frac{23 \times 8 \times 1}{1 \times 10^3} + 32 \times 0.625 = 20.184\,\text{ms}$$

Therefore, with a `adv_min_interval` of 0x20 (32 in decimal), a BLE advertising data rate of 1 Mbps, and a single advertisement event, we estimate that a single advertisement event for ESP32 lasts approximately 20.184 ms, with advertising interval contributing the majority of the time, *i.e.*, 20 ms.

The measured current and voltage consumption were 0.128 A and 5.09 V, respectively.

For Android smartphones, the default advertising mode is ADVERTISE_MODE_LOW_POWER[27], which results in 1 s advertising interval. Equation 4 is also applicable to calculate the timing information for BLE advertising packet transmissions in Samsung smartphone. Consequently, since we assume that the smartphone transmits the same advertising data, a single advertisement event for an Android smartphone lasts approximately 1000.184 ms.

For the current and voltage consumption measurements on the Samsung smartphone, we used Google's BLE Advertisement sample codes [28]. We measured 0.047 A at 4.33 V.

[27] https://developer.android.com/reference/android/bluetooth/le/AdvertiseSettings.html

[28] https://github.com/android/connectivity-samples/tree/main/BluetoothAdvertisements

Note: Similar to other measurements, these results rely on specific assumptions and will not be applicable to all cases. However, the assumptions can be easily adjusted within the simulator's source code.

Connected Mode Measurements (Tx). We repeat the same steps as above to measure and estimate the necessary values.

We make use of the ESP32 BLE notify example code[29] to implement the BLE connection timeline as presented by A. Del Campo *et al*. in [64]. In their work, the authors describe the BLE connection timeline as follows:

1) After receiving the advertisement message, the central device (in our case, the Samsung smartphone) waits for the inter-frame time of 150 μs and, then, sends the connection request packet (CONN_REQ) on the same channel. The CONN_REQ packet is usually around 34 bytes[30] and takes around 272 μs to transfer.
2) After the CONN_REQ, the central, acting now as master, and the advertiser, acting now as a slave, wait for a minimum of 1.25 ms before continuing to the data transfer.

For described packet flow see Figure 1(b).

We assume that for the connection state (Tx) measurements, the devices exchange only once the privacy policies defined within the maximum BLE packet size limit, *i.e*., 23 bytes. Consequently, the total connection timing for both devices can be calculated as follows: For ESP32:

CONN_REQ time: 272 μs

Connection interval: 40 ms

Initial wait time after CONN_REQ: 1.25 ms

Data transfer time per packet: $\frac{23 \text{ bytes}}{1{,}000{,}000 \text{ bps}}$ s

Total connection time for ESP32 starting from CONN_REQ: 0.272 ms + 1.25 ms + 40 ms + $\frac{23 \text{ bytes} * 8}{1{,}000 \text{ bpms}}$ ms = 41.706 ms

For Android device:

CONN_REQ time: 272 μs

Connection interval: 7.5 ms

Initial wait time after CONN_REQ: 1.25 ms

[29]http://www.openlabpro.com/guide/ble-notify-on-esp32-controller/

[30]https://novelbits.io/deep-dive-ble-packets-events/

Data transfer time per packet: $\frac{23 \text{ bytes}}{1{,}000{,}000 \text{ bps}}$ s

Total connection time for Android device: 0.272 ms + 1.25 ms + 7.5 ms + $\frac{23 \text{ bytes} * 8}{1{,}000 \text{ bpms}}$ ms = 9.206 ms

Consequently, for the ESP32 we have the connection interval of 0x20 units (each unit is 1.25 ms) or 40 ms. The resulting connection timing (including initial pairing/bonding) is 41.706 ms. Measured current and voltage consumption were 0.128 A and 5.09 V, respectively.

In the Samsung smartphone case, we had the minimum allowable connection interval of 7.5 ms[31]. The resulting timing (including initial pairing/bonding) was 9.206 ms. The measured current and voltage consumption were 0.05 A and 4.33 V, respectively.

Connected Mode Measurements (Rx). For Rx case we repeat the same steps and code as above to measure and estimate the current consumption values.

In the ESP32 case we measured 0.92 A, while in the Samsung smartphone case we measured 0.048 A.

WiFi. Concurring with discussions presented by Morel *et al*. in [6], we assume that the IoT device communicates directly with the PA. In this scenario, the IoT device acts as an access point and can be connected to using any device with WiFi capabilities without connecting to your router (see Figure 30).

Similar to the BLE measurements, we made use of the ESP32 and Samsung Galaxy Grand Prime devices. For the measurement scenario we assume that the EPS32 acts as an access point and sends out privacy policies to any device connecting to it. At the same time Samsung smartphone scans the surrounding environment for WiFi access points that advertise privacy policies and connects to the them to negotiate the aforementioned policies.

Note that the ESP32 and Samsung Galaxy Grand Prime support IEEE 802.11 b/g/n standards with the latter not being able to use 5 GHz[32].

Consequently, we are interested in measuring the timing, current and voltage consumption during the following device modes:

1) *Scan mode*: in this mode, the Samsung Galaxy Grand Prime smartphone actively scans its surrounding environment for available WiFi access points that are advertising privacy

[31]https://punchthrough.com/maximizing-ble-throughput-on-ios-and-android/

[32]ESP32: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/wifi.html

Samsung Galaxy Grand Prime: https://www.samsung.com/ph/smartphones/others/galaxy-grand-prime-white-8gb-sm-g530hzwdxtc/

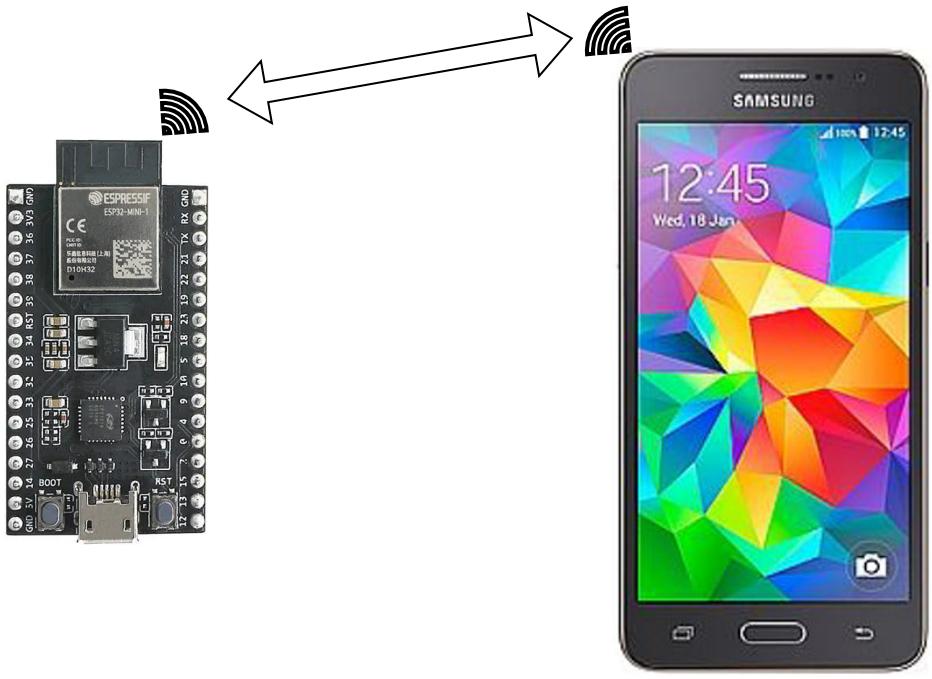


Fig. 30: WiFi measurement setup.

policies. It essentially listens for signals from nearby WiFi networks. When it detects a network broadcasting such policies, it collects information about it. We split this mode into two sub-modes: AP and STA. AP mode has been introduced before, while in the STA mode the devices is the connecting client, or the **STA**tion. *Note*: in some literature STA can also refer to an AP, but we opted to refer strictly to the client device.

We implement the STA mode for the Samsung smartphone using the Android developer WiFi scan guide.[33] At the same time, the AP mode is done utilizing the WiFi hotspot functionality, built into the Android OS. For the ESP32, we implement the STA and AP modes using WiFiScan and WiFiAP example sketches from the Arduino ESP32 library[34].

