Project Integrate: An SDN-Based Smart Home Automation System Using Raspberry Pi Networking

1st Liam Robertson
SCU School of Engineering
Santa Clara University
Santa Clara, California, United States
lprobertson@scu.edu

2nd Marley Willyoung
SCU School of Engineering
Santa Clara University
Santa Clara, California, United States
mwillyoung@scu.edu

3rd Luke Hofstetter
SCU School of Engineering
Santa Clara University
Santa Clara, California, United States
lhofstetter@scu.edu

Abstract—**Project Integrate offers an advanced smart home automation solution by leveraging a network of Raspberry Pi units to establish an intelligent, privacy-focused system. This system minimizes user interaction and achieves precise localization within an accuracy of less than one meter, using a Local Area Network (LAN) Access Point as the backbone for communication. This system utilizes inexpensive single-board computers (SBCs) that communicate using a custom application-layer protocol that is designed to coordinate distance measurements, monitor device activity and allow the system to respond to user proximity without transmitting personal data outside the local network. This new proposed architecture supports scalability and portability, fundamental to adapting to various home environments. Initial tests indicate that Project Integrate efficiently manages smart devices based on user location, offering significantly convenience for users. This paper details the design, implementation, and operational efficacy of Project Integrate, demonstrating its advantages over traditional smart home systems in terms of privacy, ease of use, and adaptability.**

Index Terms—**Networking, Portability, Scalability, User Proximity Detection, Broadcast**

I. Introduction

As smart home technologies increasingly become integral to modern living, there remains a substantial need for more inclusive and user-friendly automation systems. Traditional solutions often require direct user interactions or are hindered by inefficient response mechanisms to non-target movements, such as those caused by pets. In addition, data privacy has become a must for many consumers, which necessitates the design of systems that can provide the convenience modern users demand while also respecting the privacy they deserve [1]. However, there is little to no effective movement by major industry players in this area, indicating that modern systems are built on the concept that user privacy must be compromised in some way in order to deliver innovative and effective technologies [2].

Project Integrate seeks to overcome these limitations by deploying an innovative smart home automation system that dramatically reduces the need for user interaction while enhancing device responsiveness based on precise user proximity. Utilizing a network of Raspberry Pi units, the Pi's communicate with each other directly using a custom application-layer protocol, and leverage IPv6 networking to ensure seamless communication. This system architecture not only facilitates a non-intrusive user experience but also staunchly protects user data by confining it within the local network.

The objective of *Project Integrate* is to advance the functionality and accessibility of smart home technologies, making them more adaptable to the needs of individuals with disabilities or those preferring less overt interactions. It effectively distinguishes between human and non-human triggers, ensuring that device activation's are intentional and relevant to the user's presence. Particularly, this system empowers those seeking to automate their smart devices while also protecting their privacy. By using limited information that is already included by packets that are sent from user devices, Integrate is able to track and respond to changes in user locations without requiring greater permissions or compromising their personal information. In addition, Integrate utilizes already existing technologies to automate interactions, providing users with a low, up-front cost that allows them to instantly connect their already existing devices seamlessly.

This paper will explore the design, implementation, and operational testing of *Project Integrate*, illustrating its competitive edge over conventional systems in privacy, usability, and adaptability. We will also discuss our challenges, limitations, and future version plans. By emphasizing advanced user proximity detection and intelligent response mechanisms, *Project Integrate* introduces a new paradigm in smart home interaction. For a detailed list of terms and acronyms used throughout this document, refer to the Acronyms Table in Section III-A.

II. Methods

A. System Architecture

Project Integrate is built upon a robust architecture using four Raspberry Pi units, each serving distinct roles within the system. One unit acts as the ”root” node, which coordinates the overall system operations and manages communication among all other nodes. The remaining three units serve as ”leaf” nodes, which are responsible for detecting and responding to user proximity based on network activity. This configuration ensures a distributed approach to managing tasks and enhances system reliability by isolating critical functions to specific devices.

