UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Computer Science Curriculum

Anton Slavin

Wireless Network Connectivity in Educational IoT Environments

Master's Thesis (30 ECTS)

Supervisor: Ulrich Norbisrath, PhD

Tartu 2024

Wireless Network Connectivity in Educational IoT Environments

Abstract:
The aim of this thesis is to develop a reliable access point (AP) configuration solution for a range of Wi-Fi-enabled devices to facilitate Internet of Things (IoT) education in classrooms. By designing and conducting experiments on different hardware devices, the final goal is to create a semi-automatic AP configuration program within the custom IoTempower framework. The significance of this work lies in addressing the lack of a unified solution for setting up stable APs in educational settings, where ease of use and compatibility are crucial. This thesis includes a comprehensive comparison of APs, a proposed custom solution for Linux-enabled devices, and detailed methodology and results, ultimately contributing to the field of IoT education by simplifying the AP setup process for students and educators.

Keywords:
Access point, gateway, IoT

CERCS:
P170 Computer science, numerical analysis, systems, control
P175 Informatics, systems theory

Traadita võrguühenduvus hariduslikes IoT keskkondades

Lühikokkuvõte:
Magistritöö eesmärk on arendada usaldusväärne juurdepääsupunkti seadistamise lahendus erinevatele WiFi toega seadmetele, et hõlbustada asjade interneti (IoT) õpetamist klassiruumis. Viies läbi katsed erinevate riistvaraseadmetega, lõppeesmärk on luua programm, mis võimaldab lihtsat juurdepääsupunkti konfigureerimist *IoTempower*i raamistikus. Tulemusena viidi läbi põhjalik pääsupunktide võrdlus ning koostati uurimismetoodika. Võrdluse käigus olid tuvastatud mitmed seadmete piirangud ja probleemid, mis takistavad nende kasutamist pääsupunkti rollis. Lisaks oli loodud tarkvaralahendus Linux-toega seadmetele, mis võimaldab automaatset pääsupunkti seadistamist, arvestades seadme WiFi kiibi eripärasid.

Võtmesõnad:
Pääsupunkt, võrgulüüs, IoT

CERCS:
P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine
P175 Informaatika, süsteemiteooria

Contents

1 Introduction

The Internet of Things (IoT) is actively transforming fields ranging from healthcare to smart home technologies. This shift highlights the need to teach students essential skills for working with IoT systems, especially in a hands-on manner, to gain practical insights into building the systems. A key part of any IoT system is the access point (AP), which allows devices to communicate with each other. In this thesis, I aim to create a solution for setting up stable access points on various Wi-Fi-enabled devices, specifically for teaching IoT in classrooms. I achieve this goal through a series of experiments with different hardware devices, leading to the creation of a tool suite for managing IoT setups in educational settings. This tool suite is created for and works within the custom IoTempower framework, as detailed in Section 2.2.

Access points play a crucial role in any IoT system, affecting overall performance and user experience. Despite their importance, no unified knowledge base or solution currently documents the compatibility and stability of different hardware configurations for running APs, especially in educational contexts with students of varying backgrounds and skill levels. Existing market solutions for APs often do not target specifically IoT education, nor do they provide an auto-configuration solution that works with a wide range of devices. This gap forces students to either learn the complexities of networking or rely on specialist support, restricting the exploratory learning process of IoT education.

Moreover, the choice of an AP device is not always simple to make, especially when searching for low-cost alternatives to industrial routers. While most devices with a Wi-Fi chip might suffice, previous experience shows that many chips have undocumented limitations and problems, such as the very low client limit on Broadcom chips inside Raspberry Pi devices, among others [1].

Consider the following use case, which is based on real experiences (including my personal), and is the driving force behind this thesis:

Jaan, a curious and enthusiastic student at the University of Tartu, eagerly enrolled in a hands-on IoT class. As part of the course, he received a kit packed with various sensors, actuators, and a Raspberry Pi. Excited to dive into his first project, Jaan set up the Raspberry Pi with Linux and the IoTempower framework, anticipating a smooth start to his IoT learning journey.

However, Jaan soon encountered his first hurdle: connecting the Raspberry Pi to the university's network. Despite multiple attempts, he couldn't establish a stable connection. After several frustrating hours, he discovered that the university network required a special configuration not mentioned in any course materials or documentation.

Just as he resolved the network issue, another setback occurred. The SD card for the Raspberry Pi became corrupted, most likely from him accidentally turning off the device too fast. This unexpected mishap cost him valuable time and data, forcing him to start over with a new card and reconfigure his system from scratch.

Working on his group project for the course, Jaan embarked on an assignment that involved integrating a large amount of sensors into the IoT network. Initially, everything seemed to be working, but soon, some of the sensors began to fail intermittently. The connectivity issues were erratic, making it difficult to diagnose the problem. After exhaustive testing and research, Jaan uncovered that the root of their troubles was a little-known limitation of the Raspberry Pi: it could only handle a limited number of

client connections simultaneously. This limitation was not documented anywhere, and discovering it was a revelation that explained the erratic sensor behavior.

Wouldn't it be useful for Jaan to know all these limitations beforehand and potentially choose another device instead of the Raspberry Pi to make his learning experience smoother?

In this thesis, I provide a detailed comparison of different access points and their suitability for teaching IoT concepts, focusing on ease of setup, stability, and features. I conduct a series of experiments to evaluate the performance of AP configurations in IoT systems. By identifying key problems with current setups, I propose a custom solution for most Linux-enabled devices with wireless capabilities. This solution aims to provide a user-friendly program for IoTempower that can semi-automatically configure an AP for classroom use and is compatible with a wide range of hardware. Unlike existing standalone scripts, this solution automatically recognizes the device's Wi-Fi module and performs the necessary steps to ensure that the created AP is optimal. The main users of this system are students and educators (lab assistants) who need a simple, reliable way to set up APs without deep networking knowledge.

In Section 2, I introduce access points and describe them in detail, both in general and within the context of IoT systems. I discuss various types of access points, their typical hardware components, and the architecture of Wi-Fi chipsets. I then explore the role of APs in IoT education.

Following this, I discuss related work in Section 3. A more thorough requirements analysis is then detailed in Section 4, deriving specific requirements for the final solution from the described use case.

The following Section 5 details my methodology for testing and comparing several hardware devices as APs. I provide descriptions of each device and comments on experiments conducted.

Next, in Section 6, I introduce my custom solution for an automatic configuration system tailored for Linux-enabled devices within the IoTempower framework. I explain the codebase and its components, highlighting the programming language choice, system functionality, and integration. I discuss the strengths and limitations of the system, along with usage guidelines and some testing steps to verify the solution.

In Section 7, I discuss the results of this thesis and outline potential areas for future research and further development. Finally, Section 8 concludes this thesis and provides final remarks.

2 Background

This section describes the purpose of using an access point[1] (AP) in and outside the context of IoT systems. It begins with the background and history of access points, followed by a description of the different types of access points, their architecture, and the use of APs in IoT education.

[1]Also known as hotspot and gateway, especially in IoT contexts.

2.1 Access Point

A typical IoT system is composed of several key components: Internet access through Ethernet, a main gateway or access point, actuators, sensors, and a cloud server. The access point, crucial to the system, serves as the primary connection to the Internet and routes data between all components, often using Wi-Fi but also commonly using Bluetooth, Zigbee, or LoRa. The role of the gateway is also important due to the concept of fog computing, which has become ever so prevalent in IoT systems, wherein the gateway is used for computation and data storage between the nodes (actuators and sensors) and the cloud [2].

Wireless networks are crucial in IoT, as they allow devices to be placed virtually anywhere, especially with mobile power sources. Advances in wireless internet protocols made it possible for most IoT devices to use Wi-Fi for data transmission. Over time, the size and energy requirements of Wi-Fi-enabled IoT devices decreased, resulting in small and efficient products like the ESP8266 and ESP32 by Espressif Systems. These advancements have made it convenient to create embedded projects for various use cases. For instance, the ESP8266 is available for as low as USD 1[2], making it highly accessible.

The rapidly increasing popularity of IoT has been unprecedented, impacting various domains such as home automation (smart homes), healthcare, retail and logistics, and environmental monitoring, as discussed by Ramlowat and Pattanayak [3]. As such, teaching IoT has become increasingly important, especially in a hands-on manner. IoT concepts can be taught to both beginners and advanced students due to their scalable nature. Beginners can learn basic concepts and programming, while advanced students can implement more complex systems.

One significant advantage of IoT is its low cost of entry, with most sensors and actuators being highly affordable. For instance, the ESP8266 can be purchased for USD 1-3, the ESP32 for USD 3-6, and various sensors range from 50 cents to USD 3 each. This affordability makes experimenting with IoT systems accessible and engaging. However, managing the systems requires a central control device (the gateway), the choice of which is not as straightforward to make. While industrial IoT routers and branded devices often offer additional protocol support, such as Zigbee, they are usually expensive and require special knowledge to set up. Therefore, considering the high customizability of most sensors and actuators, it is often more practical to avoid proprietary solutions and prefer low-cost, open-source alternatives. This outlines the first requirement for the investigation and final system: the potential AP devices must be accessible (low cost) and easily configurable by students.

2.2 IoT and Education

Many professors and educators at UT employ a practical approach to teaching, incorporating hands-on classes and labs. These classes are designed to allow students to experiment with various concepts and tools, fostering a deeper understanding. To achieve this, some technical concepts, such as advanced networking and Linux/OS principles, are simplified or abstracted for beginner students.

The IoTempower framework[3] was developed to address the challenge of configuring

[2]All price estimates provided are subject to change and may vary by region.

[3]`https://github.com/iotempire/iotempower`

complex networking setups. Its core objective is to facilitate the rapid and easy setup of networks, even for those with limited networking knowledge. This framework is extensively utilized in various classes, particularly the Internet of Things (IoT) course at UT, as well as in numerous IoT workshops and classes globally.

IoTempower supports a wide range of operating systems and hardware devices. It is most commonly deployed on Raspberry Pi and similar devices, Linux laptops, Android phones, and Mac laptops. Within the framework, access point functionality is provided using the `hostapd` tool, along with an MQTT server featuring a custom CLI, capabilities for flashing various sensor/actuator devices, a web interface with console access and documentation, and integration tools such as Node-RED. These features significantly simplify the creation of IoT projects, making them particularly accessible to beginner students and creators.

In the context of teaching IoT, the choice of access point directly impacts how efficiently students can set up their systems and the level of configuration required for learning and experimentation. At UT's IoT course, the Raspberry Pi has been the preferred hardware device, mostly due to its good open-source support, especially for running Linux [4].

Despite the positive sides of using a Raspberry Pi and, recently, HP EliteBook laptops in IoT education, these devices have some significant limitations and issues that are not well documented. The limitations discussed here are derived from personal and professional experience, as well as from online discussions and forums describing common pitfalls of setting up demo networks in hands-on classes.

Firstly, there is spotty support for "AP mode" in Wi-Fi chipsets, and it is not always clear whether a device with wireless capabilities can be used as an AP. This is especially true with older hardware devices, such as old laptops.

Even if a chosen Wi-Fi chip supports AP mode, there may be limitations on the number of clients it can support, as seen with the Raspberry Pi 3 and 4. These issues are not disclosed officially and are often unknown without some research, which many users may not be able to conduct. However, it is frequently mentioned in various Raspberry Pi forum posts and in projects like Internet in a Box[4] and MoodleBox[5] [5] [6] [7] [8] [9].

During the Internet of Things course at UT, various networking issues were observed by myself, my project partners, and other course-mates. These issues were difficult to debug or fix, especially for beginner students with limited backgrounds in technology or networking. Similar to the Raspberry Pi, HP EliteBook laptops were observed to have an upper client limit, disallowing more than a handful of connections to be made simultaneously. Most manufacturers of Wi-Fi modules keep a significant portion of their inner workings secret. Even if manufacturers provide schematics, they are often not detailed enough [10], and crucial parts are frequently omitted. Furthermore, the firmware provided by manufacturers is usually in the form of closed-source binary blobs, with no explanations of how it functions.

[4] `https://internet-in-a-box.org/`
[5] `https://moodlebox.net/en/`

2.3 Architecture

In this and the following sections, I discuss the inner workings of typical access points, with a focus on Wi-Fi chipsets used for AP functionality. This information is relevant for the upcoming experimentation section, as well as the configuration system implementation.

Typically, access points employ an architecture similar to that of a computer or server, optimized for long uptime and multiple connected clients. This architecture is often referred to as a network operating system, which may be proprietary or open-source and frequently Linux-based. In addition to Linux-based systems, there are other network operating systems such as FreeBSD-based Junos OS[6], Cisco IOS[7], MikroTik's RouterOS[8], and DD-WRT[9]. However, I will focus on Linux-based systems in this thesis.

For an access point to function, it requires a CPU, much like a regular computer. Modern Wi-Fi chips in mobile devices are produced by several large manufacturers, including Qualcomm, Broadcom, MediaTek, Intel, Realtek, Samsung, and Apple, although the latter often utilize Broadcom chips in their products. These companies' chips are found in most laptops, phones, routers, and other internet devices across various price ranges and categories. These chips are also the focus of extensive research, reverse engineering efforts, documentation, and discussions due to their ubiquity. Notably, many devices discussed in this thesis use these chips, particularly those from Broadcom, Intel, and Realtek.

System-on-a-chip (SoC) architectures are particularly prominent, as they integrate all required components, including Wi-Fi (and often Bluetooth) functionality, into a single chip. These Wi-Fi chips are integrated with the device's main host operating system and include specific firmware and drivers to ensure compatibility. The most common operating systems for these devices are Linux (used in routers, embedded devices, laptops, and phones), Android (for phones), Windows, and macOS (for laptops and desktop computers). These are also very commonly found in embedded systems and small Single-Board Computers (SBC), such as the Raspberry Pi.