2) *Connection-Establishment mode*: Once the Samsung smartphone identifies a WiFi access point that advertises privacy policies and matches its criteria, it enters the connection mode. In this state, the device initiates a connection with the selected access point. This involves a negotiation process where the device and the access point communicate to establish a secure and policy-compliant connection. This mode is implemented using Android developer WiFi

[33]https://developer.android.com/guide/topics/connectivity/wifi-scan

[34]https://github.com/espressif/arduino-esp32/tree/master

P2P example codes.[35] For the ESP32 WiFi implementation we make use of the Arduino WiFi library and it's example codes.[36]

3) *Transmit mode*: this mode involves both the Samsung Galaxy Grand Prime smartphone and the ESP32 device actively sending data or signals to each other. The Samsung smartphone, in this mode, sends requests, queries, consent or user-generated data to the connected WiFi access point, *i.e.*, EP32. The ESP32, acting as an access point, transmits privacy policies and handles negotiation with the Samsung smartphone. Similar to the other modes, measuring timing, current, and voltage consumption is done using Android developer WiFi P2P and Arduino WiFi library example codes, for Samsung device and ESP32, respectively. Specifically, we transmit an example XML-encoded privacy policy as presented in Section V and observe the current consumption on the respective device.
4) *Receive mode*: in Receive Mode, both the Samsung Galaxy Grand Prime smartphone and the ESP32 device are actively listening and waiting to receive data or signals from their respective connecting parties within the network setup. The devices in this mode are essentially in a state of readiness to accept incoming data or signals. Measuring timing, current, and voltage consumption is done using Android developer WiFi P2P and Arduino WiFi library example codes, for Samsung device and ESP32, respectively. Specifically, we transmit an example XML-encoded privacy policy as presented in Section V and observe the current consumption on the respective device.

Table VIII presents the measurement results for each of the WiFi modes.

Scan Mode Measurements. To extract the scan timing information of the Samsung smartphone, when it operates as an STA, we utilize the method presented by Goovaerts *et al*. [65]. Specifically, we extracted */system/etc/firmware/wlan/prima/WCNSS_qcom_cfg.ini* file from the phone using adb. From there we found that the two values defining the dwell time are specified as follows:

- gActiveMaxChannelTime$= 80$
- gActiveMinChannelTime$= 60$

which are specified in ms. The way these values are used in practice is as follows: the minimum time the scan dwells on each channel is *gActiveMinChannelTime* ms. If no AP is found during this time frame, the scan switches to the next channel. Otherwise, the scan dwells on the channel

[35]https://developer.android.com/guide/topics/connectivity/wifip2p

[36]https://randomnerdtutorials.com/esp32-useful-wi-fi-functions-arduino

TABLE VIII: Real-world measurement results for ESP32 and Samsung Galaxy Grand Prime in different WiFi modes.

WiFi Mode	ESP32			Galaxy Grand Prime		
	Amperage (A)	Voltage (V)	Time (ms)	Amperage (A)	Voltage (V)	Time (ms)
Scan Mode (AP)	0.162	5.09	650	0.250	4.33	520
Scan Mode (STA)	0.144	5.09	650	0.060	4.33	520
Connection-Establishment Mode (AP)	0.192	5.09	0.2548	0.250	4.33	0.0655
Connection-Establishment Mode (STA)	0.188	5.09	0.238	0.250	4.33	0.061
Transmit Mode	0.212	5.09	0.1	0.250	4.33	0.028
Receive Mode	0.168	5.09	0.1	0.060	4.33	0.028

for *gActiveMaxChannelTime* ms. When Samsung smartphone is being used as an AP, the default beacon interval is 100 time units or 102.4 ms[37] [66].

For the ESP32, when it acts as an AP, the default library specifies beacon transmit interval of 100 time units or 102.4 ms[38]. When ESP32 acts as a client, we modified the *WiFiScan.cpp*[39] library file to enable passive scanning by default and set maximum dwell time of 100 ms. The value of 100 ms is to align with findings made by Goovaerts *et al.* [65] on the minimum optimal dwell time for passive scanning.

Consequently, while it is impossible to exactly determine the timing for WiFi scan mode, since it will depend on the scan timing and scan channel distance between AP and STA, we can still estimate an average time it may take to scan the channel and detect the AP. Using averages is sufficient for the purposes of our work, and may be adjusted, if necessary to the worst or best case scenarios.

Note: for the results in the Table VIII, we match the STA measured timing, with its counterpart, *i.e.*, Samsung smartphone STA scanning and detecting ESP32 AP and vice versa. You can observe that, in the Table, the timing values are exactly the same between AP and STA rows, since in both modes, we assume that the device operates in the respective mode only for as long as the

[37]https://android.googlesource.com/platform/external/wpa_supplicant_8/+/0716c12e57090ce9904fb5948da1285fc36c1fe4/wpa_supplicant/wpa_supplicant.conf

[38]https://github.com/espressif/arduino-esp32/blob/master/libraries/WiFi/src/WiFiAP.cpp

[39]https://github.com/espressif/arduino-esp32/blob/master/libraries/WiFi/src/WiFiScan.cpp

first beacon packet is not detected.

To calculate the average scan times for the devices we can use the following:

- Assume that AP chooses between which WiFi channel to send beacon on uniformly.
- Given that we know the total number of WiFi channels that the AP can be on, we then can find how many channels a STA needs to scan to find an AP with 50% probability using the following equation:

$$\frac{n}{\text{Total number of channels}} > 0.5 \tag{5}$$

, which results in 7 channels that need to be scanned on average before finding an AP on one of them.

- Now, given that it takes dwell time on each channel for a STA to move on to a different channel, then we can calculate the total time it takes for each device in STA to find an AP using the equation below:

$$T_{avg,scan} = (n-1) * \text{Dwell time} + \frac{\text{Dwell time}}{2} \tag{6}$$

, where we multiple the dwell time by the number of channels that need to be scanned for the whole duration of the dwell time plus the half of the dwell time for the last channel to detect the AP. Again, we assume that on average it takes half a dwell time to detect the AP.

The above provided description results in ESP32 taking 650 ms in scanning mode, while Samsung smartphone takes 520 ms, which aligns with the expected timing information for a passive WiFi scan[40]. Note that for Samsung smartphone we work with the maximum dwell time for calculating the timing information.

For the Samsung smartphone AP and STA we rely on the datasheet since there is no straightforward way to measure the power consumption without accessing the smartphone internals. Specifically, we tried to measure power consumption using the previously mentioned methods, however, the hotspot doesn't show up in any of the logs. Even attempting to circumvent by utilizing a AP Hotspot Manager application[41], didn't lead to any results and the logs only reflected the power consumed on screen time. Consequently, we use the datasheet for BCM4330LB, which

[40]https://microchipdeveloper.com/wifi:connecting, https://www.intuitibits.com/2017/08/11/understanding-scan-modes-wifiexplorerpro/

[41]https://github.com/vijaypatidar/AndroidWifiManager?search=1

is the WiFi module used in Samsung Galaxy Grand Prime[42]. The current consumption for WiFi data transmission is 250 μA. It is worth noting, that reviewing the datasheet also permits us to further validate our measurements, since our results, while not precisely, but align with the mentioned current consumption values.

For the ESP32 current and voltage consumption we relied on the same approach as was used for BLE measurements.

Connection-Establishment Mode Measurements. The existing works measuring the authentication, association and the 4-way handshake timings have shown that these timings are highly hardware dependant. For example, Yi *et al*. in [67] have shown that the implementation platform may introduce 10s and 100s of milliseconds of difference in WiFi 4-way handshake and authentication processes, respectively. Consequently, although we introduce some of the values based on the hardware at our disposal, it is advisable to adjust the values to the values of the simulated hardware.

In this part, we calculate the total time for the authentication, association and the 4-way handshake only as the sum of transmission times and omit other components, *e.g*., the generating time for both session keys and MICs. To this extent we use Wireshark[43] running on a 3rd device to capture and analyze the transmissions between ESP32 and the Samsung smartphone. Figure 31 shows the capture of the authentication, association and 4-way handshake process between the ESP32 (AP) and Samsung smartphone (STA) and vice versa.