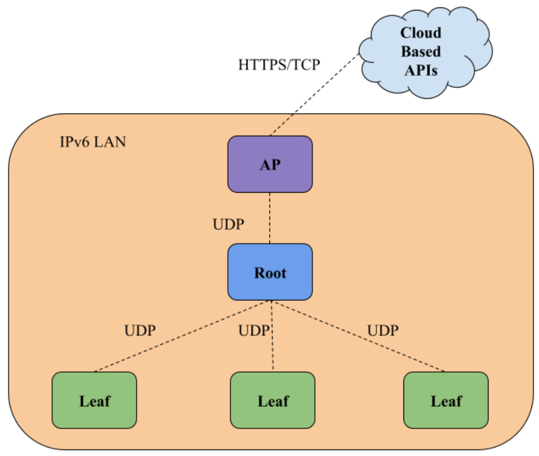

Fig. 1. Wireless Network Setup

B. Software and Network Design

Integrate is developed exclusively in C++ due to its performance, efficiency, and low overhead, three important factors when working with resource-constrained devices like the Raspberry Pi's utilized in this system. We steered away from interpreted languages like Python because they are generally slower and less performant than compiled languages like C++ [3]. The enhanced efficiency of C++ allows for more responsiveness which is a necessary consideration given the soft real-time nature of the overall system. C++ allows for control over low-level system operations, including packet capture and RSSI analysis, which are crucial for the functionality of the system. Another advantage of C++ is that it allows us to utilize existing Unix code that was developed in C, which allows for easy interaction between the software and kernel through existing, well-defined system calls.

The network setup for *Project Integrate* currently builds upon existing network infrastructure that is built into most homes - namely, network communication is facilitated using 802.11 WiFi Access Points. Here, we choose to utilize IPv6 at the network layer in order Actual communication between the nodes is formatted and defined by a custom-designed application-layer protocol (entitled **LML**), specifically tailored to optimize information transfer between the Raspberry Pi units without sending excessive amounts of data. This protocol employs IPv6 at the network layer, which allows for greater compatibility with existing network technologies and ensures that our system builds upon modern standards [4].

By emphasizing simplicity and the well-defined message structure that JSON defines, the LML protocol allows for lower probability of bit-errors, faster packet parsing and human readability. We choose to use UDP for the transport layer in order to avoid the often-unavoidable latency that comes when using TCP. However, each node dynamically samples the noise in the surrounding environment and supports falling back to TCP should network conditions necessitate.

The software architecture is modular, with clear abstractions between the network layer, application logic, and device control interfaces. Each node runs a daemon process that initializes at system boot and continuously monitors for relevant network packets. Nodes dynamically adjust their roles (either as root or leaf) based on network conditions and system needs, ensuring optimal load distribution and fault tolerance. Communication protocols and data handling routines are designed to prioritize system responsiveness and user data privacy.

III. Terminology Overview

A. Abbreviations and Acronyms

Before looking into the technical aspects of *Project Integrate*, it is vital to understand the specific terminology and structures of the project outlined below.

TABLE I
List of Acronyms

Acronym	Description
SDN	Software Defined Networking
IoT	Internet of Things
LML	Custom Protocol named after Developers
IPv6	Internet Protocol Version 6
Pi	Raspberry Pi
NIC	Network Interface Card
SoC	System on Chip
UDP	User Datagram Protocol
TCP	Transmission Control Protocol
HTTPS	Hypertext Transfer Protocol Secure
PNP	Plug and Play
BLE	Bluetooth Low Energy
POSIX	Portable Operating System Interface

B. Units

For *Project Integrate*, precision in measurements is essential for accurate system behavior and performance evaluation. The following units are primarily used:

- Time: milliseconds (ms) for latency measurements and microseconds (μs) for timing the responsiveness of sensors and actuators.
- Signal strength: decibels relative to a milliwatt (dBm) for measuring RSSI (Received Signal Strength Indicator).
- Data rate: Megabits per second (Mbps) for network throughput.
- Distance: meters (m) for the spatial calculations between nodes.

Consistent use of SI units ensures that the measurements are universally interpretable, and units are clearly stated whenever mixed measurements are unavoidable.

C. Equations

Several fundamental equations are utilized in *Project Integrate* to calculate device proximity, system efficiency, and network performance:

$$\text{Latency} = \frac{\text{Total Packet Travel Time}}{\text{Number of Hops}} + \text{Processing Time} \quad (1)$$

Latency is computed by dividing the total time taken for data packets to travel from the source to the destination by the number of hops involved and adding the processing time at each hop. In *Project Integrate*, the hops include:

- Communication between the root node and the leaf nodes,
- The hop from the root node to initiate an external API call,
- The subsequent return of the API response through the subnet's access point to control a device, such as a light.