There are two main types of Wi-Fi chipsets: **FullMAC** and **SoftMAC**[10], the former of which contains a special processing unit inside, while the latter is fully implemented in the host operating system [10]. SoftMAC chips move the processing of Wi-Fi frames entirely to the OS, which results in more power usage (but provides access to raw frames if needed), while FullMAC chips usually hide the raw frames and process them separately.

The term **SoftAP** refers to the method of utilizing some non-router device as an access point with software by using the device's existing wireless hardware and configuring it through specific software settings. SoftAP is relevant to this thesis, as it is often the only option to run an AP on old laptops or other devices that are not specifically built or marketed as routers.

[6] `https://www.juniper.net/us/en/products/network-operating-system/junos-os.html`

[7] `https://www.cisco.com/c/en/us/products/ios-nx-os-software/index.html`

[8] `https://mikrotik.com/software`

[9] `https://dd-wrt.com`

[10] MAC stands for Medium Access Control in this case, and it represents part of the Data Link layer in the Open System Interconnections (OSI) model framework – responsible for managing raw Wi-Fi frames and encapsulating them for transmission.

2.3.1 Hardware

For a Wi-Fi chipset to function, unique hardware components such as the baseband processor, RF transceiver, power amplifier, and antennas are required. As the IEEE 802.11 standards define frames and packets, their creation, processing, transmission, and reception must be possible using radio components capable of transmitting and receiving radio signals. These components are controlled using the deployed firmware, often interfaced using SDIO and PCIE protocols [10].

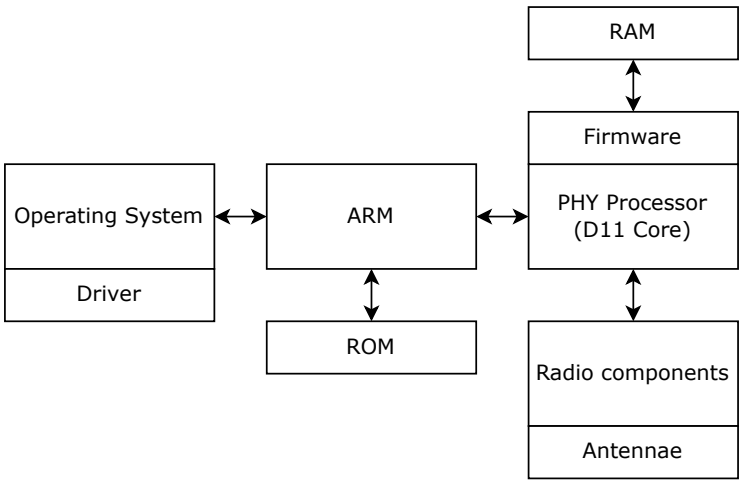

Figure 1. Simplified diagram of a typical FullMAC Wi-Fi module [10].

Generally, the following critical components are required for a Wi-Fi chip (most often as a System-on-a-chip) to function, based on the structure of Broadcom chips [11] [10]:

RAM Random Access Memory in an access point serves as the primary storage for running processes and data in transit, especially in case of SoftMAC chips. It's crucial for buffering data packets, managing active connections, and running the access point's firmware and operating system.

ROM Read-only Memory serves as the storage location for firmware (and essential initialization code) on a Wi-Fi chip. As the name suggests, this memory module only supports read operations, ensuring that the firmware remains unchanged during operation.

Data Transfer All Wi-Fi frames are sent between various components inside the Wi-Fi chipset by using either DMA (Direct Memory Access) or PCIe/SDIO interfaces. These interfaces facilitate efficient data transfer with minimal CPU intervention, thereby optimizing performance.

ARM Processor[11] This is the central processing unit for handling Wi-Fi data transmissions, embedded inside FullMAC chipsets. It manages tasks such as encoding and decoding Wi-Fi signals, frame processing, and managing wireless protocols.

Antennas Antennas are critical for transmitting and receiving radio signals. They are often designed for optimal range and signal strength and can sometimes be directional or omnidirectional, depending on the application. A special setup called multiple-input

[11]Referred to as D11/PHY Core in Broadcom chips.

and multiple-output (MIMO) is also commonly present on routers, allowing for better performance and reduced multipath propagation issues. The performance of the antennas is also dependent on the amount of obstructions around them, the distance to other devices, and the amount of power available, among other factors.

Signal Generators These components generate the radio frequency signals that are transmitted by the antennas. The quality and strength of the wireless signal depend on the signal generators. More generally, DAC and ADC components, along with various amplifiers and filters, are used to ensure high-quality signal processing and transmission.

Internal Data Processing Pipeline To transmit Wi-Fi data, the pipeline typically starts with signal generation, where data is converted into a radio signal. In the case of FullMAC chips, the embedded ARM processor handles this data, managing how it is packaged according to Wi-Fi standards. For SoftMAC chips, the host system's CPU handles these tasks, including frame processing and protocol management. Data is then temporarily stored in RAM for quick access and transmission. The antennas transmit the processed signal. When receiving data, this process is reversed, with the antennas receiving the signal, converting it back into data, and then processing it through the RAM and either the D11 core or host system CPU before it is passed to the intended application or service.

2.3.2 Firmware

Wi-Fi modules require firmware to function and communicate with the host operating system, typically with the help of a driver.

Firmware is usually stored in flash memory or the ROM component. This allows the firmware to be retained even when the device is powered off. In the case of most systems, the firmware is loaded into the memory from a binary file, stored on the OS itself.

The firmware serves as the low-level control system for the access point. It initializes and manages hardware components during boot-up and operational phases, as it has full control over all components of a Wi-Fi module [10]. Essentially, the firmware acts as an abstraction layer, translating high-level commands into low-level hardware operations. This involves controlling the flow of data between RAM, the embedded ARM processor, antennas, and signal generators and managing the operational state of these components (e.g., power management, signal modulation) [12].

Each chip manufacturer provides their own firmware, usually distributed as a closed-source binary blob. As is the case with manufacturers such as Broadcom, they keep the firmware a secret and legally protect it quite well. As a result, even in open-source systems, closed-source binary blobs are loaded into the Wi-Fi chips to enable their functionality. However, some open-source firmware projects exist, providing more transparency and control. SDR (Software Defined Radio) is utilized to allow for full control over modulation, protocol handling, and signal processing. One such project is the openwifi[12] driver and software package.

[12] `https://github.com/open-sdr/openwifi.git`

2.3.3 Drivers

Higher-level software, like the operating system, interacts with the firmware through a set of predefined interfaces or drivers. These drivers abstract the hardware specifics and provide a standardized way for software applications to interact with the hardware.

While the firmware resides within the Wi-Fi module components and is specifically tailored to the hardware of the chip, drivers are installed onto the host OS and act as a bridge between the OS and firmware. Often, one driver can support multiple different OS and firmware versions.

In the case of Linux, drivers and headers are responsible for connecting the Wi-Fi chipset hardware to the Linux Kernel (and other components running in the kernel space) and the top-level applications and tools exposed to the user (running in user space).

One of the first Wi-Fi drivers for Linux was called Wireless-Extensions (WE), created by Jean Tourrilhes in 1997 [13]. Replacing WE, newer and more refined projects have emerged, primarily the nl80211 interface and its implementations: cfg80211 for FullMAC chips and mac80211 for SoftMAC chips.

NDISWrapper is also a solution worth noting that was developed to address the lack of native Linux drivers for many Wi-Fi cards. Originating in the early 2000s, NDISWrapper allowed the use of Windows XP drivers on Linux systems by implementing a wrapper around the Windows NDIS (Network Driver Interface Specification) API. This approach enabled many users to access wireless networking on Linux before more robust and native support was available. The project saw significant use during its peak, providing a critical bridge for Linux users needing wireless connectivity. However, as native Linux drivers improved and became more widespread, the reliance on NDISWrapper diminished.

A noteworthy modern driver is the open-source brcmsmac[13], targeting a broad range of Broadcom modules. It is based on mac80211. However, it still relies on closed-source firmware blobs to function.

From the OS user-space level, separate applications are used to control and configure the Wi-Fi chip. On Linux, multiple programs exist for this task: crda, hostapd, and NetworkManager, among others. Some of them also require supplicant programs such as wpa_supplicant[14], iwd, and additional tools like iw to function. Tools such as dnsmasq and iptables may also be necessary for handling DHCP and network address translation if they are not already managed by the system.

Typically, the following structure is used on Linux: At the kernel level, either a FullMAC or SoftMAC driver operates using nl80211 or cfg80211 on top (with mac80211 implementation for SoftMAC). Then, iwd (iNet Wireless Daemon) with ell (Embedded Linux Library) acts as an intermediary, translating everything to the user level for applications such as NetworkManager and/or ConnMan. This setup is illustrated in Figure 2. While a configuration like this is functional, it is quite fragmented and presents many points of failure, such as the use of closed-source firmware files, and potential incompatibility between versions of components.

Currently, the most popular user-side programs for configuring Wi-Fi are hostapd and NetworkManager, with the former being used in OpenWrt routers [14] and the latter

[13]Previously known as brcm80211: `https://wiki.debian.org/brcm80211`.

[14]A helper program that implements key negotiation with a WPA authenticator and controls roaming and IEEE 802.11 authentication/association of the wireless driver.

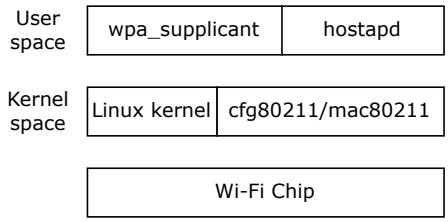

Figure 2. Diagram of a typical Wi-Fi networking setup on Linux.

becoming the preferred solution in an increasing number of desktop Linux environments. From the user's perspective, using these programs is quite straightforward if there are no technical problems with the Wi-Fi module on the kernel level. Otherwise, quite extensive debugging is required to troubleshoot issues. Moreover, not all configuration programs are compatible with each other, requiring additional (and not always well-documented) system changes to switch between programs.

This brings us to another requirement for the configuration solution: the AP program must be auto-configured automatically, ensuring that correct system settings are activated and that the correct driver and firmware are in use. If an alternative firmware version is available (which improves the performance of the AP), it must be preferred and activated automatically.

3 Related Work

In this section, I review relevant literature on the use of access points in IoT projects and similar contexts. The goal is to provide an overview of different AP implementations and uses in IoT settings.

Ford et al. [15] mention the use of MQTT and Raspberry Pi in IoT projects, which aligns well with the use case described in this thesis. Although their focus is not solely on Wi-Fi, their insights into the integration of MQTT with Raspberry Pi are relevant. From their research, we learn that we should consider more Raspberry Pi models, specifically Zero W and 3B, as they can potentially perform as well as the 4B+ at a lower cost. Additionally, using MQTT for benchmarking and measuring connections is a viable and elegant solution, which can especially be applied to IoTempower, due to the MQTT functionality already built into it.

Zhong and Liang [16] describe their process of designing a project-based IoT course using the Raspberry Pi as the main gateway and computer, which directly relates to the educational focus of this thesis. Their research highlights the difficulties students face in acquiring enough relevant information about the Raspberry Pi beyond simple instructions and initial set-up guides. As mentioned by Zhong and Liang: "The Raspberry Pi official website [...] does not have a systematic tutorial to guide the beginners; for example, the Getting Started Guide only shows how to install the system for the first time. What to do next is not clear" [16]. Moreover, their research highlights some issues with the Raspberry Pi, such as the missing support for WPA2 networks by default.

The work of Zhong and Liang highlights the importance of providing clear, simple

instructions and explanations to students, which also ties into the use case presented in the Introduction. As such, another requirement for the completed system is to have clear instructions provided to the user regarding the Wi-Fi module present and any of its limitations, if known beforehand.

Further, Roy et al. [17] explore advanced testing algorithms in IoT contexts, placing an emphasis on increasing throughput and raw performance. While their work is highly advanced and focuses on performance optimization, it is still tangentially related to the focus of this thesis.

Chang et al. [18] discuss the design of an AP for IoT, highlighting the strengths and weaknesses of different setups and system architecture in the context of IoT. Their proposed solution combines the Wi-Fi and Zigbee protocols. Although Zigbee is outside the scope of this thesis, their work is relevant to the broader context of AP design in IoT.

Machado et al. [19] discuss the use of IoTempower for IoT education, emphasizing the importance of hands-on learning. Their work supports the thesis' emphasis on practical, project-based learning in IoT education. Most notably, their work highlights the importance of the gateway (AP) as a platform in collaborative IoT projects [19].

Nagy and Colesa [20] research the use of Raspberry Pi as a router for IoT, with a focus on security and potential attacks. They bring out the most common vulnerabilities of IoT networks and create a final solution to address them. The solution consists of a configuration script for the Raspberry Pi, which includes commonly omitted steps, such as MAC filtering, the use of very strong passwords, firewall configuration to block certain ports, and more [20]. The suggestions are very valuable, but security comes at the cost of comfort and ease of use. When dealing with strictly classroom (demo) systems, it might be necessary to give up some security to simplify the network for students. However, it is still crucial to explain the importance of securing systems to learners and potentially showcase different vulnerabilities of a system as a practical exercise.

Moving on to AP configuration software, various applications exist for setting up networks and APs, but most rely on a single tool (such as iwd or NetworkManager) and provide a front-end for that tool only. One such example is the impala tool[15], which provides an intuitive UI in the terminal for configuring iwd Wi-Fi settings. Similarly, NetworkManager provides at least two different interfaces for configuring Wi-Fi: nmcli and nmtui, but both programs are capable of configuring strictly NM settings.