Note: Although it would be fairly straightforward to rely on Wireshark provided timestamps to calculate the timing information, and, in our case, reasonable due to close proximity between the devices ($\geq$ 2 cm), in general cases, the timestamp reflects the packet delivery time to the measuring device and not the time it reaches the destination. Consequently, we will still calculate the timing information independently and use the provided timestamps only as a reference point.

From Figure 31 we can observe that in total for authentication, association and the 4-way handshake, ESP32 and Samsung smartphone, need to exchange the following number of bytes:

- ESP32 (AP): 637 bytes.
- Samsung Galaxy Grand Prime (AP): 592 bytes.
- ESP32 (STA): 595 bytes.

[42]https://pdf1.alldatasheet.com/datasheet-pdf/view/1131870/BOARDCOM/BCM4330.html

[43]https://www.wireshark.org

7082	21.258381	SamsungE_6b:e5:59	Espressi_ef:16:f1	802.11	123	Probe Request, SN=377, FN=0, Flags=........C, SSID="IoT Test Env"
7084	21.260525	Espressi_ef:16:f1	SamsungE_6b:e5:59	802.11	206	Probe Response, SN=570, FN=0, Flags=........C, BI=100, SSID="IoT Test Env"
7091	21.286537	SamsungE_6b:e5:59	Espressi_ef:16:f1	802.11	59	Authentication, SN=378, FN=0, Flags=........C
7093	21.287498	Espressi_ef:16:f1	SamsungE_6b:e5:59	802.11	59	Authentication, SN=2046, FN=0, Flags=........C
7095	21.289377	SamsungE_6b:e5:59	Espressi_ef:16:f1	802.11	146	Association Request, SN=379, FN=0, Flags=........C, SSID="IoT Test Env"
7098	21.292509	Espressi_ef:16:f1	SamsungE_6b:e5:59	802.11	156	Association Response, SN=2047, FN=0, Flags=........C
7100	21.293407	Espressi_ef:16:f1	SamsungE_6b:e5:59	EAPOL	183	Key (Message 1 of 4)
7120	21.346036	SamsungE_6b:e5:59	Espressi_ef:16:f1	EAPOL	184	Key (Message 2 of 4)
7122	21.348615	Espressi_ef:16:f1	SamsungE_6b:e5:59	EAPOL	239	Key (Message 3 of 4)
7124	21.351265	SamsungE_6b:e5:59	Espressi_ef:16:f1	EAPOL	162	Key (Message 4 of 4)

(a) WiFi authentication, association and 4-way handshake Wireshark capture between ESP32 (AP with SSID = "IoT Test Env") and Samsung smartphone (STA).

125…	42.059010	SamsungE_6b:e5:59	Espressi_ef:16:f0	802.11	203	Probe Response, SN=230, FN=0, Flags=........C, BI=100, SSID="AndroidAP_7077"
128…	43.082332	Espressi_ef:16:f0	SamsungE_6b:e5:59	802.11	59	Authentication, SN=2248, FN=0, Flags=........C
128…	43.084240	SamsungE_6b:e5:59	Espressi_ef:16:f0	802.11	59	Authentication, SN=231, FN=0, Flags=........C
128…	43.085900	Espressi_ef:16:f0	SamsungE_6b:e5:59	802.11	148	Association Request, SN=2249, FN=0, Flags=........C, SSID="AndroidAP_7077"
128…	43.098859	SamsungE_6b:e5:59	Espressi_ef:16:f0	802.11	153	Association Response, SN=232, FN=0, Flags=........C
128…	43.108416	SamsungE_6b:e5:59	Espressi_ef:16:f0	EAPOL	162	Key (Message 1 of 4)
128…	43.110321	Espressi_ef:16:f0	SamsungE_6b:e5:59	EAPOL	205	Key (Message 2 of 4)
128…	43.115255	SamsungE_6b:e5:59	Espressi_ef:16:f0	EAPOL	218	Key (Message 3 of 4)
128…	43.117118	Espressi_ef:16:f0	SamsungE_6b:e5:59	EAPOL	183	Key (Message 4 of 4)

(b) WiFi authentication, association and 4-way handshake Wireshark capture between ESP32 (STA) and Samsung smartphone (AP with SSID = "AndroidAP_7077").

Fig. 31: WiFi authentication, association and 4-way handshake Wireshark capture example.

- Samsung Galaxy Grand Prime (STA): 551 bytes.

It is important to note that these bytes are transmitted over 4 packets, meaning that the timing will be the sum of transmitting the 4 packets. Given the average reported throughput of WiFi on ESP32 is 20 Mbps[44] and 72.2 Mbps on Samsung's BCM4330 WiFi modules, we can calculate how long it will take to transmit each of the frames as follows:

- ESP32 (AP): $((59*8)/(20*10^6))+((156*8)/(20*10^6))+((183*8)/(20*10^6))+((239*8)/(20*10^6))=0.0002548$ seconds or 0.2548 ms
- Samsung Galaxy Grand Prime (AP): $((59*8)/(72.2*10^6))+((153*8)/(72.2*10^6))+((162*8)/(72.2*10^6))+((218*8)/(72.2*10^6))=0.0000655$ seconds or 0.0655 ms
- ESP32 (STA): $((59*8)/(20*10^6))+((148*8)/(20*10^6))+((205*8)/(20*10^6))+((183*8)/(20*10^6))=0.000238$ seconds or 0.238 ms
- Samsung Galaxy Grand Prime (STA): $((59*8)/(72.2*10^6))+((146*8)/(72.2*10^6))+((184*8)/(72.2*10^6))+((162*8)/(72.2*10^6))=0.000061$ seconds or 0.061 ms

For current and voltage consumption measurements we follow the same approach as described above, *i.e.*, combination of datasheet provided values and measurements. For Samsung smartphone we assume the transmission power consumption, as it is the highest/worst case scenario.

Transmit Mode Measurements.

[44]https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/wifi.html

For timing information we assume that the device transmits a simple HTTP response header:

Listing 7: HTTP response header.

```
HTTP/1.1 200 OK
Content-Type: text/xml
```

which is 39 bytes long. Alongside the the response header it will transmit an XML-encoded privacy policy which is 217 bytes (as presented in Section V). Consequently, given the assumed WiFi throughput discussed above, we can calculate the timing information as follows:

- ESP32: $((39 + 217) * 8)/(20 * 10^6) = 0.0001$ seconds or 0.1 ms
- Samsung Galaxy Grand Prime: $((39+217)*8)/(72.2*10^6) = 0.000028$ seconds or 0.028 ms

For current and voltage consumption measurements we follow the same approach as described above, *i.e.*, combination of datasheet provided values and measurements. For Samsung smartphone we assume the transmission power consumption, as it is the highest/worst case scenario.

Receive Mode Measurements.

For Receive Mode we follow similar approach to the Transmit Mode.

C. Simulation Execution

To compare the real-world measurements with the simulator-produced results, we ran the same test scenarios, *i.e.*, two device communication in different modes using BLE and WiFi, on the simulator as we did on the real-world devices while recording the timing, current and voltage consumption data. These test scenarios were incorporated into testing suite of the simulator, are part of the simulator's verification (for more details on verification see Section **??**).

From previous subsection we know that to transmit X bytes it takes X time and consumes X amps at X voltage. From here we can code the simulator to account for at as follows...

BLE.

D. Comparison and Analysis

We then compared the power consumption data obtained from the simulator with the data recorded from real devices. The goal was to identify any discrepancies that cannot be accounted for by our assumptions and simplifications in the simulator's model, *e.g.*, we do not account for idle and sleep power consumptions.

E. Calibration

We then calibrated the simulator's power model or adjusted its parameters to better align with real-world behaviour.

F. Re-validation

Finally, we repeated the validation process after calibration to ensure that the simulator's power consumption behaviour now reasonably closely matches the real devices.