The number of hops and the processing times can vary depending on the network configuration and the route taken by the packets, especially for external API calls. Our system employs IPv6 with UDP to ensure efficient and rapid data transmission within the subnet, which is critical for maintaining low latency [5]. Outside our own subnet, where direct latency calculation becomes more challenging due to additional hops and external network factors, we reference existing methodologies and enhancements in network technologies [6].

$$\text{Latency}_{\text{root-leaf}} = \text{Propagation Delay} + \text{Processing Delay} \quad (2)$$

This equation quantifies the internal communication latency, which is important for optimizing inter network interactions and includes both the signal propagation time and delays due to data handling at each node.

$$\text{Latency}_{\text{API call}} = \text{Latency}_{\text{root-AP}} + \text{Latency}_{\text{Internet}} + \text{Latency}_{\text{server}} \quad (3)$$

To model the latency involved in making external API calls, essential for understanding the impact of external communications on system performance. It accounts for the local transmission to the access point, transit over the internet, and processing at the server.

$$\text{Expected Attempts} = \frac{1}{p} \quad (4)$$

To estimate the average number of transmission attempts required, assuming a packet success rate of p, to incorporate the probability of packet loss into network performance analysis of the system.

$$\text{Adjusted Latency} = \text{Latency} \times \text{Expected Attempts} \quad (5)$$

This model helps in predicting more realistic network performance, especially under varying network conditions.

$$\text{RSSI}_{\text{avg}} = \frac{\sum \text{RSSI values over sample period}}{\text{Number of Samples}} \quad (6)$$

We calculate the average RSSI to determine the proximity of the user. The RSSI values are collected over a specified sample period.

$$\text{Distance} = 10^{\frac{(\text{RSSI}_{1m} - \text{RSSI})}{10 \times n}} \quad (7)$$

Where RSSI_{1m} is the RSSI value measured at one meter from the transmitter, serving as a calibration reference, and n is the environmental attenuation factor. The value of n, typically ranging between 2 and 4, varies depending on environmental conditions and must be calibrated for accuracy. We are following established practices in RSSI-based location estimation, which rely on averaged RSSI measurements [7].

$$\text{Path Loss (dB)} = \text{PL}_0 + 10 \cdot n \cdot \log_{10}(\frac{d}{d_0}) + X_\sigma \quad (8)$$

where:

- PL_0 is the initial path loss observed at a reference distance d_0,
- n is the path loss exponent that indicates how the path loss increases with an increase in distance,
- d represents the distance to the transmitter,
- X_σ is a Gaussian random variable (in dB) accounting for shadow fading, which introduces variation in the path loss due to obstacles obstructing the signal path.

This model is integral for understanding how signal strength varies with distance and environmental conditions in wireless communications [8].

IV. Implementation

A. Hardware

Project Integrate is initially configured with a Raspberry Pi 3 Model B+ serving as the root node and three Raspberry Pi Zero units as leaf nodes. Each Raspberry Pi is equipped with its native (NIC) for standard operations and an additional external antennae configured in monitor mode for data collection. This dual setup is managed by the Channel Sync Thread which ensures all NICs operate on the same channel to maintain functionality. The system currently incorporates Govee Smart Home Lights, future versions will allow adding different brands of devices as long as they support an API.

TABLE II
Hardware Requirements

Sl.No	Hardware Requirement	Specification
1	Raspberry Pi 3 Model B+ (Root Node)	1 unit
2	Raspberry Pi Zero (Leaf Nodes)	3 units
3	Native NICs	4 units (1 per Pi)
4	AWUS036ACS Antennas	4 units (1 per Pi)
5	Govee Smart Home Lights (Model H6008)	3 units

B. Software Implementation

The system software, primarily developed in C++, optimizes performance on the Raspberry Pi Zero hardware by having full control over and managing low-level operations. The nodes run on a lightweight Linux distribution for operation of daemons and network tools to further increase performance and efficiency.