As such, we can formulate another requirement for the final solution: the system must be capable of creating and configuring multiple different AP programs based on the user's choice and available hardware.

4 Requirements Analysis

Taking the use case presented in the Introduction and related research from Section 3 into consideration, we can form a list of requirements for the AP hardware experiments and the configuration system.

The main issue highlighted so far has been related to the choice of the AP hardware: the previously chosen Raspberry Pi has been continuously showing signs of issues during classroom usage, often requiring extensive debugging and troubleshooting. Considering

[15] `https://github.com/pythops/impala`

that the Raspberry Pi is simply a single-board computer with a Wi-Fi chip, there are many alternatives to it. However, there is no guide or single source of information to rely on when choosing an alternative to the Raspberry Pi. So firstly, an investigation needs to be done, comparing a selection of devices by their performance in a demo classroom setting. Section 5 details the experimentation process for each device in the selection.

Before selecting the devices, the following requirements must be highlighted:

1. The device must be reasonably low-cost and easily obtainable, especially in bulk.
2. The device must feature a Wi-Fi module and be easily configurable.
3. The Wi-Fi module must support AP mode.
4. Depending on the class size and projects undertaken, the device must support 15-25 connected clients without issue. This includes devices connected for monitoring (potentially 1-5 if multiple group members need to monitor and administer), as well as various sensors and actuators.
5. If the device supports Linux, either OpenWrt or some other distribution capable of running IoTempower must be supported.

Devices that meet these criteria include the Raspberry Pi 3B, 4B+, Zero, and potentially the new model 5; other boards such as the Atomic Pi, external routers and boards such as the GL.iNet GL-MT3000 and the Hi-Link HLK-7628N kit. Next, most laptops (such as Lenovo ThinkPads and HP EliteBooks), smartphones, and even the ESP32 itself can potentially work and can be compared.

In general, the following categories of products can be considered as potential access point devices for the described use case, as summarized in Table 1. The specific examples of devices mentioned here are chosen based on their availability and affordability, as well as recommendations and insight from Ulrich Norbisrath.

#	**Category**	**Devices**
1	Single-boards PCs	Raspberry Pi, Atomic Pi
2	Routers	GL-MT3000, MT7628N, Tenda TX2 Pro
3	Laptops	Lenovo ThinkPad, HP EliteBook
4	Smartphones	Samsung Galaxy, Other Android
5	Miscellaneous	ESP32, USB Wi-Fi adapters

Table 1. Examples of devices potentially suitable for the described use case.

Besides the experiments, another goal of this thesis is to create a software solution capable of assisting students in setting up an AP on a range of different hardware. There are specific requirements for the configuration system as well, considering the use case and architectural differences of access points:

1. The system should be integrated into the IoTempower framework.
2. The system must be launched as a single program and be self-contained.

3. Instructions must be provided to the user at every step, and any information about the system hardware, features, and limitations must be conveyed. If possible, recommendations can be made to suggest alternative hardware to successfully run an AP if the user's device can not facilitate it.

4. The Wi-Fi module details (among other information) on the device must be detected, and any available bug fixes or patches must be applied by the program automatically.

5. Depending on the choice of the user, multiple AP programs must be supported (such as both hostapd and NetworkManager), and any relevant configuration changes on the host system must be made automatically.

The listed requirements above ensure that the finished solution solves most of the problems described in the use case and transforms the experience of students in an IoT classroom setting. Section 6 is dedicated to the network configuration system solution, detailing the structure of the system and its functionality. Details, such as the chosen programming languages, used libraries, and any shortcomings of the solution, are described there.

5 Connectivity Experiments

In this section, I describe the experimental procedures of the chosen hardware devices, including the criteria of selection for devices to be tested, the choice of the client device, methodology, and setup.

As mentioned in Section 4, there is a wide choice of potential AP devices available, ranging from specialized routers to miscellaneous devices with wireless networking capabilities. To measure the suitability of each device as an AP in the classroom, I am conducting a series of tests to gauge the ease of use of the device, discover any limitations or issues, and find the most suitable architecture or Wi-Fi chipset for the role.

For the experiments, I am considering the following aspects of each device for the role of an access point:

1. Supported operating system and architecture.

2. Time of release, manufacturer.

3. Wi-Fi module used inside, compatible firmware and drivers.

4. Price of the device, general availability.

5. Maximum number of clients supported, if documented.

After acquiring the devices, a sample classroom set-up is used for conducting all of the experiments, introduced in more detail in Section 5.1. Besides the requirements listed in Section 4, the experiments aim to measure the ease of use of each device in a classroom/lab setting, both for the instructors and the students.

In order to get the required 15-25 connections for every device, the ESP8266 microcontroller was chosen for this role, representing the connected sensors and actuators. A photo of the device can be seen in Figure 3.

The choice of the ESP8266 aligns with the context of IoT teaching, having a track record of great wireless support, range, and functions, featuring multiple digital and analog I/O pins, and 4MB of flash memory [21]. Moreover, the device is well-researched and highly supported in IoTempower. Even more, the device offers a low price (USD 2 on average) and low power requirements.

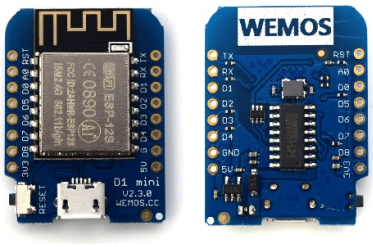

Figure 3. Wemos D1 mini (ESP8266) [22].

The device does lack some advanced features, such as Bluetooth, in comparison to its alternative – the ESP32. However, Bluetooth is not used in the experiments, so this feature is irrelevant to us. One feature that might be relevant, however, is 5G support, which the ESP8266 unfortunately lacks.

5.1 Experimental Setup

The first step of the experiments is to configure and ready the ESP8266 clients. For this thesis, I have acquired a total of 23 devices. Each device is inspected for issues and verified that it is working. Then, the following steps are taken:

1. Reset the device by pressing the RESET button.
2. Connect the device to a system running IoTempower, capable of flashing over a serial connection. For this thesis, a Raspberry Pi 4B+ with DietPi and the latest version of IoTempower is used.
3. Flash the ESP8266 with an IoTempower demo application. With the application, correct network settings are provided, mainly the network name and password of the access point to which to connect.

Upon boot, each ESP8266 finds the access point network and connects using the defined password. After the connection is established, a periodic "ping" message is sent from the device over MQTT to report that the device is alive and still connected. This is a default feature that IoTempower applications have and is beneficial to us for monitoring the live connections.

After completing the steps above, every client device is ready for experiments. By ensuring that every access point under consideration is using the same network name and password, the client devices do not have to be re-flashed throughout the experiments.

With the client devices ready, we move on to the actual experiments for every access point device in Table 1[16]. The aim is to configure each device into a simple minimal

[16]Except for the Lenovo Thinkpads, as I could not find suitable devices in time for the experiments.

state and attempt to set up an access point (using IoTempower or otherwise) to support at least 20 connected MQTT clients.

The experiment plan for every chosen device is as follows:

1. Obtain the device and reset all configuration options to factory state.
2. Read the documentation and configuration steps to start the device and establish a local network connection.
    - If possible – install and configure Linux on the device. Then, install software to enable access point and MQTT server functionality – in this case, install IoTempower.
    - If installing Linux and/or IoTempower is not possible, use a built-in method of creating an access point with a custom network name and password. In this case, the MQTT server will be operating from a different device.
3. Establish a stable connection to the system for monitoring and configuration purposes, preferably over SSH.
4. If possible and applicable – upload custom scripts for monitoring the amount of live connections[17]. Alternatively, use a different device on the same network for running the scripts, or monitor the number of connections through some built-in application or method.
5. Ensure the access point is configured properly and is enabled and running. Make sure an MQTT server is enabled, running, and reachable.
6. Connect ESP8266 devices to the host device and monitor the connections.
7. Document the findings and specify the number of client devices successfully connected and stable.
    - In case of failure/errors: observe `dmesg` logs to detect any issues on the kernel and firmware level. Document all findings.
8. In case the full amount of clients can not be established, debug the failure and attempt to find a reason for any limitations encountered.
    - If a workaround, patch, or any other remedy exists to increase the amount of connected clients – apply it, and re-do the test.

The outlined steps can be applied to most of the devices in Table 1, yet some, such as the Tenda router and other OpenWrt-based routers, can not fully realize all the steps due to the different architectures and the inability to run IoTempower.

Multiple scripts in Python and bash were developed to count the number of active connections to the device in access point mode and monitor the devices while testing. For this, the following Linux tools were used to get a list of connected hosts: `iw`, `ip`, and `arp-scan`, the output of which was further parsed and displayed.

[17]The scripts are accessible at this repository: `https://github.com/tonysln/msc`.

Moreover, a second approach was chosen in conjunction with gathering a list of connected hosts: listening to MQTT ping messages emitted by the ESP8266 devices. This approach is feasible for devices flashed using the default IoTempower configuration, as a special "ping" message is continuously transmitted by every active device. The script subscribes to the "#" topic from the MQTT broker and periodically processes all recent messages to report the devices recently seen. This script is built using Python and the `paho-mqtt` library, and is also available at the previously mentioned repository.

5.2 Experimental Procedures

In this section, I present the experimental procedures for each selected device, with comments on every step from the plan, as outlined in Section 5.1. I introduce each device and its specifications, describe the configurations I apply and the results I get after attempting to connect all of the ESP8266 client devices.

5.2.1 Raspberry Pi 4B+

Figure 4. Raspberry Pi 4B+. Michael H. („Laserlicht“), CC BY-SA 4.0, via Wikimedia Commons.

Acquired from Ulrich Norbisrath, used in the "Internet of Things" course kits. The specifications can be seen in Table 2 and a photo of the device in Figure 4.

According to my previous investigations of the Raspberry Pi in the role of an access point, as well as other numerous independent tests by developers, as outlined in Section 2.2, this Raspberry Pi model, in its default configuration, does not allow more than 8 concurrent connections in AP mode. The limitation stems from the latest versions of firmware for the Wi-Fi chip, as released by Cypress. One way to mitigate this problem is to use a special "minimal" firmware version developed specifically for the Raspberry Pi or use downgraded firmware versions from before this limitation appeared. I am testing both the default firmware and minimal firmware for this model.

I flashed DietPi v8 Bookworm on an empty SD card and installed it on the first boot. The choice of this OS comes down to its simplicity, ARM support, and small size. For better monitoring of the Wi-Fi chip, I enabled special debug messages by adding the following line to the kernel device tree settings: `brcmfmac.debug=0x100000`. The additional debug messages can be accessed using the `dmesg` tool.

Next, I downloaded and installed IoTempower on the system. For the internal Wi-Fi chipset to work, an "On-Board Wi-Fi" setting was enabled in the `dietpi-config` settings menu, which activated the wireless module. Finally, additional tools, such as `ufw`, `dnsmasq`, and `hostapd`, had to be installed on the system for IoTempower to function.

Table 2. Raspberry Pi 4B+ testing configuration.

1	**Architecture**	ARMv8 64-bit
2	**Chosen OS**	DietPi v8 Bookworm
3	**Wi-Fi chip**	Broadcom BCM2711
4	**IEEE standard**	802.11ac
5	**Client limit**	*Not mentioned*
6	**Manufacturer**	RPi Foundation, 2019
7	**Price**	From $35

After connecting to the Raspberry Pi over SSH, copying over, and starting the monitoring scripts, I created an AP using the `accesspoint` command in IoTempower, alongside an MQTT server, using `mqtt_broker`.

On the first try, only 7 devices connected. Various error messages were observed in the system logs (`dmesg`):

```
wl0: wlc_ap_authresp: out of scbs for bc:ff:4d:2a:7f:ee
wl0: wlc_recvfilter: scb is null, dont resend deauth
wl0: wlc_userscb_alloc: no SCBs available to reclaim
wl0: wlc_ampdu_recv_addba_resp: Failed. status 1 wsize 64 policy 1
```

This finding mirrors my previous observations, as well as the numerous other people who mentioned this issue on Raspberry Pi forums. Even so, this time, one less client could connect for some reason (instead of the more common 8 clients).

Despite the issue being recognized by developers of the Raspberry Pi [1], I spent additional time debugging it in an attempt to find the cause. When the 9th connection fails, it happens silently, as no error appears in hostapd logs. Only by activating the Broadcom debug messages do the aforementioned errors appear in the output of `dmesg`. Moreover, I used an external Wi-Fi adapter to monitor the packets sent between the Pi and ESP8266 clients using Wireshark, which did not reveal anything of interest. During the connection failure, no packets are sent out at all.

Researching the source code for the brcmsmac driver showed that a structure called SCB exists and might be relevant to the issue [23]. A new instance of this structure is created for every newly connected client, and it seems that the Wi-Fi module runs out of internal memory (inside the embedded ARM processor) after 8 clients connect, allowing no more clients to join. According to the comments of developer Phil Elwell under the minimal firmware file [1], the following features of the 802.11 protocol had to be disabled in order to increase the number of maximum clients supported: advanced roaming features, `dfsradar`, Auto Channel Support and Opportunistic Key Caching among others. These features are removed from the minimal firmware file [24].