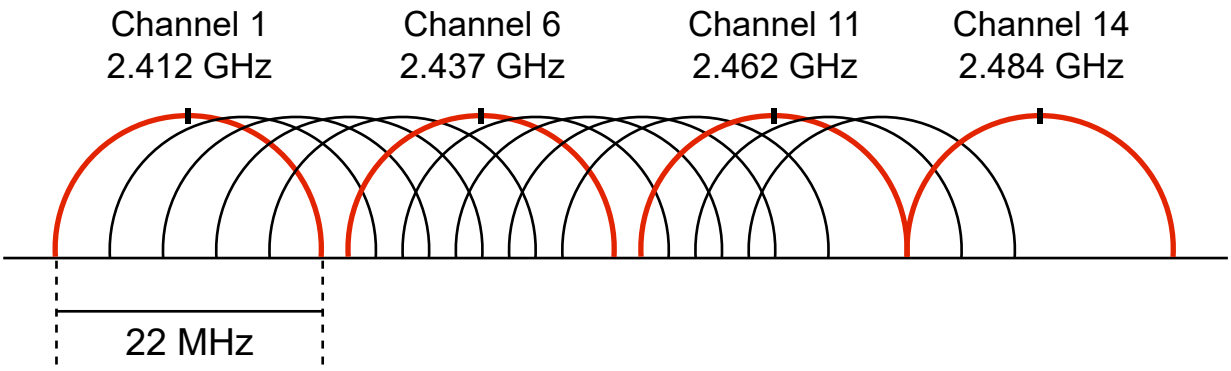
APPENDIX B
WIFI PROTOCOL FUNDAMENTALS

Omitted because could not find a good measurement paper and analytical model for power and time consumption. Replaced with ZigBee.

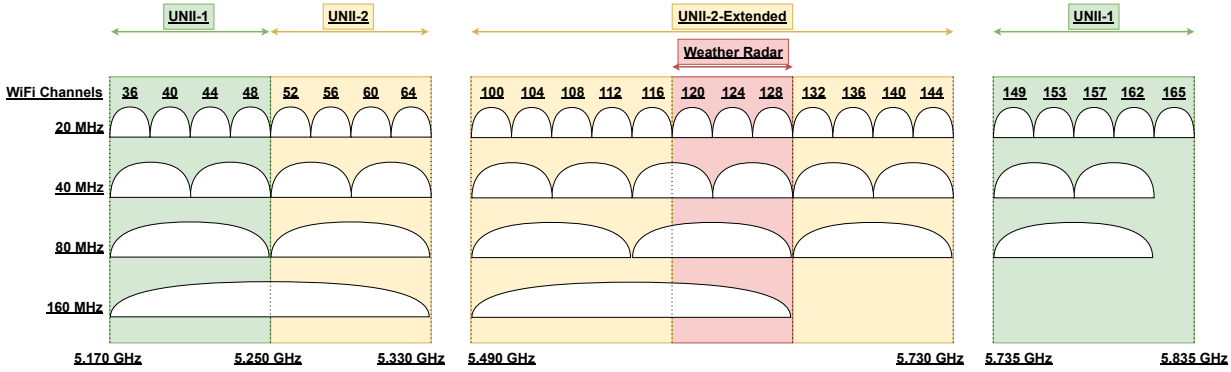
In this section, we provide an overview of the WiFi, also known as Wireless Fidelity, protocol's main features relevant to our proposed simulator. For a comprehensive understanding of the WiFi protocol we refer to existing works in the literature, *e.g.*, review provided by Qureshi and Asghar [68]. WiFi is a widely used name for devices that follow IEEE 802.11 set of standards for wireless LAN communication and operate in various frequency bands, commonly 2.4 GHz and 5 GHz, with varying channel widths and modulation schemes. In the Figure 33 we present the channel centre frequency distribution for 2.4 GHz and 5 GHz. As can be seen from the figure, in the 2.4 GHz band, there are 14 channels, each 20 MHz wide with a 2 MHz gap as a guard band between channels. In the 5 GHz band, each channel is also 20 MHz wide. However, channels can be combined to form 40, 80, or even 160 MHz channels, allowing for incredibly high bandwidths. The radio bands for 5 GHz is defined by the United States Federal Communications Commission as Unlicensed National Information Infrastructure or UNII and is part of the radio frequency spectrum used by WLAN devices. As part of the definition there are parts in the 5 GHz frequency band that are not usable due to various factors, *e.g.*, reserved for weather radars. It's important to note that in WiFi the neighbouring channels overlap to some extent but it is possible to find a set of non-overlapping channels, *e.g.*, in 2.4 GHz, channels 1, 6 and 11 do not overlap.

At a high level, two WiFi-enabled devices attempting to establish a connection will follow the following set of steps:

1) *Network Discovery Mode*: There are two main scanning modes WiFi client device can use to discover existing access points (AP)s: passive and active. In passive scanning mode, the client radio listens on each channel for beacons sent periodically by an AP. In an active scanning mode, the client radio transmits a probe request and listens for a probe response from an AP. For our scenario we assume that the client operates using passive scanning and AP periodically sends beacons.
2) *Association and Authentication Mode*: When a client device finds an AP it wants to connect to, it initiates an association process. In this process, the client device sends an association



(a) WiFi channels distribution in 2.4 GHz frequency band.



(b) WiFi channels distribution in 5 GHz frequency band.

Fig. 32: Packet flow in BLE protocol.

request to the AP. If the network is also secured, the devices will engage in an authentication process to prove their credentials. Upon successful completion of these processes, the client device becomes associated with the network.

3) *Data Transfer Mode*: After successfully associating with the network and obtaining an IP (Internet Protocol) address, both devices transition into the ”connected” state. In this state, they can exchange data packets with each other or with devices on the internet. WiFi networks support various data rates, depending on the signal strength and interference, ranging from a few megabits per second (Mbps) to several gigabits per second (Gbps) for newer standards like WiFi 6 and WiFi 6E.

For the purpose of our simulator and to align the WiFi protocol model with existing PA network designs, we distinguish between the following WiFi simulation modes:

1) *Scan Mode*: In this mode, the Samsung Galaxy Grand Prime smartphone passively scans [45] its surrounding environment for available WiFi AP that are advertising as privacy policy providers. It essentially listens for beacon packets from nearby WiFi networks. When it detects an AP broadcasting such functionality, it collects information about it. We implement this mode using the Android developer WiFi scan guide[46]. See Figure 33(a) for a high-level representation of the scan mode. In the figure, dwell time is the amount of time a device uses a particular channel.
2) *Connection-Establishment Mode*: Once the Samsung smartphone identifies a WiFi AP that it wants to connect to, it enters the connection mode. In this state, the device initiates a connection with the selected AP. This involves a negotiation process where the device and the AP communicate to establish, usually, a secure connection. This mode commonly consists of 3 major steps: authentication, association, and 4-way handshake.

   Authentication will be determined by the security standard utilized, common ones being WPA (WiFi Protected Access), WPA2 and WPA3. Although the WPA3 is the latest standard and is mandatory for the WiFi certified devices after July 2020, for our work we focus on WPA2, since that is the standard that is commonly used in the existing literature and by the devices in our experimental setup. At this stage, in WPA2 case, the STA and AP both possess a pre-shared key (PSK). The STA initiates a connection request to the AP and provides the PSK. The AP then validates the PSK and, given the validity of the key, accepts the connection. At a high level this is achieved by exchanging two Authentication frames.

   Association is achieved by exchanging Association Request and Association Response frames. Association Request frame includes a variety of information, *e.g.*, Listen Interval field which is used to indicate to the AP how often a client in power save mode wakes to listen to Beacon management frames. After acknowledging reception of the Association Request frame, the AP examine each field of the request and verifies that they all match its own parameters. If there is a mismatch, AP decides whether this difference is a blocking factor to the association. If the differences is not blocking, the AP take note of those differences and grants access to the network, indicating its own parameters in the Association Response frame. If the difference is blocking, then AP rejects the association. The general

[45]https://developer.android.com/guide/topics/connectivity/wifi-scan

[46]https://developer.android.com/guide/topics/connectivity/wifi-scan

frame flow for the Association is shown in Figure 33(c).

Finally, STA and AP execute the 4-way handshake (show in Figure **??**) to negotiate a fresh session key. This key will be used to encrypt and authenticate traffic data frames using data protection protocol, *e.g.*, in WPA2 it is CCM (counter with cipher block chaining message authentication code) mode protocol or CCMP (AES in counter mode for encryption and CBC MAC for integrity check and message authentication).