TABLE III
SOFTWARE COMPONENTS AND VERSIONS

Component	Description	Version
Operating System	Linux	Raspbian Lite
Development Language	Application Code	C++
Network Tools	iw, awk	Standard
Daemon	System Daemon	Custom

C. System Integration

Upon booting, each node automatically starts the system software from a pre-configured image, to determine and assign network roles. This custom protocol, manages robust and secure interactions over IPv6. After establishing roles, the nodes proceed to a calibration phase where they synchronize to create an accurate network topology and establish baseline RSSI values for distance measurement. This allows for the dynamic adjustment of Govee lights in response to user movement, maintaining all data processing and storage within the local network to prioritize privacy and security.

V. SYSTEM OVERVIEW

A. System Initialization

At the onset of operation, each node within the system—comprising multiple Raspberry Pis—undergoes an initialization phase. This phase sets up the nodes for their respective roles within the network, whether as a root or a leaf node. The process begins immediately upon system boot, triggered by a predefined job that ensures the automation software executes as soon as the operating system is loaded. First, a dedicated log file is created to facilitate debugging by recording operational events and anomalies. Concurrently, a UDP socket is established to manage data transmissions over the network. This socket operates on a predefined pairing port, which is essential for the subsequent node pairing phase.

Each node is programmed to adhere to IPv6 protocols by default, enhancing the system's compatibility with modern network standards. The initialization script dynamically adjusts the scheduling policy of the thread handling network communications to SCHED_RR (Round Robin).

During this phase, nodes broadcast pairing packets using the default link-local address for IPv6. This communication is designed to establish a hierarchical node structure by determining the role of each node based on network responses. A node will assume the role of a root if no responses are received within a specified timeout, signifying its responsibility to coordinate the network. Receiving a valid response with specific fields will designate the node as a leaf, responsible for data collection and direct interaction with the root.

B. Root Node Thread

The Root Node Thread serves as the network's coordination hub, initiating a UDP socket on an IPv6 pairing port to manage Leaf Node registration and synchronization.

1) Initialization and Pairing: Upon activation, the Root Node configures a listening socket to receive calibration packets containing signal noise levels and packet counts from Leaf Nodes. This information is used to optimize network traffic flow and prevent congestion by adjusting the packet emission intervals of Leaf Nodes.

2) Calibration and Dynamic Response: Following initial data receipt, the Root Node enters a continuous monitoring phase. It processes incoming RSSI values to calculate precise node distances and makes API calls based on these measurements. These calls facilitate dynamic responses within the network's infrastructure or external systems, essential for location-based services.

Listing 1. Pseudo-code for Root Node Operations

```
BEGIN
    Establish and bind the socket for network
        ↪ pairing
    WHILE not all leaves are paired
        IF packet received THEN
            Parse and extract data
            IF new Leaf Node THEN
                Register and send calibration
                    ↪ settings
            END IF
        END IF
    END WHILE
    Transition to monitoring
    WHILE true
        Receive and process packet data
        IF distance criteria met THEN
            Execute API calls for system
                ↪ actions
        END IF
        Log updates and manage system health
    END WHILE
END
```

C. Leaf Node Thread

Leaf Nodes are data collectors and communicators within the network, following a successful pairing and calibration process with the Root Node. These nodes are responsible for capturing network packets and extracting RSSI values using pcap libraries, which are for finding the physical proximity of network devices. Upon completion of the pairing process, each Leaf Node enters its operational phase, during which it periodically emits calibration packets. These packets are intercepted by sibling nodes, enabling each Leaf to autonomously calculate distances to others and report these metrics back to the Root Node for analysis.

1) Device Monitoring and Categorization: In addition to processing network traffic, Leaf Nodes actively monitor for new devices. They manage lists that categorize devices into blocked, candidate, or permanent categories based on observed mobility patterns and signal consistency.