I installed the minimal firmware file and immediately achieved better results. A total of 20 clients connected to the Raspberry Pi this time. However, after 16-17 clients, the following unidentified errors appeared in `dmesg` logs:

```
wl0: unable to find iovar "tdls_sta_info"
wl0: wlc_iovar_op: tdls_sta_info BCME -23 (Unsupported)
wl0: wlc_ampdu_recv_addba_resp: Failed. status 1 wsize 64 policy 1
```

Besides the minimal firmware, a downgraded version of the Broadcom firmware can be installed to increase the client limit. If the limit is caused by new features of the 802.11 protocol consuming more memory and leaving less available to allocate for connected clients, then downgrading to a version with fewer 802.11 features is a potential solution. Information on this can be found in a GitHub Issue discussion post for the Internet-in-a-Box (IIAB) project repository [25]. Users and developers of that project discussed the same issue we faced with the low client limit on the Raspberry Pi and provided links to old versions of firmware to try. I downloaded and installed the following firmware versions: `2015-03-01_7.45.18.0_ub19.10.1` and a newer `2017-10-23_7.45.132`, which also required a special CLM blob file, for which I used one with version `2018-02-26_rp`. The CLM file defines the Country Locale Matrix, which is usually included inside the firmware file but can also be loaded separately.

With the first downgraded firmware (from 2015), all 23 clients connected successfully. As such, the upper limit could not be established. The IIAB discussions mention 32 as the maximum number of supported clients with this version [25].

Using the second downgraded firmware file (from 2017), only 15 clients could connect, which is better than using the default firmware but worse than minimal firmware.

It is important to point out that using an outdated version of Wi-Fi firmware might pose a serious security risk, as older versions do not have the latest security patches, bug fixes, and updates, which puts the whole network at risk, even when using it strictly in a classroom demo setting.

In total, the process of configuring the Raspberry Pi is reasonably straightforward, although some additional packages have to be manually installed and modules enabled on the DietPi due to its super lightweight nature. When trying to set an AP, no information about the low client limit can be easily found beforehand, as there are no disclaimers about this in the Raspberry Pi documentation. Even with the minimal firmware installed, only 20 clients can be connected, which might be too low for some projects. Using an outdated firmware version does increase this limit further, although it comes with potential critical security risks.

5.2.2 Raspberry Pi 3B

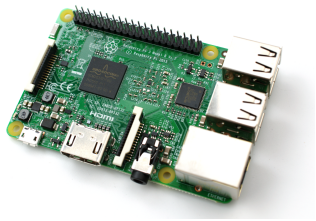

Figure 5. Raspberry Pi 3B. Gareth Halfacree from Bradford, UK, CC BY 2.0, via Wikimedia Commons.

Acquired from Ulrich Norbisrath. The specifications can be seen in Table 3 and a photo of the device in Figure 5.

Similar to the Raspberry Pi 4B+, the default configuration supports only a limited number of connected clients. Unfortunately, the architecture of the Wi-Fi chipset on this model does not allow the use of the minimal firmware version. However, downgraded firmware versions can still be used for the experiment.

Table 3. Raspberry Pi 3B testing configuration.

1	**Architecture**	ARMv8 64-bit
2	**Chosen OS**	DietPi v8 Bookworm
3	**Wi-Fi chip**	Broadcom BCM43438
4	**IEEE standard**	802.11n
5	**Client limit**	*Not mentioned*
6	**Manufacturer**	RPi Foundation, 2016
7	**Price**	$35

The testing procedure for this model was very similar to that of 4B+. DietPi v8 Bookworm was installed and set up. Special debug messages were enabled by adding the following line to the kernel device tree settings: `brcmfmac.debug=0x100000`.

Next, I downloaded and installed IoTempower on the system. Custom monitoring scripts were then copied over to the device to observe the connected clients.

The "On-Board Wi-Fi" setting was enabled in the `dietpi-config`. Subsequently, additional tools such as `ufw`, `dnsmasq`, and `hostapd` were installed. After establishing a connection to the device via SSH and initiating the monitoring scripts, I created an access point using the `accesspoint` command in IoTempower, alongside an MQTT server.

Initially, a total of 10 clients connected to the device. Any remaining connections failed in a manner similar to that of the model 4B+.

As the minimal firmware version is not available, only downgraded firmware versions can be tested. Using the IIAB discussion post [25], I first downloaded and tested the

firmware version `2017-10-23_7.45.98.38_ub19.10.1_ub20.04`, which did not bring any improved results, with the client limit staying at 10.

Following this, I moved on to firmware version `2018-09-11_7.45.98.65`, with the CLM file version `018-09-11_7.45.98.65`, which resulted in a total of 19 connections. Further, I switched out the CLM file for a newer version `2020-02-16_7.45.98.97`, which worked even better, allowing 20 clients to be established.

In the end, the process of configuring model 3B was similar to the Raspberry Pi 4B+, although due to its lower processor speed, completing all of the steps took more time. Moreover, as minimal firmware is not available for this device, the 10-client limitation puts this model at a great disadvantage for IoT classroom usage.

5.2.3 Raspberry Pi Zero W

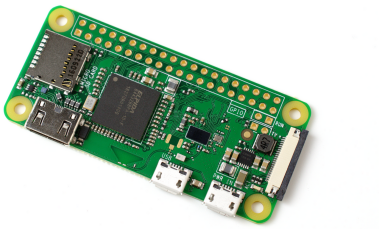

Figure 6. Raspberry Pi Zero W. Gareth Halfacree from Bradford, UK, CC BY 2.0, via Wikimedia Commons.

Acquired from Ulrich Norbisrath. The specifications can be seen in Table 4 and a photo of the device in Figure 6.

Table 4. Raspberry Pi Zero W testing configuration.

1	**Architecture**	ARMv6 32-bit
2	**Chosen OS**	DietPi v8 Bookworm
3	**Wi-Fi chip**	Broadcom BCM2835
4	**IEEE standard**	802.11n
5	**Client limit**	*Not mentioned*
6	**Manufacturer**	RPi Foundation, 2017
7	**Price**	$10

The architecture of this model is very similar to that of model 3B, so the testing procedure is nearly identical. The same downgraded firmware versions are also used due to the Wi-Fi chipset similarity, as minimal firmware is once again not available.

DietPi Linux v8 was installed and configured. Next, I downloaded and installed IoTempower, together with any missing packages. The on-board Wi-Fi setting was also enabled for DietPi. After establishing a connection to the Pi Zero over SSH, monitoring

scripts were copied over and launched. After starting the access point and MQTT broker programs in IoTempower, clients could be connected.

First, only nine clients connected to the board. The remaining clients failed analogously to the models 4B+ and 3B.

Because minimal firmware is not available for this model, downgraded firmware versions were used once again, acquired from the IIAB discussion [25]. First, I downloaded and installed firmware version `2017-10-23_7.45.98.38_ub19.10.1_ub20.04`, which did not result in a change, with the client limit staying at 9.

Then, using a slightly newer firmware version `2018-09-11_7.45.98.65`, alongside a CLM file version `018-09-11_7.45.98.65`, a total of 19 connections could be established. Finally, I switched out the CLM file for a newer version `2020-02-16_7.45.98.97`, which resulted in an increase of up to 20 connections, just like with the 3B.

The results of this Raspberry Pi model follow the 3B closely, albeit with one less client able to connect initially. While the price of this particular model is much lower than that of the other Raspberry Pi models, a very important thing to note is that this model is extremely slow (due to a weaker processor), so completing any kind of task takes a shockingly long amount of time. As such, using this for IoT education is definitely not recommended.

5.2.4 Raspberry Pi 5

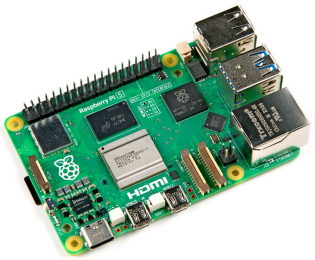

Figure 7. Raspberry Pi 5. SparkFun Electronics, CC BY 2.0, via Wikimedia Commons.

Acquired from Ulrich Norbisrath. The specifications are in Table 5 and a photo of the device in Figure 7.

This is the latest addition to the Raspberry Pi model line-up as of writing this thesis. The Wi-Fi chipset inside is based on the same chip as in the 4B+ model. As such, the testing procedure is also similar to the one of 4B+, including the additional minimal firmware testing part.

For this device, a beta version of DietPi was used, as a full version was not yet available. After installing the beta version of DietPi, IoTempower was downloaded and installed alongside the additional packages required to run it. The On-Board Wi-Fi setting had to be enabled as well, mirroring the setup of the other Pi models.

After connecting to the Pi over SSH, launching the monitoring scripts and starting the AP and MQTT server applications, clients could be connected.

Initially, only seven clients were able to connect to the device. After activating the minimal firmware, the limit increased to 20.

Table 5. Raspberry Pi 5 testing configuration.

1	**Architecture**	ARMv8.2 64-bit
2	**Chosen OS**	DietPi v8 Bookworm Beta
3	**Wi-Fi chip**	Broadcom BCM2712
4	**IEEE standard**	802.11ac
5	**Client limit**	*Not mentioned*
6	**Manufacturer**	RPi Foundation, 2023
7	**Price**	From $60

With the old firmware version `2015-03-01_7.45.18.0_ub19.10.1`, at least 23 clients could be connected, and with version `2017-10-23_7.45.132` only 15 connections were established.

The performance of Raspberry Pi 5 resembles that of 4B+, albeit with much faster processing speeds. Still, the same issues persist, and the only solution to the client limit seems to be to use the minimal firmware with a limit of 20 or a potentially unsafe outdated version of firmware. Also, the price of Raspberry Pi model 5 is the highest of all the models so far, making it a questionable choice over the 4B+ and even 3B.

5.2.5 Realtek RTL8188CUS

Instead of relying on the built-in Wi-Fi module of the Raspberry Pi, an external USB adapter can be used to potentially fix the client limit and improve performance. For this test and the following USB adapter tests, I will be using the Raspberry Pi 4B+ as the base model, connecting the USB adapters to it, and running the access point using them as the wireless device.

To detect the connected USB device, `lsusb` and `iwconfig` commands were used. One modification was made in the IoTempower configuration file – instead of the default wireless device name ("wlan0" in my case), the newly connected USB adapter was selected instead. In this case, the new name of the adapter was "wlan1".

After launching the accesspoint program, I received the following warning message:

```
WARN: Realtek drivers usually have problems with WPA1, enabling -w 2
```

Despite the warning, the access point program started successfully, and a total of 7 devices connected to the adapter. When attempting to connect any more devices, the following messages were observed in `dmesg` logs:

```
[ 1411.429708] rtlwifi: ------hwsec_cam_bitmap: 0x0 entry_idx=4
[ 1411.573655] rtlwifi: ------hwsec_cam_bitmap: 0x10 entry_idx=5
[ 1411.605627] rtlwifi: ------hwsec_cam_bitmap: 0x30 entry_idx=6
[ 1411.625496] rtlwifi: ------hwsec_cam_bitmap: 0x70 entry_idx=7
[ 1411.645494] rtlwifi: ------hwsec_cam_bitmap: 0xf0 entry_idx=8
[ 1411.661570] rtlwifi: ------hwsec_cam_bitmap: 0x1f0 entry_idx=9
[ 1411.697499] rtlwifi: ------hwsec_cam_bitmap: 0x3f0 entry_idx=10
```

These messages were displayed for every additional client that attempted to establish a connection, failing to do so. When searching for more information regarding this error message, I discovered the rtlwifi-new driver repository[18], in which the file `cam.c` contains a function called `rtl_cam_get_free_entry`. The error message(s) we received originated from that function, caused by failed attempts at allocating new CAM objects for connected clients. From this, we can assume that the CAM object is an analogous structure to the SCB object in Broadcom drivers, as referenced in the `brcmsmac` driver.

Additionally, I made an attempt to change the underlying driver used by hostapd. For this, I navigated to the `/iot/.local/external/create_ap` folder and edited the `create_ap.conf` file by changing the "DRIVER=" value from nl80211 to rtl871x. Unfortunately, this attempt did not change the client limit or the behavior of the adapter.

5.2.6 Ralink MT7601U

This is a different USB adapter, this time featuring a Ralink chip inside. As the device name for this chip was confirmed to be "wlan1" again (using `lsusb`), the accesspoint program could be simply re-started without further adjustments.

After attempting to launch the accesspoint program with the Ralink USB adapter selected, I received the following error:

```
Your adapter does not support AP (master) mode
```

I attempted to enable AP mode (also called master mode) using the `iwconfig` utility but got the following error:

```
Error for wireless request "Set Mode" (8B06):
    SET failed on device wlan1 ; Invalid argument.
```

After some more research, it became clear that this adapter does not support AP mode natively. While there do exist some projects which attempt to add support for this driver [26], setting these projects up is not as straightforward (requiring a platform specific build process), and integration with IoTempower might not be possible. As such, I decided to move on to the next adapter.

5.2.7 Realtek RTL8188SU

Analogously to the previous USB adapters, I tested another Realtek-based adapter, this time with a slightly different chipset to the adapter in Section 5.2.5.

This adapter was also recognized by the system and activated under the name "wlan1". After attempting to restart the accesspoint script, I noticed the following error, which prevented the AP from starting:

```
nl80211: Could not configure driver mode
nl80211: deinit ifname=wlan1 disabled_11b_rates=0
nl80211 driver initialization failed.
wlan1: interface state UNINITIALIZED->DISABLED
wlan1: AP-DISABLED
wlan1: CTRL-EVENT-TERMINATING
hostapd_free_hapd_data: Interface wlan1 was not started
```

[18] `https://github.com/rtlwifi-linux/rtlwifi_new`

Attempts at using rtl871x as the driver instead of nl80211 did not help in this case. Further attempts to enable master mode using the `iwconfig` tool resulted in the same error as before, detailed in Section 5.2.6:

```
Error for wireless request "Set Mode" (8B06):
    SET failed on device wlan1 ; Invalid argument.
```

Similarly to the Ralink adapter, I concluded that AP mode is not natively supported, and enabling it would require the use of possibly outdated drivers that are not supported by the latest versions of the Linux kernel.