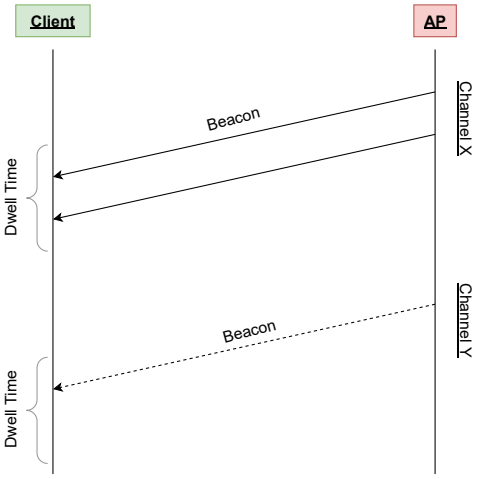
The Connection-Establishment mode is implemented using Android developer WiFi P2P example codes[47]. For the ESP32 WiFi implementation, we make use of the Arduino WiFi library and its example codes[48].

3) *Transmit Mode*: This mode involves both the Samsung Galaxy Grand Prime smartphone and the ESP32 device actively sending data to each other. The Samsung smartphone, in this mode, sends requests, queries, consent, or user-generated data to the connected WiFi AP, *i.e.*, ESP32. The ESP32, acting as an AP, transmits privacy policies and handles negotiation with the Samsung smartphone. Similar to the other modes, measuring timing, current, and voltage consumption is done using Android developer WiFi P2P and Arduino WiFi library example codes, for the Samsung device and ESP32, respectively. Figure 33(d) provides a high-level representation of a successful WiFi transmission. Specifically, we include RTS and CTS packets representing collision avoidance mechanism.
4) *Receive Mode*: In Receive Mode, both the Samsung Galaxy Grand Prime smartphone and the ESP32 device are actively listening and waiting to receive data from their respective connecting parties within the network setup. Measuring timing, current, and voltage consumption is done using Android developer WiFi P2P and Arduino WiFi library example codes, for the Samsung device and ESP32, respectively.

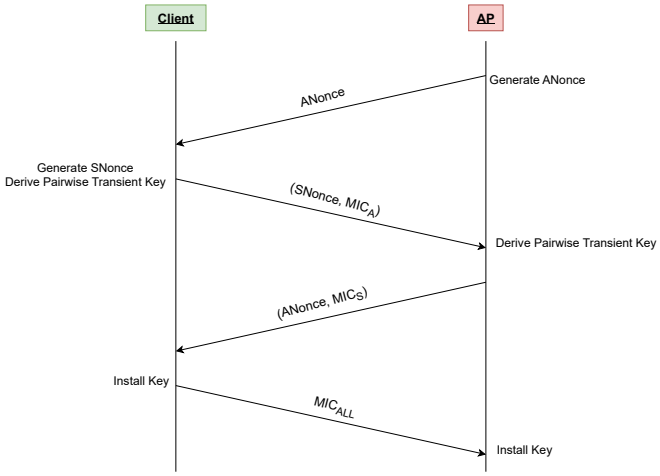
Note: It is important to note that over the years the 802.11 standard that WiFi is built upon has evolved and gone through various changes. Starting with 1997 standard that relied on the direct-sequence spread spectrum (DSSS) for modulation and provided maximum link rate of 1 Mbit/s, all the way to the most recent 802.11ax that uses orthogonal frequency-division multiple access (OFDMA) modulation and offers up to 1201 Mbit/s maximum link rate per spatial stream

[47]https://developer.android.com/guide/topics/connectivity/wifip2p

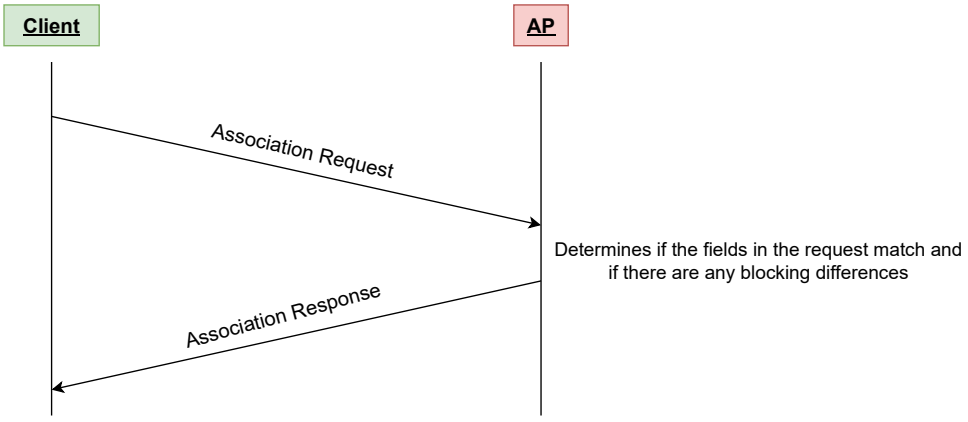
[48]https://randomnerdtutorials.com/esp32-useful-wi-fi-functions-arduino



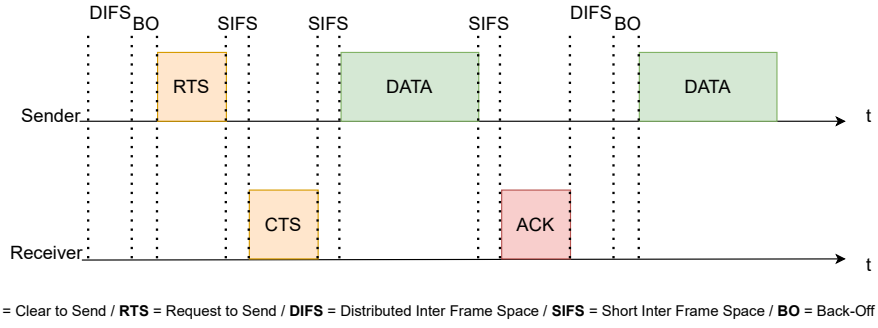
(a) High-level representation of the WiFi scan mode.



(b) High-level representation of the WiFi 4-way handshake process..



(c) High-level representation of the WiFi association process..



CTS = Clear to Send / **RTS** = Request to Send / **DIFS** = Distributed Inter Frame Space / **SIFS** = Short Inter Frame Space / **BO** = Back-Off count

(d) High-level representation of the WiFi transmission process.

Fig. 33: High-level representation of the main frame flows in the WiFi protocol.

for 160 MHz channels[49]. For the purposes of our work we assume a generic 802.11 WiFi implementation that follows the behaviour commonly described in existing literature of a similar nature, *e.g.*, guide to 802.11 presented by Gast M. [69] and work presented by Tilghman P. and Rosenbluth D. [70].

Determine which is used in the Samsung and ESP32?

[49]https://www.wi-fi.org/discover-wi-fi/wi-fi-certified-6

APPENDIX C
RESULTS

Opted to show only the main results or results of interest in the main part of the text.

1) Average User Consent: We measure the average user consent as a percentage of consent collected by an IoT device, averaged over all the runs. The main results are presented and discussed in Section VI-B1.

Fig. 34: Average percentage of consented users for different network technologies and scenarios under Cunche.

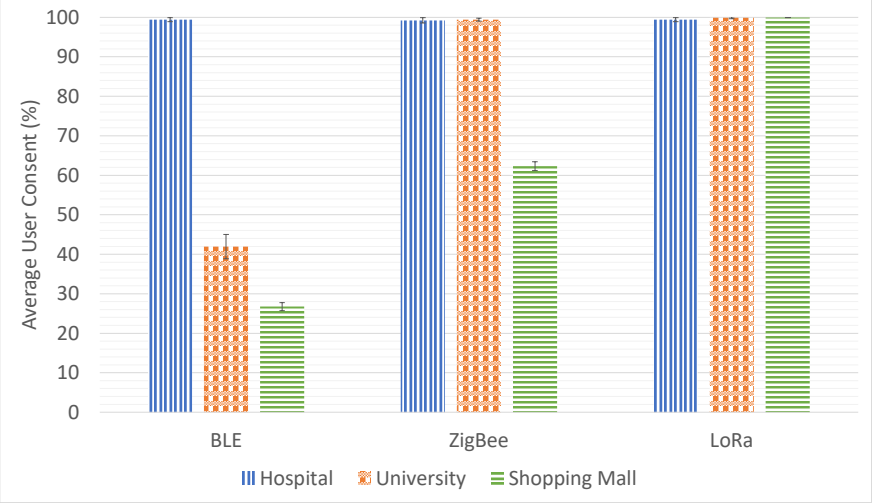


Fig. 35: Average percentage of consented users for different network technologies and scenarios under Concession.