Listing 2. Pseudo-code for Leaf Node Operations

```
BEGIN
    Establish communication with Root Node for
        ↪ pairing
    WHILE pairing not confirmed
        Send pairing requests and process
            ↪ responses
        IF pairing confirmed THEN
            Enter calibration mode
        END IF
    END WHILE
    WHILE true
        Monitor network for packets
        Extract RSSI and calculate proximity
        IF new device detected THEN
            Categorize device based on
                ↪ criteria
            IF device qualifies as candidate
                ↪ THEN
                Forward details to Root Node
            END IF
        END IF
        Adjust operational parameters based on
            ↪ Root instructions
    END WHILE
END
```

D. RSSI Thread: Signal Strength Monitoring and Logging

The RSSI thread is integral to the system's ability to monitor wireless traffic for device tracking and spatial analysis. This thread utilizes the pcap library to capture wireless packets, allowing for the extraction of RSSI values critical for distance estimation [9]. Key functions from the pcap library include:

- *pcap_create* — Initializes a packet capture handle for examining packets on a specified network interface. It must be activated with *pcap_activate* before capturing packets [9].
- *pcap_setfilter* — Applies a compiled filter to the capture handle, allowing only the specified traffic to be captured. This function is for focusing on relevant data and enhancing performance.
- *pcap_loop* — Captures packets indefinitely until a specified count is reached, a stop condition is met, or an error occurs.

The Radiotap header of each packet is dissected to retrieve the RSSI value, which is used for analyzing signal strength and estimating the distance of devices.

Listing 3. Pseudo-code for the RSSI Thread

```
BEGIN
    Initialize pcap on network interface
    Configure interface to monitor mode using
        ↪ pcap_set_rfmon
    Compile and set pcap filter for target
        ↪ traffic
    Open log file for RSSI recording
    WHILE true
        Capture packet using pcap_loop
        IF packet is captured THEN
            Parse packet to extract RSSI from
                ↪ Radiotap header
            Extract MAC address and optional
                ↪ OUI
            Calculate distance based on RSSI
                ↪ value
            Log RSSI, MAC, OUI, and distance
                ↪ to file
        END IF
        Check log file size
        IF log file size exceeds threshold
            ↪ THEN
            Rotate log file (Function Call)
        END IF
        Handle errors and log exceptions
    END WHILE
    Close pcap handle and clean up resources
END
```

The system periodically archives the RSSI log file to prevent overflow and ensures data is maintained over extended operations. This involves a periodic review of the log file size and the execution of a log rotation mechanism to manage space efficiently. Specifically, the *checkAndRotateLog* function:

- File Monitoring: The function first checks the size of the current RSSI log file.
- Log Rotation: If the file size exceeds a predefined threshold, the function proceeds to close the existing file and delete it. This prevents the system from using excessive disk space and ensures that only the most recent data is stored.
- File Recreation: After deletion, a new log file is immediately created to continue logging without interruption. This ensures that no data is lost when the old file is closed and the new one begins.

Listing 4. Pseudocode for checkAndRotateLog

```
Function checkAndRotateLog
    Lock mutex to ensure thread safety
    Define filename as "rssi.txt"
    Output "Attempting to delete and recreate
        ↪ filename"

    Try to delete the file with name filename
    Create a new log file with name filename
        ↪ for appending

    If file creation is successful
        Output Success
    Else
        Output Fail

    Unlock mutex
End Function
```

E. Channel Sync Thread: Synchronization of Network Interfaces

The Channel Sync Thread is to ensure all network interfaces, especially those equipped with external antennas for monitoring, are synchronized to the same channel as the default network interface. This synchronization addresses the challenge posed by access points (APs) that dynamically and unpredictably change channel assignments, potentially leading to data capture discrepancies [10].

1) Purpose and Functionality: The thread actively adjusts the channel of any monitoring network interfaces (NICs) to align with the default NIC, which is critical in environments with variable AP channel configurations. Utilizing shell commands, the system identifies non-default interfaces and adjusts their settings to match the operational channel of the primary NIC. Commands involving 'iw', 'grep', and 'awk' are used for interface management and data parsing, ensuring the monitoring interfaces operate on the correct channel.