In the end, using the USB Wi-Fi adapters did not prove to be a viable alternative to using the built-in adapters of the Raspberry Pi models. Considering that there are many more adapters on the market, it is likely that some of them will perform much better and will support more clients. As such, further experiments are recommended for future research, specifically targeting low-cost USB adapters.

5.2.8 Atomic Pi

Figure 8. Atomic Pi [27].

Acquired from Ulrich Norbisrath. Specifications can be seen in Table 6 and a photo of the device in Figure 8.

The Atomic Pi is an alternative to the Raspberry Pi boards, with a more powerful processor and different architecture, featuring an Intel Atom chip inside. Besides this, a Ralink Wi-Fi module is utilized on the board for Wi-Fi functionality.

Table 6. Atomic Pi testing configuration.

1	**Architecture**	x86-64
2	**Chosen OS**	Debian
3	**Wi-Fi chip**	MediaTek (Ralink) RT5572
4	**IEEE standard**	802.11n
5	**Client limit**	*Not mentioned*
6	**Manufacturer**	ameriDroid
7	**Price**	From $40

I installed Debian Bookworm as the OS on this device, mainly due to its compatibility with the hardware and ease of use. Then, I installed and configured IoTempower. There were no problems completing these steps, so the accesspoint program was launched alongside an MQTT server.

In total, 23 clients connected to the Atomic Pi without issue, as no errors could be observed in `dmesg` logs. As such, no upper client limit can be established for this device with this experiment. However, the Atomic Pi became visibly slow to respond after the 20th client had connected, and any new clients took a longer time to form the connection.

Despite the worsened performance towards the end, the Atomic Pi performed well and was simple to set up. No special fixes have to be activated for all clients to be supported. Considering that the price of the Atomic Pi is similar to that of Raspberry Pi models 3 and 4, this device is a promising alternative to the Pi.

5.2.9 HP EliteBook 840 G1

Figure 9. HP EliteBook 840 G1. Elias Ojala, CC BY-SA 4.0, via Wikimedia Commons.

Acquired from Ulrich Norbisrath. Specifications can be seen in Table 7 and a photo of the device in Figure 9.

In addition to low-powered devices such as the Raspberry Pi and Atomic Pi, older (used) laptops can be found for quite low and reasonable prices. The HP EliteBook 840 G1 is the first in the series of old, cheap laptops that I will test, and is potentially an excellent replacement to the devices I have tested so far.

Table 7. HP EliteBook 840 G1 testing configuration.

1	**Architecture**	x86-64
2	**Chosen OS**	Debian
3	**Wi-Fi chip**	Broadcom BCM43228
4	**IEEE standard**	802.11n
5	**Client limit**	*Not mentioned*
6	**Manufacturer**	Hewlett-Packard
7	**Price**	From $100

The installation procedure of this laptop was similar to that of the Atomic Pi. I installed Debian and IoTempower with all the missing required packages. Next, attempting to run the accesspoint command resulted in errors regarding AP mode not being supported, similar to the Ralink MT7601U USB adapter, detailed in Section 5.2.6.

As an additional attempt, I connected the Realtek RTL8188CUS USB adapter to the laptop and selected it as the wireless interface. After doing so, the accesspoint command worked, and I activated the AP alongside an MQTT broker. Surprisingly, 14 clients connected to the adapter this time, with the remaining clients failing to do so. In comparison to the same Realtek adapter being used with a Raspberry Pi 4B+, 7 more clients were able to connect. Overall, the experience of using this laptop was similar to that of the Atomic Pi, except that the device could not be used in AP mode by default, and an external adapter had to be used.

5.2.10 HP EliteBook 840 G5

Acquired from Ulrich Norbisrath. The specifications of this laptop can be seen in Table 8.

This is a much newer version of the previous EliteBook 840 laptop. This laptop model was used for the Internet of Things course in Spring of 2024, and it was acquired already pre-configured and ready for the experiment.

Table 8. HP EliteBook 840 G5 testing configuration.

1	**Architecture**	x86-64
2	**Chosen OS**	Manjaro
3	**Wi-Fi chip**	Intel(R) Wireless-AC 8265
4	**IEEE standard**	802.11ac
5	**Client limit**	*Not mentioned*
6	**Manufacturer**	Hewlett-Packard
7	**Price**	From $275

This laptop features an Intel Wi-Fi chipset instead of the Broadcom module and uses the iwlwifi driver. For the OS, Manjaro Linux was pre-installed (with a GUI), and as a novel feature, NetworkManager (NM) was used for AP functionality instead of the default hostapd-based `accesspoint` script. Using NM with this setup was simpler and more accessible for students, as a GUI tool was available to make the AP creation process more streamlined. One additional change had to be made to the created connection for the ESP8266 clients to work: the PMF (Protected Management Frames) feature had to be disabled, or else the clients would refuse to connect.

The experimentation process took place during an Internet of Things class on 16.04.2024, with the help of multiple groups of students. All students used this laptop model and had the same NetworkManager setup activated. All ESP8266 client devices were also flashed with a demo IoTempower application. After starting the MQTT broker in IoTempower, testing could begin.

All groups of students achieved the same result: only 11 clients connected to the AP,

with the remaining clients failing to connect. Searching for this problem online, forum posts can be found from other users facing a similar issue [28] [29], with some suggesting that the limitation is caused by the Intel Wi-Fi chip itself, suggesting that no workaround or solution exists.

5.2.11 HP EliteBook 840 G6

Acquired from University of Tartu. The specifications can be seen in Table 9. This is an updated model of the EliteBook 840 series, featuring a newer Intel Wi-Fi module inside.

Table 9. HP EliteBook 840 G6 testing configuration.

1	**Architecture**	x86-64
2	**Chosen OS**	DietPi v8 Bookworm UEFI PC version
3	**Wi-Fi chip**	Intel(R) Wireless-AC 9560
4	**IEEE standard**	802.11ac
5	**Client limit**	*Not mentioned*
6	**Manufacturer**	Hewlett-Packard
7	**Price**	From $300

On this laptop, DietPi v8 Bookworm UEFI PC version was installed as the OS. Then, IoTempower was downloaded and installed. To enable the Wi-Fi modules, the "On-Board Wi-Fi" setting was enabled in the `dietpi-config` menu. Finally, missing packages were installed: `ufw`, `dnsmasq`, and `hostapd`. After establishing a connection to the device over SSH to upload the monitoring scripts, the access point program was activated alongside an MQTT broker.

A total of 11 connections were created to the laptop. After the 11th client, the following error messages were displayed for all additional clients (as seen in `dmesg` logs):

```
ap0: STA bc:ff:4d:2a:e0:5e IEEE 802.11: Could not add STA to kernel driver
```

Searching for this problem online revealed forum posts with other people facing an issue like this, also using an Intel Wi-Fi module and Linux [30]. The forum posts speculate that the problem is in the hardware.

As an additional experiment step, I installed an alternative access point solution, NetworkManager, and configured it with the same settings as the IoTempower `hostapd` access point. This step required significant troubleshooting, as simply starting the newly installed NM was not enough, and changes to the system configuration had to be made. Specifically, the wireless interface had to be "released" from the system so that NetworkManager could take full control of it.

Moreover, MFP (Management Frame Protection) was turned off to allow the ESP8266 clients to establish new connections. Despite these changes, no more than 11 clients could be connected at a time and all the previous error messages in `dmesg` could be observed. Together with the EliteBook 840 G5, it seems that the client limit is a part of the Intel Wi-Fi chip family and can not be circumvented easily.

The last few experiments with laptops showed us that each laptop should be carefully considered, as limitations can be present even inside them. Intel Wi-Fi modules are not recommended, as the limitation inside them does not appear to be officially recognized, and no workaround method (such as the minimal firmware in the case of some Broadcom chips) exists. Still, USB Wi-Fi adapters can be easily connected for an increased client limit in some cases, and the general experience of using a laptop is far more suitable for classroom usage due to the presence of a screen and keyboard.

5.2.12 GL.iNet GL-MT300N-v2

Figure 10. GL.iNet GL-MT300N-v2. gl-inet.com

Acquired from Ulrich Norbisrath. The specifications can be seen in Table 10 and a photo of the device in Figure 10.

This device is different from the previously tested devices due to its architecture and primary use case – the GL-MT300N-v2 is a "travel router" with an MIPS processor inside. The device supports OpenWrt and is specifically tailored to networking applications.

Table 10. GL-MT300N-v2 testing configuration.

1	**Architecture**	MIPS
2	**Chosen OS**	OpenWrt 23.05.0
3	**Wi-Fi chip**	MediaTek MT7628AN
4	**IEEE standard**	802.11n
5	**Client limit**	39 [31]
6	**Manufacturer**	gl.iNet
7	**Price**	From $25

I connected the device to my local network over Ethernet, then connected to the device's Wi-Fi network and navigated to the web interface (by default, accessible at URL `http://192.168.8.1`). Using the interface, I installed OpenWrt version 23 onto the device for increased flexibility[19] and better configuration options. With the new OS installed, I downloaded the package "mosquitto-nossl" to enable MQTT server functionality right from the router itself. Moreover, a plugin for configuring MQTT over the LuCI web interface was installed, and the following settings were applied:

[19] OpenWrt is an open-source OS for routers: `https://openwrt.org/`.

- Anonymous connections: Allowed
- MQTT port: 1883, open

Additionally, the most verbose logging setting (level 0) was enabled by creating an SSH connection to the router and running the following commands:

```
uci set wireless.radio0.log_level=0
uci commit wireless
wifi up
```

By doing this, the most detailed debug messages are written to the system log file, which is also accessible through the LuCI web interface.

Finally, I specified the required network name and password using the router's web interface. After saving the settings, connections could be made to the router.

In total, 23 clients successfully connected and were confirmed to be alive through the OpenWrt web interface. The router's performance remained consistent throughout the experiment, so I believe many more clients could be connected. This aligns with the experiment conducted by gl.iNet, where it was determined that at least 39 clients can be successfully connected [31].

In the end, setting up and using this router was reasonably straightforward and fast, as all of the operations could be completed through the web interface. However, the interface might still be confusing to a student with no prior experience.

Moreover, the router did not present any limitations and even allowed for an MQTT server to be created. All in all, this device looks to be a promising replacement for the Raspberry Pi or Atomic Pi boards, or a companion to them, given that all of the networking functionality can be handled by the router.

5.2.13 Hi-Link HLK-7628N Kit

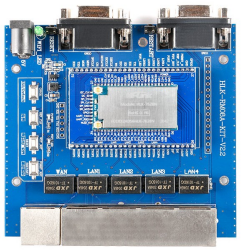

Figure 11. Hi-Link HLK-7628N kit [32].

Acquired from Ulrich Norbisrath. The specifications can be seen in Table 11 and a photo of the device in Figure 11.

This device features the exact same processor as the previously tested GL-MT300N-v2 but on a different board. It can be purchased for a very low price from online retail stores such as AliExpress. This particular device features a combination of a processor and a board with various connections.

Table 11. Hi-Link HLK-7628N kit testing configuration.

1	**Architecture**	MIPS
2	**Chosen OS**	OpenWrt 23.05.1
3	**Wi-Fi chip**	MediaTek MT7628AN
4	**IEEE standard**	802.11n
5	**Client limit**	*Not mentioned*
6	**Manufacturer**	Hi-Link
7	**Price**	From $20

The experiment for this device mirrored the experiment in Section 5.2.12 closely. OpenWrt version 23 was installed on this device. Then, the plugin "mosquitto-nossl" was installed to enable MQTT broker functionality. The plugin was configured to allow anonymous connections, and port 1883 was left open for MQTT. Finally, network credentials were configured in the settings menu, and the experiment could begin.

All 23 clients connected without issue. The board did not slow down or become unresponsive, so more clients could most likely connect. In general, the performance of this board was similar to that of the GL-MT300N-v2, which is not surprising, considering they share the same Wi-Fi module.

Considering that this device is even cheaper (especially if purchasing in bulk) than the GL.iNet router, it proved itself as a suitable alternative to the Raspberry Pi boards and a solid choice for classroom usage, but only if some additional instructions are provided to students.

5.2.14 Android Phone

Smartphones are ubiquitous, with many running Android, which is based on Linux. Given that nearly all smartphones are equipped with Wi-Fi-capable chips, they potentially serve as suitable candidates for makeshift access points.

To explore this possibility, I acquired a Samsung Galaxy A5 (SM-A520F) running LineageOS 16 (Android 9) and tested its AP capabilities. This phone is powered by an Exynos 7880 chip. I configured the AP through the system settings, where I easily set the network name and password with no more settings available to edit. After creating the network, I began connecting client devices. In total, 10 clients connected before the device refused to accept additional connections. No errors could be observed for the failed connections.

As an alternative, the terminal emulator Termux[20] can make Android resemble Linux even more closely. Participants in the Internet of Things course at UT collaborated with me on this experiment, using a smartphone with Android 13 installed and Termux. They installed IoTempower and attempted to run various access point scripts. However, they encountered significant difficulties. Further research by the students revealed that Android 13 has implemented security measures that restrict many Linux networking

[20] `https://termux.dev/en/`

utilities, even within Termux. It appears that creating an AP outside the phone's default AP mode is impossible due to these security limitations, so no additional experiments could be run.

The combination of an MQTT broker (launched in Termux) and the default AP solution remains a viable option, although with the limitation of just 10 clients.

5.2.15 Tenda TX2 Pro

Figure 12. Tenda TX2 Pro [33].

Acquired from Danielle Morgan. The specifications can be seen in Table 12 and a photo of the device in Figure 12.

Table 12. Tenda TX2 Pro testing configuration.

