



IOT Based Secure Microgrid Monitoring System with IDS

A Project Report

Submitted by

Adithya Shashidhar – 20221CCS0055

Under the guidance of,
Dr. Ruhin Kouser

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING,
CYBERSERCURITY

PRESIDENCY UNIVERSITY
BENGALURU

DECEMBER 2025

Chapter 1

INTRODUCTION

1.1 Background

The rise of IoT is reshaping how today's power grids track and control scattered energy setups. Instead of relying solely on big central plants, local mini grids now play a key role in using clean energy, powering remote areas, or running independent electricity zones. These small-scale networks typically link solar panels with battery banks and intelligent detectors – this combo helps keep supply steady, reliable, and cleaner. Since more of them use internet-connected gadgets for oversight, hackers pose a growing threat; bad readings from hacked sensors might mess up realworld grid functions.

IoT gadgets such as the ESP8266 pop up everywhere — cheap, wireless, easy to plug in. Still, research from IEEE along with NIST reveals a gap: plenty of industrial IoT links run without encryption or solid login checks. That gap lets hackers tweak live data, feed fake numbers, even trigger service shutdowns, possibly crashing power setups. Because of this mess, tight comms security paired with instant attack spotting is key to keeping microgrids stable and safe.

1.2 Statistics

Cyberattacks on power systems are growing fast these days. Data from India's emergency tech team shows a 53% rise in digital threats to key operations gear from 2021 to 2023. Around the world, attacks tied to internet-connected devices went up – from 32 million in 2018 to more than 112 million by 2022, according to Statista's 2023 report. On top of that, SEPA found nearly six out of ten microgrids in Asia don't follow set security rules.

In India, small power grids have spread fast – particularly in villages getting electricity for the first time. By 2023, the Power Ministry reported over 350 local microgrids up and running, most using cheap internet-connected gadgets to keep track of performance. But here's the catch: more than seven out of ten don't use full encryption when sending data, making them vulnerable to hacking or sneaky changes. Because of this, experts stress the need for better protection, so energy systems stay trustworthy and accurate.

1.3 Prior Existing Technologies

Some IoT-powered microgrid monitors exist, yet they come up with serious drawbacks.

1. Traditional SCADA Systems

Supervisory Control and Data Acquisition (SCADA) setups handle big-scale monitoring – yet they cost a lot, involve heavy setup, so don't fit budget microgrids well (IEEE, 2017).

2. Basic IoT Monitoring Solutions

Devices like NodeMCU or Raspberry Pi grab info from sensors, then push it online yet a lot of skip encryption when using MQTT.

Limitations:

- No encryption
- No intrusion detection
- Static dashboards
- Weak authentication

3. MQTT-Based Monitoring Systems

Research into MQTT messaging shows it doesn't come with native protection – so risks pop up when TLS isn't used.

Limitations:

- Replay attack risk
- Spoofing
- Packet tampering
- No built-in warnings
- Man-in-the-Middle (MitM) attacks
- Credential exposure
- Message replay or topic hijacking at broker level

Prior existing technologies as compared in the Table 1.1 are mentioned above

Table 1.1: Comparison of Prior Existing Technologies

Sl. No.	Technology / Approach	Strengths	Limitations (Gap Identified)	Relevance to Proposed System
1	Traditional SCADA Systems	Highly reliable, robust monitoring for industrial grids	Very expensive, not suitable for low-cost microgrids; complex deployment	Proposed system adopts lightweight IoT architecture instead of SCADA
2	Basic IoT Monitoring (NodeMCU/Raspberry Pi)	Low-cost sensing, simple dashboards	Often use unencrypted MQTT; no IDS; weak authentication	Proposed system adds TLS, authentication & real-time IDS
3	MQTT-Based Monitoring (Unsecured)	Lightweight communication, fast publish–subscribe model	Vulnerable to replay, spoofing, tampering; no built-in security	Proposed system uses MQTT over TLS (port 8883) via HiveMQ Cloud
4	IoT IDS Frameworks	Lightweight anomaly detection	Not designed for microgrids; lack real-time visualization	Proposed system integrates IDS + dashboard + secure communication

4. IoT IDS Frameworks

Presidency School of Computer Science Engineering, Presidency University

Lightweight IDS tools exist, but most don't integrate with microgrid monitoring.