Listing 5. Pseudo-code for the Channel Synchronization Thread

```
BEGIN
    Print "Channel synchronization thread
        ↪ started."
    Define command to find the interface not
        ↪ set as default wireless
    Execute command to find non-default
        ↪ interface:
        "iw dev | grep Interface | awk '{print
            ↪  $2}' | grep -v '^"
            ↪ DEFAULT_WIRELESS "$' | head -n
            ↪ 1"
    Store result in antennaInterface
    IF antennaInterface is empty THEN
        Use fallback interface
        Print "Fallback interface used: [
            ↪ Interface Name]"
    END IF

    Configure the identified interface in
        ↪ monitor mode:
        "ip link set [Interface] down && iw
            ↪ dev [Interface] set type monitor
            ↪  && ip link set [Interface] up"
    Initialize variable for current channel
    WHILE true
        Get current channel of default
            ↪ wireless interface:
            "iw dev " DEFAULT_WIRELESS " info
                ↪ | grep channel | awk '{print
                ↪  $2}'"
        Compare and update if different:
            IF new_channel != current_channel
                ↪ THEN
                Update channel of monitoring
                    ↪ interface:
                    "iw dev [Interface] set
                        ↪ channel [new_channel
                        ↪ ]"
                Print "Channel updated to [
                    ↪ new_channel] on [
                    ↪ Interface Name]"
            END IF
        Sleep for 60 seconds
    END WHILE
    Close resources
END
```

F. Node Communication

Communication between the nodes is facilitated using sockets that send small chunks of data using the well-defined, simple LML protocol. Packets are formatted using JSON in order to utilize an already-established data exchange format and allow for simple parsing by recipient nodes.

The LML protocol specifies a few different fields, each with their own purpose. There is only a few options for each field name, primarily for the purpose of reducing the latency that may occur when parsing complicated JSON messages. These field options and their possible values are specified below:

```
    "type": "pairing" | "calibration"
    | "candidate" | "trial" | "distance" | "
        ↪ device_location",
    "noise": x dBm
    "interval": x ms,
    "leaf_name": 4 bit unique id,
    "number_of_calibration_packets": 0-255 (1
        ↪ byte),
    "candidate_addition": 1 byte unique id,
    "candidate_removal": 1 bit (0 for removal,
    1 for block),
    "unique_name_of_device": 1 byte,
    "MAC": 48-bit MAC address,
    "distance": 2 byte distance (in meters)
```

Each field after "type" and "noise" is only present when the corresponding field that necessitates the additional information is present. In other words, fields that are unnecessary are not included in the packet. The reason for the small field size is to prepare for easier future transition to more efficient and better suited network protocols, such as 802.15.4 or BLE.

G. API Call and Actions

API calls are handled by the root node using HTTPS/TCP, and only occur when the location of tracked user devices is near the location of smart devices within the home. The location of these smart devices is determined ahead of time during an initial location phase that occurs after nodes have been successfully paired and calibrated to function on the same network. In order to achieve this, the root first informs the leaves that the root will be placing status calls to the smart devices, so that the leaves can immediately listen for the device's responses on the network. Then, the root issues "status" calls to the smart devices that are present on the network. The leaves, having been informed by the root prior to the issue of the API calls, is able to listen for the responses of the smart devices on the network and locate the devices based on the signal strength of these responses.

H. Continuous Operation

All threads run continuously and will keep listening to their sockets and wait for more data to signal more commands and API calls. The nodes are informed at the pairing phase of the interval that they are to use to communicate with the root. This allows the leaves to choose when to listen to network activity and when to send updates to the root. While Linux doesn't fully support real-time requirements such as these, it does allow for limited support by letting us choose the scheduling algorithm that the threads follow. In addition, by specifying these algorithms ahead of time, we choose when to yield the processor to the other threads. Specifically, we choose to use the POSIX implementation of threads in order to allow for greater portability of the software to other devices.

VI. Results

A. Functionality Testing

We conducted preliminary tests to validate the system's effectiveness in real-world scenarios. The root node consistently established itself within 10 seconds, and leaf nodes paired rapidly, with an average pairing time in the order of microseconds. Future comprehensive testing will evaluate the system under various operational conditions to ensure robust performance.

B. Performance Metrics

Initial performance metrics concentrated on response times, energy consumption, and the precision of device localization. Notably, the system demonstrated swift response times surpassing anticipated benchmarks. The software architecture is notably lightweight, consuming minimal system memory and primarily using statically linked functions to enhance execution speed and reduce runtime dependencies. Future testing phases will develop comprehensive benchmarks to thoroughly quantify these performance aspects, ensuring the system's efficiency and reliability in diverse operational environments.