1	**Architecture**	MIPS/RISC-V
2	**Chosen OS**	Unknown
3	**Wi-Fi chip**	Realtek RTL8197H + RTL8832BR
4	**IEEE standard**	802.11ax
5	**Client limit**	*Not mentioned*
6	**Manufacturer**	Tenda, 2023
7	**Price**	From $40

This is a conventional modern router for home use, offered at a reasonably low price. It features multiple Realtek chipsets inside to handle the Ethernet, 2G, and 5G Wi-Fi functionality separately. Unfortunately, there is not a lot of information available regarding its operating system and architecture. As such, OpenWrt could not be easily installed on this router. However, the device does feature a web interface for easy configuration.

I configured the router by connecting it to my local network using Ethernet, connecting to its Wi-Fi network using the provided credentials, opening the web interface, and entering my desired network name and password. After this, the router was configured and ready to be used. As there is no support for MQTT on the router, the MQTT server had to be launched from a different device on the same network (Raspberry Pi 4B+ was

used for this test). Additionally, the static IP address of the router was reserved on the local network for the configuration to work.

After activating the clients, all 23 of them successfully connected to the router and were observed over MQTT. The router seemed stable, so most likely, many more clients could be connected to it. All in all, using this router was not as straightforward as running an OpenWrt router or some other device with the MQTT server built into it. The additional setup steps are not ideal for classroom usage.

5.2.16 ESP32

As a final option, the ESP32 microcontroller was considered for the role of an access point. The ESP32 is a more advanced alternative to the ESP8266, featuring a more powerful processor and more functionality while still being much less powerful than the Raspberry Pi boards. A photo of the device can be seen in Figure 13. The average price of an ESP32 is USD 2-5.

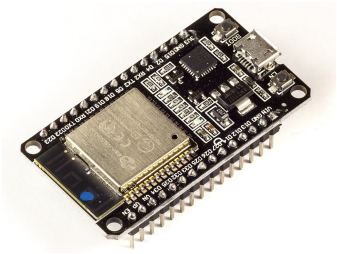

Figure 13. ESP32. Ubahnverleih, CC0, via Wikimedia Commons.

There are multiple ways to program the ESP32, but the two main approaches are to either use the Arduino IDE and libraries created for the ESP32 or write a program from scratch using the ESP-IDF framework[21], the latter of which is much more labor intensive but produces compact and fast code.

For this experiment, I chose the first approach of using Arduino. I utilized the "WiFi" library, available as part of the "esp32" device library [34], and used the "WiFiAccessPoint" example program, created by Elochukwu Ifediora [35], as the base program.

The base program was configured by setting the desired network name and password and then uploaded to the ESP32.

Initially, only four clients connected to the device. After some research, I discovered that there is a limit in place caused by the `max_connected` variable in the `softAP` function call. I updated the variable to 32 and uploaded the updated program onto the ESP32.

With the updated program, 10 clients connected connected to the ESP32. However, the last few clients established the connection with a great delay (over 3 minutes for each new connection), hinting at performance issues with the ESP32.

In total, I did not find the ESP32 to be as reliable as many of the other devices under consideration, and I do not recommend it for classroom usage due to the low client limit and performance problems.

[21] `https://docs.espressif.com/projects/esp-idf/en/stable/esp32/`

6 Access Point Configuration Tool

In this section, I present my custom solution for creating an access point on a Linux system with IoTempower. The requirements from Section 4 are considered and utilized to handle different Wi-Fi chip architectures and limitations differently.

The proposed solution is a comprehensive system consisting of a set of modular scripts and programs, along with a primarily text-based User Interface (TUI). The system has integration with IoTempower to utilize the existing (separate) scripts for launching an access point, configuring an external OpenWrt router, launching an MQTT broker, and detecting the connected MQTT devices.

The final product is a unified interface with multiple modules:

- Detection of the current platform and architecture, with a focus on identifying the Wi-Fi chipset and firmware.
- Compatibility check of the present Wi-Fi chipset for AP mode.
- Options to apply patches or fixes to the system to overcome known limitations, enhancing support for maximum connected clients, speed, or other factors.
- Intuitive UI for easy configuration of the system, with provided instructions and information.
- The ability to port the system to environments outside the command line, such as a web browser or desktop application.

The completed implementation supports both x86 and ARM architectures, and the main target operating system is Linux, as macOS and Windows are not fully supported due to several limitations. Recent Windows versions have disabled support for SoftAP due to a new driver model [36], and both Windows and macOS lack some critical system tools used in this application, such as `lshw` and `lspci`. Configuring an access point on these systems is only possible through built-in system dialogs and applications using their respective interfaces.

The completed system is designed to have minimal external library dependencies, relying on commonly found UNIX tools and libraries instead. The user interface of the system is developed using Python and the Textual UI framework[22]. The choice of Textual is based on its robust platform support, compatibility, popularity, comprehensive documentation, and a simple API.

6.1 Implementation

The completed application is composed of multiple Python and bash scripts, each serving a distinct purpose. The bash scripts are responsible for system information gathering and interaction with IoTempower, while the Python scripts manage the user interface. The core idea is to utilize various tools to collect data on system components, log messages, and network information. The gathered information is then processed by either parsing known output structures or searching for specific keywords to identify Wi-Fi chips and

[22] `https://textual.textualize.io/`

features. This method allows the application to support a broad range of hardware and systems.

A crucial feature of the final system is its seamless integration with the IoTempower framework. If the IoTempower environment is not active (typically activated using the `iot` command after installation), the application alerts the user, prompting them to activate the environment before proceeding.

The following sections describe the various modules of the completed application and their respective roles.

6.1.1 Interface

This is the main module of the application and is responsible for drawing the UI, managing calls to background processes, and receiving all input from the user. Using the interface, users can set up an access point, get status information, read about system specifications, and monitor the number of live connections.

The completed interface is simple and has a familiar feeling, resembling a classical GUI. This ensures that students have no issue recognizing the functionality of the interface and can traverse it easily without reading prior documentation. The interface can be navigated using both a keyboard and a mouse.

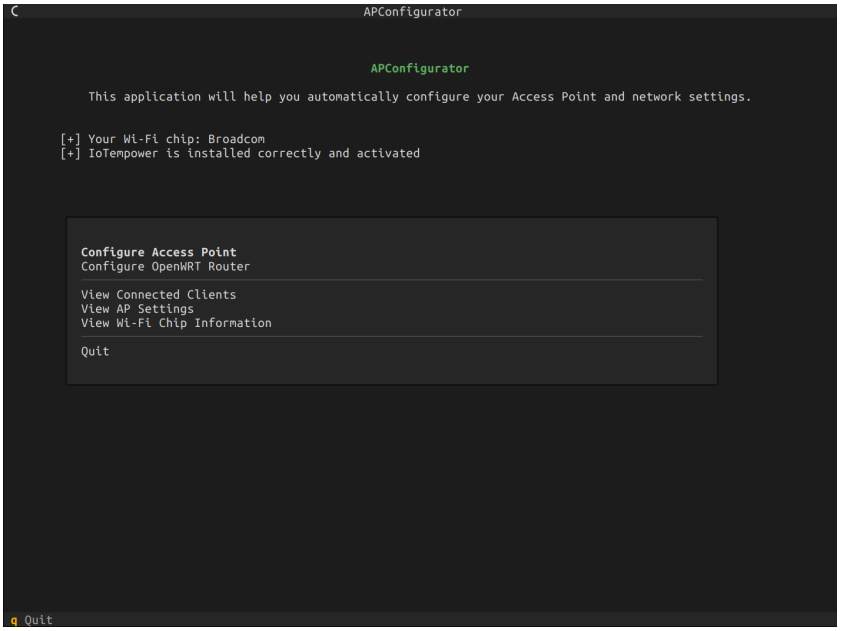

Figure 14. Main menu of the completed application.

A screenshot of the main menu can be seen in Figure 14, captured the way a user would view the application. This screen features an introductory paragraph with a welcome message, status indicators of the detected Wi-Fi chipset and system information, and a main menu with clickable options. Additionally, a header (title bar) and footer are present, closely resembling a traditional GUI. Choosing any of the menu items will open a new "screen" on top of the current one, with the option to return to the menu or quit at any moment while using the app.

If an active access point is detected after launching the program, an additional entry is added to the menu to disable the already running AP. The user is asked to re-launch the application for changes to take effect.

6.1.2 Wi-Fi Module Detection

This module is responsible for detecting the Wi-Fi chipset make, model, features and limitations on the given system. For this, Linux tools `lshw, lspci, lsusb, iwconfig` and `dmesg` are used to get information about the system. Specifically, the hardware configuration of the system, PCI devices, USB devices, and network configuration settings are analyzed. From the output of these tools, special keywords, representing the names of common Wi-Fi drivers (such as "ath9k" and "brcm") are searched, and any matched keywords are saved. Further, additional parsing is done to extract Wi-Fi chipset vendor and driver information.

There is no separate screen for this module. Instead, the returned string is used in other modules of the application.

6.1.3 Local Access Point Configuration

This module configures a local access point using the provided settings and applies any necessary fixes to the Wi-Fi chip, such as the Broadcom minimal firmware adjustment. The access point can be set up using one of two software solutions: hostapd or NetworkManager (NM). While hostapd is the default option, users can opt for NM if preferred. The local AP configuration screen can be seen in Figure 15.

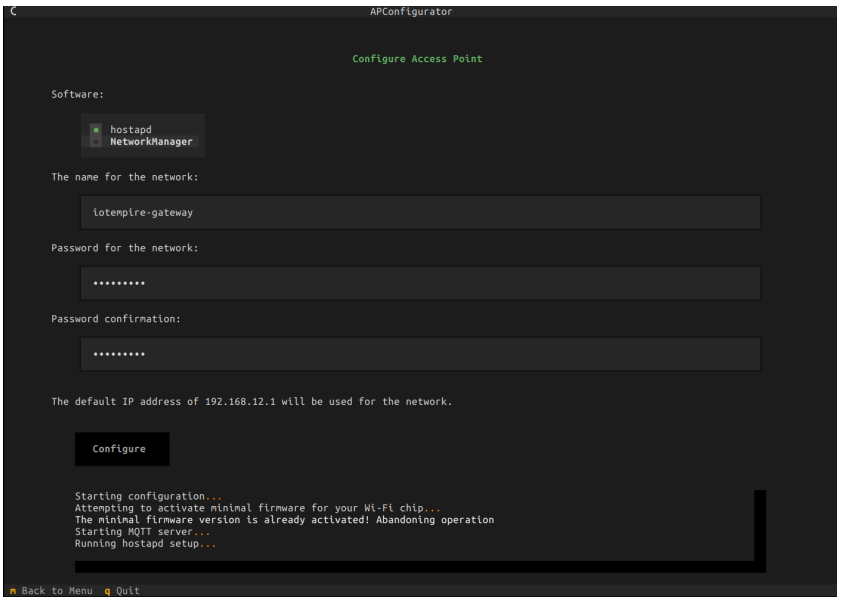

Figure 15. Local access point configuration screen.

Upon selecting the Local AP Configuration option in the app menu, a new screen appears with simple fields to choose the software back-end (hostapd or NM), enter the network name, and set the password (entered twice for confirmation). Once the user

has completed these fields, the information is verified and saved into an IoTempower configuration file. A pre-configured MQTT server is then automatically started, followed by the selected SoftAP system being initiated using the corresponding IoTempower scripts, ensuring all necessary variables are passed along. Any additional instructions and information are printed in the log area on the bottom of the screen.

Switching between hostapd and NetworkManager can be challenging if the system was initially configured with hostapd, as the `wpa_supplicant` tool (used by hostapd) needs to be disabled and the wireless device released for NM to take control. This process is not intuitive, particularly for beginners. As such, the configuration application attempts to automate these adjustments, allowing users to switch between hostapd and NM seamlessly. Multiple configuration files are modified to release the wireless device from the system, allowing NM to manage it. In this case, a reboot may be necessary after setting up an AP.

6.1.4 OpenWrt Router Configuration

If the built-in Wi-Fi chipset of the system does not allow for a sufficient amount of clients and there are no fixes available, the system might recommend the user to find an external OpenWrt-compatible router to use instead. An external router can be configured using this application by choosing the second option in the main menu. The OpenWrt router configuration screen can be seen in Figure 16.

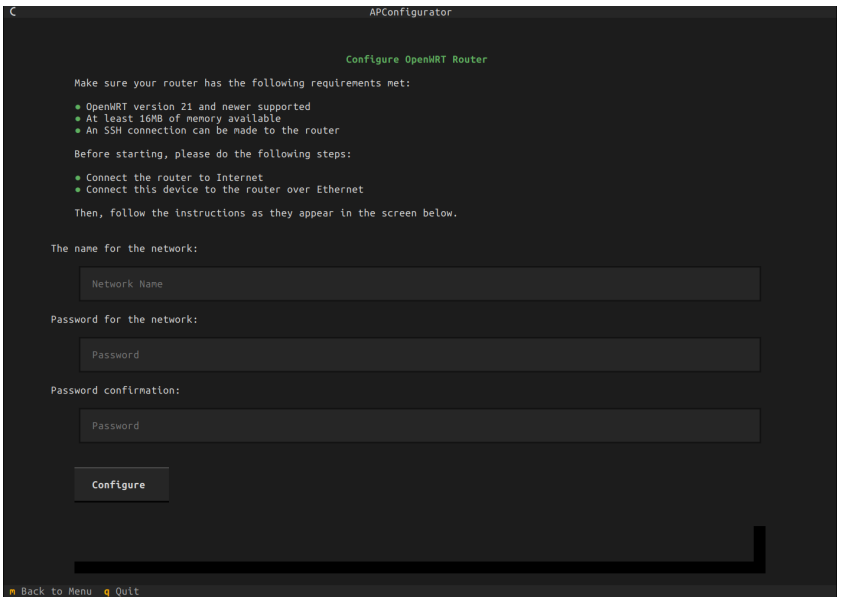

Figure 16. External OpenWrt router configuration screen.