2) Power Consumption (Users): In this metric we measured the combined power consumption of all user device, *e.g.*, smartphones and averaged it over the total number of runs. The main results are presented and discussed in Section VI-B2.

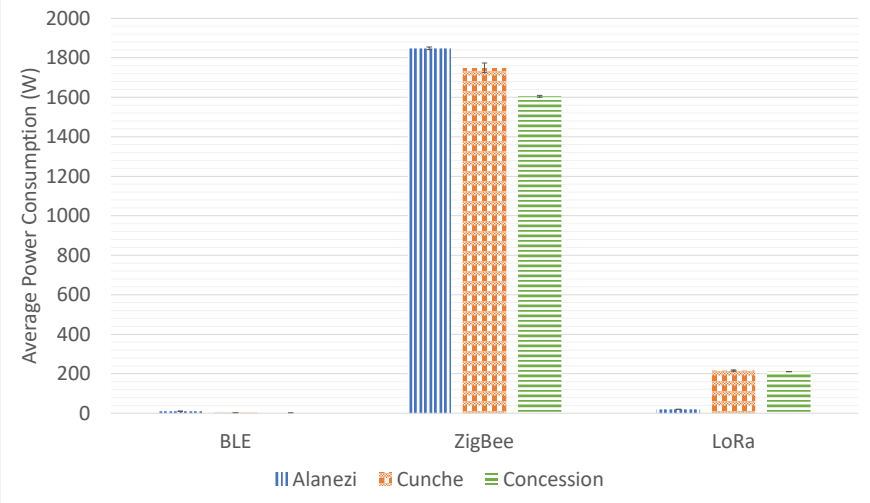


Fig. 36: Average user power consumption for different network technologies and negotiation algorithms in University scenario.

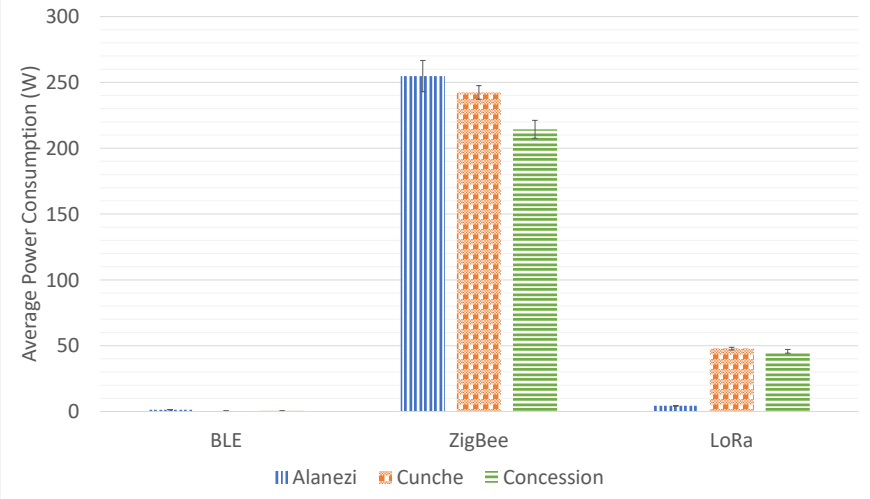


Fig. 37: Average user power consumption for different network technologies and negotiation algorithms in Shopping Mall scenario.

3) Power Consumption (IoT Device): In this metric we measured the power consumption of an IoT device averaged over the total number of runs. The main results are presented and discussed in Section VI-B3.

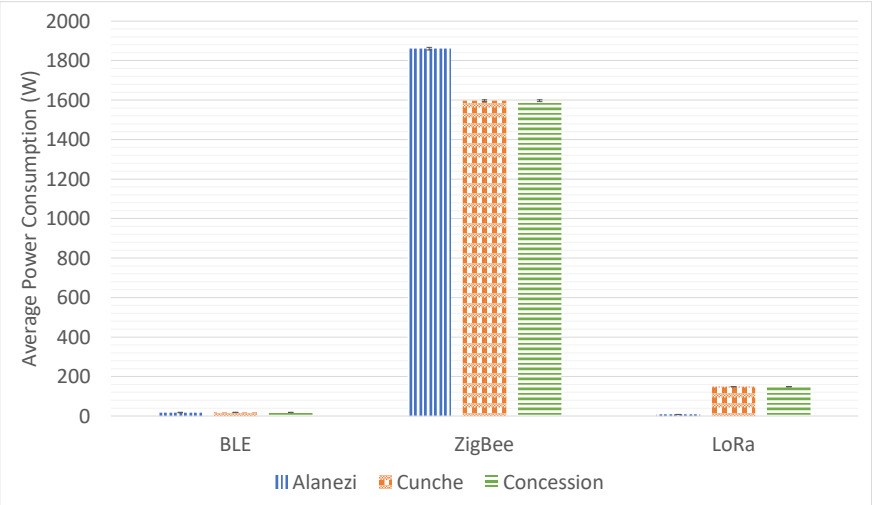


Fig. 38: Average IoT device power consumption for different network technologies and negotiation algorithms in University scenario.

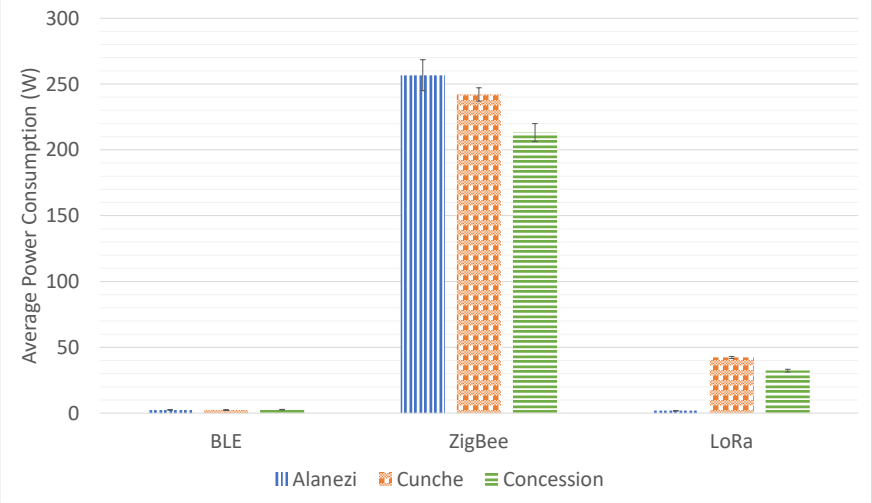


Fig. 39: Average IoT device power consumption for different network technologies and negotiation algorithms in Shopping Mall scenario.

REFERENCES

[1] P. Datta and B. Sharma, "A survey on IoT architectures, protocols, security and smart city based applications," in *2017 8th International conference on computing, communication and networking technologies (ICCCNT)*. IEEE, 2017, pp. 1–5.

[2] FinancesOnline. Number of Internet of Things (IoT) Connected Devices Worldwide 2024: Breakdowns, Growth and Predictions. Accessed: Dec., 2023. [Online]. Available: https://financesonline.com/number-of-internet-of-things-connected-devices/

[3] M. Seliem, K. Elgazzar, and K. Khalil, "Towards privacy preserving iot environments: a survey," *Wireless Communications and Mobile Computing*, vol. 2018, pp. 1–15, 2018.

[4] G. Chikukwa, "A Consent Framework for the Internet of Things in the GDPR Era," 2021.

[5] D. J. Solove, "Introduction: Privacy self-management and the consent dilemma," *Harv. L. Rev.*, vol. 126, p. 1880, 2012.