C. Challenges and Solutions

Developing a custom smart home architecture presented significant challenges, particularly in protocol development and node differentiation. To address these, we implemented a dynamic role assignment mechanism based on network conditions and node capabilities. We need to ensure efficient socket management and stable network communications, even with the complexities of maintaining simultaneous connections across multiple devices.

VII. Discussion

A. System Benefits

The main benefit of the system is the automation that it provides. The system is designed to be plug-in-play so anyone, regardless of technology knowledge, will be able to setup this system. The system also enhances user mobility as it requires little interaction from the users due to the localization performed by the system. User's personal data is also secured because the system is completely localized as no data is sent out to the cloud or any network outside the LAN.

B. Limitations and Shortcomings

One of the limitations of the system is the lack of manual configuration. Although the main goal of the system was to achieve full automation by designing a Plug-And-Play system, we still believe that the option for manual configuration is beneficial as some users may prefer this route over a fully automated system. We will explore this in the second version of our project.

VIII. Future Versions- Enhancements and Extensions

A. Control Application

One major enhancement to the overall system would be the addition of an app that can be used to manually configure a significant portion of the system. Internally the app will see all devices connected to the network and it will display these devices to the user. The user will then be able to manually add these devices to their proper lists accordingly (whether the device should be tracked or not). If a new device joins the network, the app will then send a notification to the user that a new device has joined the network and the user will then be able to take an action on this device if they please (add it to the tracked devices list). By default, all devices will not be tracked, so if the user takes no action of new devices connected to the network, then this device will not be tracked by the system. Since the system will be integrated in an app, we can add manual control of all smart devices natively within the app. Not only will manual controls within the app will be available, but the possible use of Siri, Google, and other AI conversational chat-bots could be used to execute functions within the app.

B. Application Iteration

The addition of an app allows us to expand the scope of our implementation in the initial version. The manual controls built into the app allows us to have full control over other smart devices such as smart plugs and other smart devices not considered in the previous version. While the previous version controlled smart devices solely based on a users distance to the device, the app can also allow a user to turn on devices based on the current time. While location tracking of user mobile devices is still the basis of the system, the user can configure whether each device will work according to location like in the previous version or work in a manual configuration whether that be solely time based set by the user or manual activation of the device. Although the ladder addition is not hands off (PNP), we want to provide the user with many ways to have control over their smart devices and tailor them to their specific needs.

C. Mesh Networking

A second enhancement that we will consider making is changing the way we implement communication between the nodes. The current version utilizes WiFi to mediate communication between nodes, but we may consider utilizing Bluetooth Low-energy (BLE) or creating a Thread network.

BLE may be considered due to its power efficiency and network utilization. Given that with the current system design not a significant amount of data needs to be sent between nodes, BLE can be used over its BR/EDR counterpart. With BLE, advertisement channels can be used so each node can broadcast its message to every other node in the system.

Thread is a technology built off of IEEE 802.15.4, a standard that allows for low-energy utilization by simplify the physical and MAC layers and reducing the packet size transmitted between devices.

D. Scalability

The initial implementation of our smart home system exhibits scalability limitations that are not conducive to a dynamic (PNP) environment:

- Hard-coded Elements: The maximum number of leaf nodes, along with their device IDs, are hardcoded, restricting the system's flexibility to adapt to changes without manual reconfiguration.
- Fixed Network Topology: The system is currently limited to a fixed topology with one root and three leaf nodes, which constrains scalability to larger and more complex setups.

1) Enhancements Leveraging the Control Application: Building on the foundation laid by the control application discussed in the previous section:

- Dynamic Device Integration: The app, which allows for manual configuration and monitoring of devices, can be enhanced to support dynamic discovery and integration of new devices, achieved through service discovery protocols that automate the detection and setup of new devices as they join the network.
- Enhanced User Interface for Device Management: By expanding the app's capabilities to include more intuitive controls for device management, users can easily add or reconfigure devices.
- Automated Configuration Profiles: Integrating automated configuration profiles within the app can facilitate scaling by allowing users to apply predefined settings to new devices based on their type or location.

2) Flexible Network Topology and Device Support:

- Mesh Networking Capabilities: Implementing a mesh network framework would allow the system to support a variable number of nodes dynamically. This topology adapts more efficiently to changes and increases in the number of connected devices without requiring a centralized control node.
- Expanded Device Ecosystem: Support beyond lighting to include other smart devices such as locks, cameras, and thermostats would transform the smart home system into a comprehensive solution. The control app could serve as a central point for managing these devices.