This module uses the IoTempower OpenWrt router setup script, newly developed at the time of writing this thesis, to configure an external router with step-by-step instructions and guidance. A wide range of routers is supported, provided they are compatible with the latest versions of OpenWrt and can be accessed via SSH.

Much like the local access point configuration process, users are provided with straightforward instructions and requirements for the router to be used. Once these

prerequisites are met, the user needs to enter the network name and password. These details are then saved into a configuration file, after which IoTempower scripts are initiated to begin the OpenWrt setup. This setup process is more involved and requires the user to access the router's web interface and upload an OpenWrt firmware file. After a reboot of the router, configuration is automatically continued and finished. The instructions to complete these steps, alongside any additional information, are printed in the log area on the bottom of the screen.

6.1.5 Connected Clients Report

After setting up an access point and connecting clients, their connections can be monitored using the "View Connected Clients" option in the application. A screenshot of this screen is shown in Figure 17.

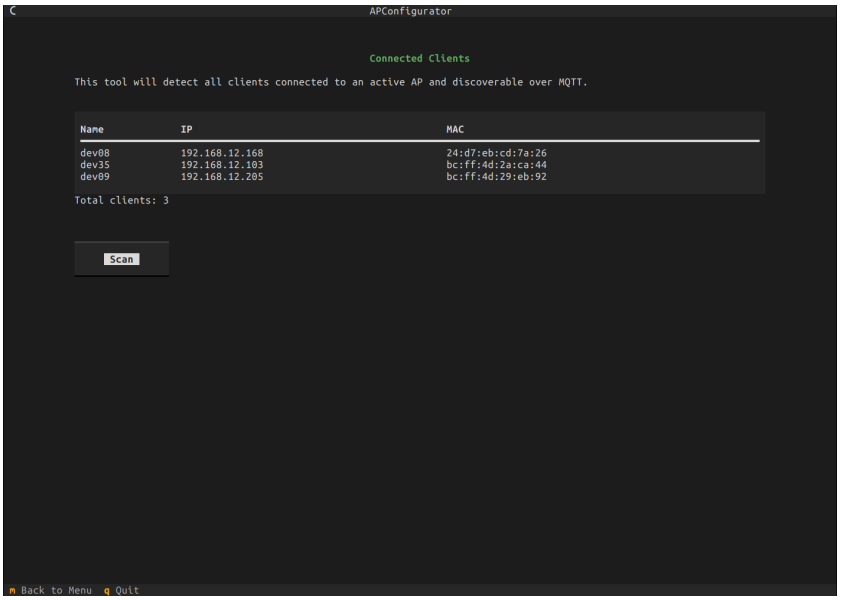

Figure 17. Connected clients screen.

This feature is powered by a script that operates in two stages: first, it detects the IP addresses of active MQTT connections, and then it uses the `arp-scan` tool to retrieve the IP and MAC addresses of connected Wi-Fi clients. The combined output is displayed to the user in a table, listing all connected clients along with their name (if set during the IoTempower flashing process), IP address, and MAC address.

6.1.6 Access Point Settings View

This screen allows the user to see the saved Wi-Fi credentials and other information about their system. A screenshot of this screen can be seen in Figure 18.

When the user selects the "AP Settings" option from the main menu, a new screen appears with the following details: the chosen network name (SSID), IP address, the default IP address used by IoTempower, the name of the output wireless device on the system, an indicator showing whether IoTempower is active, an indicator showing

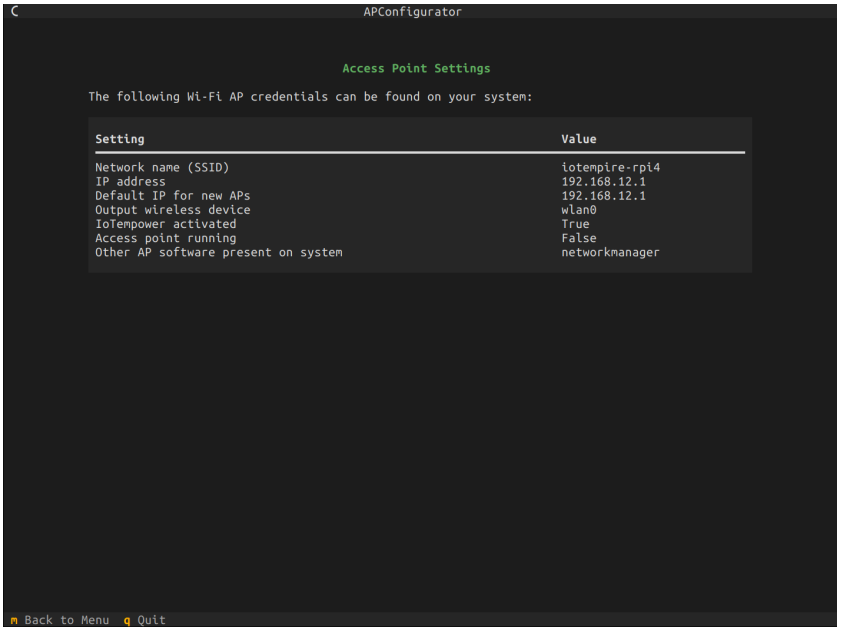

Figure 18. Access point settings screen.

whether an access point is currently running, and the name of any AP software installed on the system outside of IoTempower.

While this screen provides a more comprehensive view, the information remains straightforward, allowing the user to verify if the access point is correctly configured and operational without being overwhelmed by too much information.

6.1.7 Wi-Fi Module Information View

This screen allows the user to see details about their Wi-Fi module and more general technical system information as gathered by the application. A screenshot of this screen can be seen in Figure 19.

While the Access Point Settings view offers basic configuration details, this screen provides more technical information about the system, gathered and parsed by the Wi-Fi detection script. Here, users can see information about their Wi-Fi hardware, including the manufacturer, the specific chip or USB device name, the driver in use, CPU details, and a platform/system identifier string that includes the system architecture, kernel version, and other relevant details as provided by the "platform" Python library.

For systems with Broadcom or Intel Wi-Fi chipsets, an additional notice is displayed, alerting users to potential client limits in AP mode. As identified during the experiments, these manufacturers' chipsets often have a maximum number of clients that can connect simultaneously. The screen informs users of these limitations and offers recommendations for overcoming them.

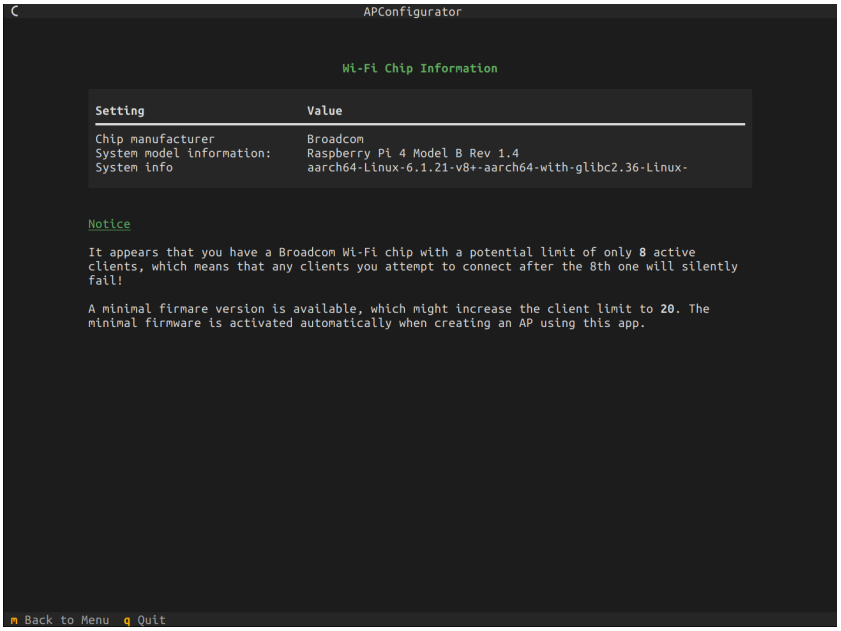

Figure 19. Wi-Fi module information screen.

6.1.8 Broadcom Minimal Firmware Activation

This module is designed for systems specifically using Broadcom Wi-Fi modules that support minimal firmware, primarily applicable to Raspberry Pi 4 and 5 models. The module first verifies the location of the minimal firmware file and then creates a system link, replacing the general firmware path with the minimal firmware. To activate the newly applied firmware, the kernel module is reloaded.

If the system detects a compatible Broadcom Wi-Fi chipset, this minimal firmware patch is automatically applied when setting up a local access point.

This module operates internally and does not have a separate user interface screen.

6.2 Codebase

The implementation is written in a combination of Python and bash. The code is available at a public repository on GitHub[23].

To run the program, Python and bash have to be installed on the system, the latest version of IoTempower has to be installed and activated, and the user needs to have sudo rights. Additionally, the following libraries need to be installed and available: `lshw`, `lsusb`, `lspci`, `dmesg`, `arp-scan`, `systemctl`, `ip` and `textual`, `paho-mqtt` for Python.

Most of the required libraries are commonly pre-installed on Linux systems.

The program can be started using the `run.sh` script in the repository. The app will first prompt for the superuser password, as many commands within the app require elevated privileges. Once the password is entered, the Python-based TUI will be launched, clearing the current terminal window.

[23] `https://github.com/tonysln/msc`

6.3 Testing

In this section, I describe the testing procedure of the completed solution, which I conducted with additional feedback from Ulrich Norbisrath.

To evaluate the program, it was installed on a Raspberry Pi 4B+ with DietPi Linux and IoTempower pre-installed. Missing libraries, such as Textual, were installed, and the previously used ESP8266 client devices were utilized once again.

The first test involved creating a local hotspot using hostapd. This was achieved by launching the program, selecting the first menu option, entering the necessary details, and clicking on "Configure". Messages appeared detailing that activating minimal firmware was necessary (but the step was skipped, as minimal firmware was already active on the system). An MQTT server was automatically started. After waiting for the newly created network to appear in the list of available networks (verified using a system tool), clients were connected to the hotspot, and their connections were accurately monitored using the Connected Clients feature of the program.

The next test involved creating a NetworkManager hotspot. For this, the program was launched, and the previous hotspot was disabled using the menu. After re-launching the program, the same steps were taken as in the hostapd local hotspot creation, but this time, NetworkManager was chosen as the back-end software. After entering network details and clicking on "Configure", messages appeared, requesting the user to restart their device for the AP to activate. After completing the restart and launching the program again, an active AP was detected (reported by the program). Further, clients were connected and could be observed through the program's Connected Clients screen.

Unfortunately, the external router configuration test could not commence due to hardware being unavailable, in addition to time constraints. However, it was verified that the configuration script is launched correctly and without issue after selecting the external router configuration option from the menu and following the presented instructions.

Besides creating an AP, the Settings and Wi-Fi chipset information screens were verified to show the correct information: recently used credentials for the access points, correctly identified wireless device name, and correct chip manufacturer and model names. The Raspberry Pi model information was correctly reported, and a notice regarding the client limit on Broadcom hardware was present.

7 Discussion

In this section, I present and discuss the results of this thesis.

First, I discuss the results of the AP device experiments. The experiments aimed to evaluate the performance and reliability of various access point devices in an IoT educational setting. Each device was tested for its ability to support multiple connected clients, ease of setup, and overall stability, among other requirements.

Further, I discuss the completed access point configuration tool. The custom AP configuration system, integrated with the IoTempower framework, aims to assist students in setting up an AP on a range of different hardware devices with different Wi-Fi chips present, providing clear instructions and hiding complexity from novice learners.

7.1 Connectivity Experiments

In Section 5.2 of this paper, I described the process of comparing 16 different hardware configurations as potential access point hosts for use in IoT education. A summary of all experiments is presented in Table 13. When testing the equipment, a specific set of steps was designed with an IoT classroom use case in mind. While most of the equipment proved adequate for this scenario, most presented at least some issues.

Among the tested devices, the following stood out as the best choice: Atomic Pi, GL-MT300N-v2, and the HLK-7628N kit. These devices met the criteria most effectively, encountered the fewest setup issues, and were the most user-friendly. Additionally, they supported a wide range of clients and were both affordable and readily available. Additionally, these devices met all of the requirements outlined in Section 4.

Of the remaining devices, most presented some issues or limitations, most commonly in the number of maximum connected clients, with 10 clients being a common limit. For some, solutions exist to overcome this limitation and increase the allowed amount of connected clients. Such is the case with the Raspberry Pi 4 and 5 boards. For others, there are no easy ways to circumvent the problems. Moreover, some of the devices failed to function in AP mode at all or were too unstable to rely on.

In general, the experimental process was limited by the availability of devices, time, and resources. A wider range of devices is definitely recommended for future experiments, including more USB Wi-Fi adapters, different laptops, and other external router options.

The defined use case and steps can be applied to a broader range of hardware for future experiments. Additional parameters, such as speed and latency, can also be measured to make the procedure more refined.

The experiments conducted here were exploratory rather than strictly structured, which leaves room for improvement through a more rigorous testing process. Some devices may benefit from deeper troubleshooting and debugging of their limitations. Conducting a much more thorough search of custom drivers or other fixes might improve the performance of some devices.

7.2 Access Point Configuration Tool

A custom software solution was developed to join together the various fragmented methods of running an AP in IoTempower. These methods can be confusing and error-prone for beginners and inexperienced students, so the goal was to package them into a cohesive interface with modules that detect hardware and handle configuration in the background automatically.