[6] V. Morel, M. Cunche, and D. Le Métayer, "A Generic Information and Consent Framework for the IoT," in *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, Aug. 2019, pp. 366–373.

[7] T. Baarslag, M. Kaisers, E. Gerding, C. M. Jonker, and J. Gratch, "When will negotiation agents be able to represent us? The challenges and opportunities for autonomous negotiators." International Joint Conferences on Artificial Intelligence, 2017.

[8] A. Das, M. Degeling, D. Smullen, and N. Sadeh, "Personalized privacy assistants for the internet of things: Providing users with notice and choice," *IEEE Pervasive Computing*, vol. 17, no. 3, pp. 35–46, 2018.

[9] J. Colnago, Y. Feng, T. Palanivel, S. Pearman, M. Ung, A. Acquisti, L. F. Cranor, and N. Sadeh, "Informing the design of a personalized privacy assistant for the internet of things," in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 2020, pp. 1–13.

[10] M. Cunche, D. L. Métayer, and V. Morel, "ColoT: A consent and information assistant for the IoT," in *Proceedings of the 13th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. ACM, Jul. 2020, pp. 334–336.

[11] T. Baarslag, "What to bid and when to stop," Ph.D. dissertation, Delft University of Technology, 2014.

[12] M. J. Osborne, "Bargaining and markets," 1990.

[13] E. H. Gerding, D. D. B. van Bragt, and J. A. La Poutré, *Scientific approaches and techniques for negotiation: a game theoretic and artificial intelligence perspective*. Citeseer, 2000.

[14] K. Binmore and N. Vulkan, "Applying game theory to automated negotiation," *Netnomics*, vol. 1, pp. 1–9, 1999.

[15] J. Z. Rubin and B. R. Brown, *The social psychology of bargaining and negotiation*. Elsevier, 2013.

[16] C. Beam and A. Segev, "Automated negotiations: A survey of the state of the art," *Wirtschaftsinformatik*, vol. 39, no. 3, pp. 263–268, 1997.

[17] R. H. Guttman and P. Maes, "Agent-mediated integrative negotiation for retail electronic commerce," in *International Workshop on Agent-Mediated Electronic Trading*. Springer, 1998, pp. 70–90.

[18] T. Bosse and C. M. Jonker, "Human vs. computer behavior in multi-issue negotiation," in *Rational, Robust, and Secure Negotiation Mechanisms in Multi-Agent Systems (RRS'05)*. IEEE, 2005, pp. 11–24.

[19] R.-J. Dzeng and Y.-C. Lin, "Searching for better negotiation agreement based on genetic algorithm," *Computer-Aided Civil and Infrastructure Engineering*, vol. 20, no. 4, pp. 280–293, 2005.

[20] M. M. Delaney, A. Foroughi, and W. C. Perkins, "An empirical study of the efficacy of a computerized negotiation support system (NSS)," *Decision Support Systems*, vol. 20, no. 3, pp. 185–197, 1997.

[21] T. Baarslag, A. T. Alan, R. Gomer, M. Alam, C. Perera, E. H. Gerding, and m. schraefel, "An Automated Negotiation Agent for Permission Management," in *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, 2017, pp. 380–390.

[22] D. Filipczuk, T. Baarslag, E. H. Gerding, and M. Schraefel, "Automated privacy negotiations with preference uncertainty," *Autonomous Agents and Multi-Agent Systems*, vol. 36, no. 2, p. 49, 2022.

[23] Y. Mohammad and S. Nakadai, "Utility elicitation during negotiation with practical elicitation strategies," in *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2018, pp. 3100–3107.

[24] N. Kökciyan, P. Yolum *et al.*, "Taking situation-based privacy decisions: Privacy assistants working with humans," in *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, 2022, pp. 703–709.

[25] O. R. Sanchez, I. Torre, and B. P. Knijnenburg, "Semantic-based privacy settings negotiation and management," *Future Generation Computer Systems*, vol. 111, pp. 879–898, 2020.

[26] M. Langheinrich, "A privacy awareness system for ubiquitous computing environments," in *UbiComp 2002: Ubiquitous Computing: 4th International Conference Göteborg, Sweden, September 29–October 1, 2002 Proceedings 4*. Springer, 2002, pp. 237–245.

[27] R. Neisse, G. Baldini, G. Steri, Y. Miyake, S. Kiyomoto, and A. R. Biswas, "An agent-based framework for informed consent in the internet of things," in *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*. IEEE, 2015, pp. 789–794.

[28] S.-C. Cha, M.-S. Chuang, K.-H. Yeh, Z.-J. Huang, and C. Su, "A user-friendly privacy framework for users to achieve consents with nearby BLE devices," *IEEE Access*, vol. 6, pp. 20 779–20 787, 2018.

[29] K. Alanezi and S. Mishra, "A Privacy Negotiation Mechanism for the Internet of Things," in *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*, Aug. 2018, pp. 512–519.

[30] ——, "Incorporating Individual and Group Privacy Preferences in the Internet of Things," *Journal of Ambient Intelligence and Humanized Computing*, vol. 13, no. 4, pp. 1969–1984, Apr. 2022.

[31] OMNeT++. Accessed: Dec. 20, 2023. [Online]. Available: https://omnetpp.org

[32] NS-3. Accessed: Dec.20, 2023. [Online]. Available: https://www.nsnam.org

[33] R. Sharma, V. Vashisht, and U. Singh, "Modelling and simulation frameworks for wireless sensor networks: a comparative study," *IET Wireless Sensor Systems*, vol. 10, no. 5, pp. 181–197, 2020.

[34] H. M. A. Fahmy, "Simulators and emulators for wsns," in *Concepts, Applications, Experimentation and Analysis of Wireless Sensor Networks*. Springer, 2023, pp. 547–663.

[35] M. Radhika and P. Sivakumar, "Performance Analysis of Energy Efficient Video Transmission Using LEACH Based Protocol in WSN," in *Computer Vision and Machine Intelligence Paradigms for SDGs: Select Proceedings of ICRTAC-CVMIP 2021*. Springer, 2023, pp. 103–111.

[36] H. Joshi, M. Bakhar, and R. Eklarker, "Modified X-MAC/CA Protocol to Improve Throughput and Energy Efficiency in Wireless Sensor Networks," *International Journal of Intelligent Systems and Applications in Engineering*, vol. 11, no. 3, pp. 731–737, 2023.

[37] D. Godfrey, B. Suh, B. H. Lim, K.-C. Lee, and K.-I. Kim, "An Energy-Efficient Routing Protocol with Reinforcement Learning in Software-Defined Wireless Sensor Networks," *Sensors*, vol. 23, no. 20, p. 8435, 2023.

[38] B. T. Website. Core Specification. Accessed: Dec., 2023. [Online]. Available: https://www.bluetooth.com/specifications/specs/core-specification-5-4/

[39] A. M. Yousuf, E. M. Rochester, B. Ousat, and M. Ghaderi, "Throughput, coverage and scalability of LoRa LPWAN for internet of things," in *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*. IEEE, 2018, pp. 1–10.

[40] LoRa Alliance. Accessed: Dec., 2023. [Online]. Available: https://www.lora-alliance.org

[41] ——. Technical Specifications. Accessed: Dec., 2023. [Online]. Available: https://resources.lora-alliance.org/technical-specifications

[42] ZigBee Alliance. Zigbee Specification. Accessed: Dec. 20, 2023. [Online]. Available: https://zigbeealliance.org/wp-content/uploads/2019/11/docs-05-3474-21-0csg-zigbee-specification.pdf

[43] E. Casilari, J. M. Cano-García, and G. Campos-Garrido, "Modeling of current consumption in 802.15.4/zigbee sensor motes," *Sensors*, vol. 10, no. 6, pp. 5443–5468, 2010.

[44] C. M. Ramya, M. Shanmugaraj, and R. Prabakaran, "Study on ZigBee technology," in *2011 3rd international conference on electronics computer technology*, vol. 6. IEEE, 2011.

[45] A. Tomar, "Introduction to ZigBee technology," *Global Technology Centre*, vol. 1, 2011.