E. Technological Advancements

One technological innovation that can be used to optimize the overall system is a custom designed System on a Chip (SoC). We can design a special SoC that meets all requirements of our system by adding necessary components and removing unnecessary ones. One major change we would add to our SoC is the addition of a second Network Interface Cards (NIC's). This addition will simplify the system design and need for an external WiFi antenna. Apart from increased simplicity, We would expect a small decrease in system latency because the NIC is directly connected to the board as opposed to going through a USB connection.

We would also consider adding a specialized chip that can handle some form of encoding to convert a 24-bit OUI address into a 1 byte unique name. A 1-byte name will allow for 2^8 or 256 unique names to identify devices. Although this will suffice for small applications such as a home or small business, larger implementations of this technology will quickly run out of unique device names. For this reason, we must either make the name of each device longer than 1-byte to allow for more devices or consider a new naming scheme that can be used internally.

IX. Conclusion

A. Key Outcomes

The primary goal and achievement of this project was the development of a smart home architecture that operates independently from conventional methods like motion detectors and voice commands. This approach eliminates the need for such traditional inputs, creating a non intrusive and privacy focused living environment. By minimizing user input, our system enhances convenience through automatic adjustments to lighting and device activations, ensuring these interactions are both timely and relevant to the context, thereby enriching daily technological engagements within the home.

B. Impact on Future Research

This project serves as a beacon for further exploration into autonomous and scalable smart environments. Future research could evolve to incorporate sophisticated algorithms capable of predictive behaviors and context-aware automation, building on our foundation to innovate within IoT connectivity. Exploring technologies such as Thread or BLE could further enhance interoperability and energy efficiency. This approach positions Project Integrate as a potential alternative or complementary framework to the emerging Matter protocol, potentially steering future smart home standards.

References

[1] S. Zheng, N. Apthorpe, M. Chetty, and N. Feamster, "User perceptions of smart home iot privacy," *Proc. ACM Hum.-Comput. Interact.*, vol. 2, no. CSCW, nov 2018. [Online]. Available: https://doi.org/10.1145/3274469

[2] Y. Yao, J. R. Basdeo, O. R. Mcdonough, and Y. Wang, "Privacy perceptions and designs of bystanders in smart homes," *Proc. ACM Hum.-Comput. Interact.*, vol. 3, no. CSCW, nov 2019. [Online]. Available: https://doi.org/10.1145/3359161

[3] L. Prechelt, "An empirical comparison of c, c++, java, perl, python, rexx and tcl," *IEEE Computer*, vol. 33, no. 10, pp. 23–29, 2000.

[4] A. Hovav, R. Patnayakuni, and D. Schuff, "A model of internet standards adoption: the case of ipv6," *Information Systems Journal*, vol. 14, no. 3, pp. 265–294, 2004.

[5] S. K. Shukla and M. K. Farrens, "Leveraging network delay variability to improve qoe of latency critical services," in *2021 IEEE International Conference on Networking, Architecture and Storage (NAS)*, 2021, pp. 1–8.

[6] Y. Yang and L. Hanzo, "Permutation-based tcp and udp transmissions to improve goodput and latency in the internet of things," *IEEE Internet of Things Journal*, vol. 8, no. 18, pp. 14 276–14 286, 2021.

[7] R. Al Alawi, "Rssi based location estimation in wireless sensors networks," in *2011 17th IEEE International Conference on Networks*. IEEE, 2011, pp. 118–122.

[8] B. Blaszczyszyn, M. K. Karray, and F.-X. Klepper, "Impact of the geometry, path-loss exponent and random shadowing on the mean interference factor in wireless cellular networks," in *WMNC2010*, 2010, pp. 1–6.

[9] tcpdump.org, "pcap - Packet Capture library," https://www.tcpdump.org/manpages/pcap.3pcap.html, Accessed: Year.

[10] Y.-J. Choi and S. Bahk, "Multichannel wireless scheduling under limited terminal capability," *IEEE Transactions on Wireless Communications*, vol. 7, no. 2, pp. 611–617, 2008.