This software is a novel addition to IoTempower, utilizing a new interface library. While the solution features a moderate level of complexity - consisting of multiple components and modules - it remains relatively simple in nature, functioning mainly as a front-end to underlying scripts. This simplicity comes at an advantage, as it allows for the possibility of developing alternative front-ends with minimal effort in the future.

The completed solution meets all of the criteria from Section 4, being specifically developed with classroom usage in mind. The program performs well and meets its intended purpose, as demonstrated by the initial, albeit limited, testing. However, extensive testing across a wider range of devices, architectures, and configurations is necessary, as the diversity of hardware combinations may reveal new issues.

Table 13. Access point hardware experiment results summarized.

Category	**Device**	**Wi-Fi Chip**	**OS**	**Max. Clients**	**MQTT Broker**
Boards	Raspberry Pi 4B+	Broadcom BCM2711	Linux	8[a]	Yes
	Raspberry Pi 5	Broadcom BCM2712	Linux	7[a]	Yes
	Raspberry Pi 3B	Broadcom BCM43438	Linux	10[b]	Yes
	Raspberry Pi Zero W	Broadcom BCM2835	Linux	9[b]	Yes
	Atomic Pi	MediaTek (Ralink) RT5572	Linux	At least 23	Yes
Routers	GL.iNet GL-MT300N-v2	MediaTek MT7628AN	OpenWrt	At least 23	Yes
	HLK-7628N Kit	MediaTek MT7628AN	OpenWrt	At least 23	Yes
	Tenda TX2 Pro	Realtek RTL8197H + RTL8832BR	Unknown	At least 23	No
Laptops	HP EliteBook 840 G1	Broadcom BCM43228	Linux	None	Yes
	HP EliteBook 840 G5	Intel(R) Wireless-AC 8265	Linux	11	Yes
	HP EliteBook 840 G6	Intel(R) Wireless-AC 9560	Linux	11	Yes
Phones	Samsung Galaxy A5	Exynos 7880	Android	10[c]	Yes
Misc.	ESP32	Xtensa LX6	Custom	10	Not tested
	RPi 4B+ and USB adapter	Ralink MT7601U	Linux	None	Yes
		Realtek RTL8188CUS	Linux	7	Yes
		Realtek RTL8188SU	Linux	None	Yes

[a] Up to 20 with minimal firmware and at least 23 with downgraded firmware.
[b] Up to 20 with downgraded firmware.
[c] None if using Termux and Android 13 or newer.

One of the key features of this application is its ability to automatically detect and configure many system elements, effectively hiding the underlying complexity from the user. While this approach has had some success, there is room for improvement. Currently, the script relies on running various Linux tools to gather system information and then search for known keywords in the output. This method works for many platforms, but the variability in tool output makes parsing challenging. A more intelligent approach should be considered for future iterations.

Additionally, developing the app involved overcoming a learning curve with the Textual library, which required time to master despite the detailed documentation available. The interface, though functional, may not be as intuitive or polished as it could be, potentially containing usability issues. Consulting with a designer or UX expert could enhance the design and user experience.

8 Conclusion

In this thesis, I explored the use of access points within IoT systems by comparing potential hardware devices for the role of an access point in a series of experiments. Further, I created a custom software solution for semi-automatic AP configuration on Linux-enabled devices, specifically tailored for IoT education use.

A story-driven use case was presented, which highlights the issues students face with AP devices, from which specific requirements were derived as a basis for the experiments and final software solution.

My experiments revealed that while a variety of hardware devices seem suitable for IoT classrooms, many show significant limitations, particularly in the maximum number of clients they can support. Many of the devices did not function as access points, despite having wireless capabilities. Additionally, various configuration challenges emerged, requiring troubleshooting and networking knowledge to resolve, which novice students lack. To address these challenges, I developed a custom system to simplify the process of setting up an AP across a range of equipment, considering the differences between Wi-Fi chips. The software, while command-line-based, offers an intuitive interface that mimics a graphical user interface (GUI) experience, making it accessible even to those with limited technical expertise. The completed program is specifically built to handle different Wi-Fi chipsets in devices and make necessary adjustments automatically to ensure the most optimal performance of the device in AP mode.

Still, there remains room for improvement in the completed solution, particularly in the area of system and chip detection, which could be enhanced to cover a broader range of devices and scenarios. Current detection methods are functional but lack the robustness needed for a wider scope of devices. Future research could delve deeper into the limitations identified, exploring reverse engineering techniques and experimenting with alternative drivers for different devices and chips. This could lead to a better understanding and solutions for the challenges faced in IoT education.

Overall, this thesis highlights the complexities of selecting and configuring access point devices for IoT education, provides a use case and testing framework to assess hardware suitability, offers insights into the behavior of various Wi-Fi chips, and presents a configuration system that aims to simplify the process of setting up an access point.

References

[1] pelwell, "Opinions about pi 4b/3b+ wifi features · issue #2853 · iiab/iiab." https://github.com/iiab/iiab/issues/2853. Accessed: Oct. 29, 2023.

[2] H. F. Atlam, R. J. Walters, and G. B. Wills, "Fog computing and the internet of things: A review," *big data and cognitive computing*, vol. 2, no. 2, p. 10, 2018.

[3] D. D. Ramlowat and B. K. Pattanayak, *Exploring the Internet of Things (IoT) in Education: A Review*, vol. 863 of *Advances in Intelligent Systems and Computing*, p. 245–255. Singapore: Springer Singapore, 2019.

[4] U. Norbisrath, R. P. Machado, A. Slavin, and M. B. Zorec, "Putting the s back into iot: Tutorial on sensor and actor systems in small portable ad-hoc networks," in *Proceedings of the 18th ACM International Conference on Distributed and Event-Based Systems*, DEBS '24, (New York, NY, USA), p. 208–211, Association for Computing Machinery, 2024.

[5] dorapi, "wireless - only 8 devices can connect to my pi 4 - raspberry pi stack exchange." https://raspberrypi.stackexchange.com/questions/124231/only-8-devices-can-connect-to-my-pi-4. Accessed: Oct. 09, 2023.

[6] martignoni, "Number of wireless clients in ap mode is very limited · issue #3010 · raspberrypi/linux · github." https://github.com/raspberrypi/linux/issues/3010. Accessed: Oct. 09, 2023.

[7] tohiko, "Cannot connect more than 10 devices to raspberry pi - raspberry pi forums." https://forums.raspberrypi.com/viewtopic.php?t=303631. Accessed: Oct. 09, 2023.

[8] tohiko, "Max connected users limit wifi ap raspberry pi 3b+ - raspberry pi forums." https://forums.raspberrypi.com/viewtopic.php?t=210147. Accessed: Oct. 09, 2023.

[9] 2innocent, "number of users that can connect to raspberry 3b - raspberry pi forums." https://forums.raspberrypi.com/viewtopic.php?t=241223. Accessed: Oct. 09, 2023.

[10] M. T. Schulz, "Teaching your wireless card new tricks: Smartphone performance and security enhancements through wi-fi firmware modifications," 2018.

[11] M. Schulz, D. Wegemer, and M. Hollick, "Demo: Using nexmon, the c-based wifi firmware modification framework," in *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, (Darmstadt Germany), p. 213–215, ACM, July 2016.

[12] A. Blanco and M. Eissler, "One firmware to monitor 'em all.," 2012.

[13] J. Tourrilhes, "Wireless extensions for linux." https://hewlettpackard.github.io/wireless-tools/Linux.Wireless.Extensions.html, 1997. Accessed: 2024-06-27.

[14] O. Project, "Wireless utilities." https://openwrt.org/docs/guide-user/network/wifi/wireless-tool/wireless.utilities, 2024. Accessed: 2024-06-27.

[15] T. N. Ford, E. Gamess, and C. Ogden, "Performance evaluation of different raspberry pi models as mqtt servers and clients," *International journal of Computer Networks & Communications*, vol. 14, p. 1–18, Mar. 2022.

[16] X. Zhong and Y. Liang, "Raspberry pi: An effective vehicle in teaching the internet of things in computer science and engineering," *Electronics*, vol. 5, p. 56, Sept. 2016.

[17] S. C. Roy, N. Funabiki, M. M. Rahman, B. Wu, M. Kuribayashi, and W.-C. Kao, "A study of the active access-point configuration algorithm under channel bonding to dual ieee 802.11n and 11ac interfaces in an elastic wlan system for iot applications," *Signals*, vol. 4, p. 274–296, Apr. 2023.

[18] C.-Y. Chang, C.-H. Kuo, J.-C. Chen, and T.-C. Wang, "Design and implementation of an iot access point for smart home," *Applied Sciences*, vol. 5, p. 1882–1903, Dec. 2015.

[19] R. P. Machado, U. Norbisrath, and R. Jubeh, "Iot educational framework case study: Devices as things for hands-on collaboration," *Journal of Engineering Education Transformations*, no. 37, 2024.

[20] L. Nagy and A. Colesa, "Router-based iot security using raspberry pi," in *2019 18th RoEduNet Conference: Networking in Education and Research (RoEduNet)*, (Galati, Romania), p. 1–6, IEEE, Oct. 2019.

[21] wemos.cc, "Lolin d1 mini v3.1.0 — wemos documentation." https://www.wemos.cc/en/latest/d1/d1_mini_3.1.0.html. Accessed: Aug. 07, 2024.

[22] G. Cyrino, D. Cavalcante, M. Aureliano, A. Bucioli, I. Avelar, A. Cardoso, E. Lamounier Jr, and G. Mendes de Lima, "An interactive holographic environment for visualization of heart structure and activity," pp. 1–6, 11 2018.

[23] torvalds, "linux/drivers/net/wireless/broadcom/brcm80211/brcmsmac/scb.h at master · torvalds/linux." https://github.com/torvalds/linux/blob/master/drivers/net/wireless/broadcom/brcm80211/brcmsmac/scb.h. Accessed: Nov. 05, 2023.

[24] RPi-Distro, "firmware-nonfree/debian/config/brcm80211/cypress at bullseye · rpi-distro/firmware-nonfree." https://github.com/RPi-Distro/firmware-nonfree/tree/bullseye/debian/config/brcm80211/cypress. Accessed: Nov. 28, 2023.

[25] holta, "Raspberry pi 3 used to support 32 wifi connections but is now limited to 10. rpi 3 b+ and rpi 4 are now limited to 4 [replacing raspbian with ubuntu 19.10.1 increases this to 32; what does ubuntu 20.04 offer?] · issue #823 · iiab/iiab · github." https://github.com/iiab/iiab/issues/823. Accessed: Oct. 09, 2023.

[26] Anthony96922, "Ap driver for mt7601u dongles." https://github.com/Anthony96922/mt7601u-ap, July 2024. Accessed: Aug. 07, 2024.

[27] ameriDroid, "Atomic pi – ameridroid." https://ameridroid.com/products/atomic-pi. Accessed: Aug. 11, 2024.

[28] MPANN1, "Max clients on wifi access point with 7265 - intel community." https://community.intel.com/t5/Wireless/Max-clients-on-wifi-access-point-with-7265/m-p/410553#M11378, Feb. 2018. Accessed: Aug. 08, 2024.

[29] carras3, "Ac8265 in access point mode - intel community." https://community.intel.com/t5/Wireless/AC8265-in-Access-Point-Mode/m-p/1234590, 2020. Accessed: Aug. 08, 2024.

[30] A. Kulkarni, "Answer to "'could not add sta to kernel driver' hostapd - not able to connect more then 10 clients"." https://askubuntu.com/a/779728, May 2016. Accessed: Aug. 08, 2024.

[31] Hill, "How many wireless devices can one gl mini router handle?." https://blog.gl-inet.com/testing-how-many-wireless-devices-can-one-gl-mini-router-handle/, June 2017. Accessed: Aug. 09, 2024.

[32] vcarvalho12, "Mounting a microsd on hi-link board hlk7628n - for developers - openwrt forum." https://forum.openwrt.org/t/mounting-a-microsd-on-hi-link-board-hlk7628n/100374, June 2021. Accessed: Aug. 09, 2024.

[33] tenda.cn, "Tx2 pro _tenda-all for better networking." https://www.tendacn.com/product/specification/tx2pro.html. Accessed: Aug. 09, 2024.

[34] E. Systems, "Arduino core for the esp32," July 2024. Accessed: Aug. 02, 2024.

[35] espressif, "arduino-esp32/libraries/wifi/examples/wifiaccesspoint/wifiaccesspoint.ino at master · espressif/arduino-esp32." https://github.com/espressif/arduino-esp32/blob/master/libraries/WiFi/examples/WiFiAccessPoint/WiFiAccessPoint.ino. Accessed: Aug. 10, 2024.

[36] Govert, "Answer to "how to force enable 'soft ap' to allow hostednetwork?"." https://superuser.com/a/1590342, Oct. 2020. Accessed: Aug. 02, 2024.

Appendix

I. Glossary

AP Access Point.
A device for wireless communication.

DHCP Dynamic Host Configuration Protocol.
Protocol for automatically assigning IP addresses.

Hostapd Host Access Point Daemon.
Software that enables AP functionality.

IoT Internet of Things.
Network of interconnected smart devices.

MQTT Message Queuing Telemetry Transport.
Messaging protocol for IoT.

SBC Single-Board Computers.
Compact computers with all components on a single board.

SoC System-on-a-Chip.
An integrated circuit with multiple components.

SoftAP Software Access Point.
Strictly software-based access point.

TUI Text-based/Terminal User Interface.
An interface implemented in the terminal.

II. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Anton Slavin**,
(author's name)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Wireless Network Connectivity in Educational IoT Environments,
   (title of thesis)

   supervised by Ulrich Norbisrath.
   (supervisor's name)

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Anton Slavin
13/08/2024