[46] P. H. Kindt, D. Yunge, R. Diemer, and S. Chakraborty, "Energy Modeling for the Bluetooth Low Energy Protocol," *ACM Transactions on Embedded Computing Systems*, vol. 19, no. 2, pp. 1–32, Mar. 2020.

[47] Y. Chen, M. Li, P. Chen, and S. Xia, "Survey of cross-technology communication for IoT heterogeneous devices," *IET Communications*, vol. 13, no. 12, pp. 1709–1720, 2019.

[48] G. Sreelekshmi, G. S. Kumar, and P. Ushakumari, "Application of Queueing Model to an Outpatient Flow in a Multi-Speciality Hospital," *Journal of Pharmaceutical Negative Results*, pp. 332–337, 2022.

[49] Y.-C. Chen, J. Kurose, and D. Towsley, "A mixed queueing network model of mobility in a campus wireless network," in *2012 Proceedings IEEE INFOCOM*. IEEE, 2012, pp. 2656–2660.

[50] F. Ying and N. O'Clery, "Modelling COVID-19 transmission in supermarkets using an agent-based model," *Plos one*, vol. 16, no. 4, p. e0249821, 2021.

[51] K. Fitzpatrick, M. A. Brewer, and S. Turner, "Another look at pedestrian walking speed," *Transportation research record*, vol. 1982, no. 1, pp. 21–29, 2006.

[52] P. Kumaraguru and L. F. Cranor, "Privacy indexes: a survey of Westin's studies," 2005.

[53] J. Phelps, G. Nowak, and E. Ferrell, "Privacy Concerns and Consumer Willingness to Provide Personal Information," *Journal of Public Policy & Marketing*, vol. 19, no. 1, pp. 27–41, Apr. 2000.

[54] T. Bouguera, J.-F. Diouris, J.-J. Chaillout, R. Jaouadi, and G. Andrieux, "Energy consumption model for sensor nodes based on LoRa and LoRaWAN," *Sensors*, vol. 18, no. 7, p. 2104, 2018.

[55] J.-R. Lin, T. Talty, and O. K. Tonguz, "On the potential of bluetooth low energy technology for vehicular applications," *IEEE Communications Magazine*, vol. 53, no. 1, pp. 267–275, 2015.

[56] S. Preibusch, "Implementing privacy negotiations in e-commerce," in *Asia-Pacific Web Conference*. Springer, 2006, pp. 604–615.

[57] "Top 10 IoT applications in 2020," https://iot-analytics.com/top-10-iot-applications-in-2020/, Jul. 2020, Accessed: May 2, 2023.

[58] "Resources — RetailNext," https://retailnext.net/resources, Accessed: May 2, 2023.

[59] "Occupancy Monitoring with IoT Sensors," https://www.iotforall.com/occupancy-monitoring-with-iot-sensors, Accessed: May 2 2023.

[60] S. Kokolakis, "Privacy Attitudes and Privacy Behaviour: A Review of Current Research on the Privacy Paradox Phenomenon," *Computers & Security*, vol. 64, pp. 122–134, Jan. 2017.

[61] J. L. Dupree, R. Devries, D. M. Berry, and E. Lank, "Privacy personas: Clustering users via attitudes and behaviors toward security practices," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, 2016, pp. 5228–5239.

[62] R. G. Sargent, "Verification and Validation of Simulation Models," in *Proceedings of the 2010 Winter Simulation Conference*, Dec. 2010, pp. 166–183.

[63] S. Kamath and J. Lindh. Application Note AN092: Measuring Bluetooth Low Energy Power Consumption. Revision SWRA347a. Accessed: Sep. 1, 2023. [Online]. Available: https://www.ti.com/lit/an/swra347a/swra347a.pdf

[64] A. Del Campo, L. Cintioni, S. Spinsante, and E. Gambi, "Analysis and tools for improved management of connectionless and connection-oriented BLE devices coexistence," *Sensors*, vol. 17, no. 4, p. 792, 2017.

[65] F. Goovaerts, G. Acar, R. Galvez, F. Piessens, and M. Vanhoef, "Improving privacy through fast passive Wi-Fi scanning," in *Secure IT Systems: 24th Nordic Conference, NordSec 2019, Aalborg, Denmark, November 18–20, 2019, Proceedings 24*. Springer, 2019, pp. 37–52.

[66] V. Jain, V. Laxmi, M. S. Gaur, and M. Mosbah, "ETGuard: Detecting D2D attacks using wireless evil twins," *Computers & Security*, vol. 83, pp. 389–405, 2019.

[67] Y. Yi, G. Gong, and K. Mandal, "Implementation of three LWC Schemes in the WiFi 4-Way Handshake with Software Defined Radio," *arXiv preprint arXiv:1909.11707*, 2019.

[68] I. A. Qureshi and S. Asghar, "A Systematic Review of the IEEE-802.11 Standard's Enhancements and Limitations," *Wireless Personal Communications*, vol. 131, no. 4, pp. 2539–2572, Aug. 2023.

[69] M. Gast, *802.11 wireless networks: the definitive guide*. " O'Reilly Media, Inc.", 2005.

[70] P. Tilghman and D. Rosenbluth, "Inferring wireless communications links and network topology from externals using granger causality," in *MILCOM 2013-2013 IEEE Military Communications Conference*. IEEE, 2013, pp. 1284–1289.

ACRONYMS

Notation	Description	Page List
ADR	Adaptive Data Rate	15
ADV	Advertisement	12–14
ANOVA	Analysis of Variance	49
AP	Access Point	68, 70–73, 76–79
BI	Beacon Interval	18
BLE	Bluetooth Low Power	9, 10, 12–14, 19, 21, 22, 24, 26, 33, 35–37, 42, 45, 47, 48, 52, 56, 58–67, 72, 74
CA	Collision Avoidance	18
CAP	Contention Access Period	18
CFP	Contention Free Period	18
CR	Coding Rate	15
CSMA	Carrier Sense Multiple Access	18
CSS	Chirp Spread Spectrum	15
CTC	Cross-Technology Communications	33
DC	Data Collector	22
DS	Data Subject	4, 5, 7–10, 20, 24, 53
DUNE	Device, User, Network, Environment	5, 6, 19–24, 32, 33, 53, 54
FEC	Forward Error Correction	15
FFD	Full-Function Device	17, 18

Notation	Description	Page List
GATT	Generic ATTribute Profile	9
GEPARD	General Environment for Privacy Assistant Research and Design	5, 6, 11, 22–25, 29, 32–37, 39, 44, 52, 54
GLM	Generalized Linear Model	49
HTTP	Hypertext Transmission Protocol	74
IEEE	Institute of Electrical and Electronics Engineers	16, 17
IoT	Internet of Things	4–6, 8–11, 14, 19–24, 31, 33–35, 37–40, 42–45, 48, 49, 52–54, 60, 67, 73, 82, 83
IR	Infrared	40
IRR	IoT Resource Registry	10
ISM	Industrial, Scientific, and Medical	12, 14
LoRa	Long Range	14–16, 24, 26, 33, 35–37, 45, 47, 48
LPWAN	Low-Power Local Area Network	14
MAC	Media Access Control	15–17, 42, 43
MEM	Mixed Effects Model	49, 50
ML	Machine Learning	53
OLS	Ordinary Least Squares	49, 50

Notation	Description	Page List
PA	Privacy Assistant	4–6, 8–12, 19–24, 27, 34–39, 44, 45, 47–56, 60, 64, 67, 77
pawS	privacy awareness Systems	9
PDC	Personal Data Custodian	22
PEP	Policy Enforcement Point	10
PHY	Physical	14, 16, 17
PP	Privacy Policies	5, 9, 19, 20, 22, 26, 27, 29, 36–40, 42, 43, 47, 49, 51, 53
PPA	Personalized Privacy Assistant	10
RAM	Random Access Memory	52
RFD	Reduced-Function Device	17
RPOV	Relative Proportion of Variation	25, 32
SF	Spreading Factor	15
SHA	Secure Hash Algorithm	43
WPAN	Wireless Personal Area Network	16
WSN	Wireless Sensor Network	9
XML	Extensible Markup Language	29, 31, 40, 42, 63, 69, 74