Limitations:

- Not built for live sensor display
- No secure MQTT pipeline
- High false alarms

1.4 Proposed Approach

A new IoT-powered setup fixes weak spots in current microgrid tracking systems - offering strong security without heavy tech demands. It's built on four connected parts: gathering sensor info, safe data transfer, spotting threats through analysis, plus dashboards for oversight and commands. Each piece works together to guard power grid details from start to finish. At the same time, it runs well on energy-saving devices common in local energy networks.

The sensing setup uses IoT devices built on ESP8266 chips, fitted with DHT11 and LDR sensors to track surroundings affecting microgrid performance. Instead of just one data source, DHT11 delivers temp and moisture readings useful for watching heat changes in the grid. Meanwhile, the LDR picks up shifts in natural light levels - key for small-scale solar output nearby. Combining these inputs adds backup layers plus helps spot odd environmental trends sooner, fitting well with earlier IoT research findings [2], [10].

The secure link at the edge keeps sensor info private and steady using MQTT with TLS on port 8883 - backed by studies highlighting encryption for pub-sub setups in industrial IoT [3], [11]. Instead of managing it yourself, HiveMQ Cloud runs against the broker, using cert-authenticated TLS to block spying, fake identities, or altered messages. Before sending data, the ESP8266 handles basic cleanup, saving bandwidth without heavy processing, which fits typical limits seen in small IoT gadgets [5], [8]. This protected flow tackles known weak spots found in smart grid control networks [7].

The analytics plus intrusion detection setup runs on a Flask backend that works with both rule-driven and behavior-tracking methods to spot threats. Instead of just one method, it uses a mix to catch common IoT attack patterns – such as replay, injection, spoofing, and burst-publish flooding – based on known cyber-physical risks from recent studies [4], [6]. Rather than ignoring

inputs, the Flask app listens to MQTT channels, checks if messages are legit, scans data flow for weird timing or payloads, then stores each event in an organized SQLite file. Not heavy or complex, this combined system fits guidelines for efficient threat spotting in low-power IoT environments where resources matter [5], [6].

The display and connection part shows a live feed made with Flask-SocketIO that updates instantly. It tracks heat, moisture, and brightness changes while also showing up-to-the-minute threat warnings, types of breaches, and device records. Instead of copying standard layouts, it uses ideas from protected IoT setups focused on clear visuals, fast feedback, and active monitoring [12], [13]. Access depends on user roles - only approved people get in - which lines up with solid security rules for energy grids and smart devices [9].

The new setup brings fresh ideas that stand out from typical microgrid tracking tools. For starters, it uses MQTT messages protected by TLS encryption, tackling the common problem of unprotected IoT setups seen across many smart grid reports [2], [3], [8]. Instead of skipping security checks, this version includes a built-in Advanced IDS Engine - an upgrade missing in earlier models found in several reviews where basic IoT monitors couldn't spot intrusions on their own [5], [10]. Besides better protection, it's made for budget-friendly rollout, so smaller grids can get high-level cyber safeguards even without pricey gear.

What's more, thanks to its flexible layout, support for open standards, and data-centered backbone, users could later add things like additional environmental sensors, machine learning alerts for odd behavior, or clearer dashboards - fitting well with future-focused energy upgrades and global green targets underlined in UN SDG plans [15].

Access depends on user roles - only approved people get in - which lines up with solid security rules for energy grids and smart devices [9]. Besides better protection, it's made for budget-friendly rollout, so smaller grids can get high-level cyber safeguards even without pricey gear. Access depends on user roles - only approved people get in - which lines up with solid security rules for energy grids and smart devices [9].

System overview as shown in Figure 1.1

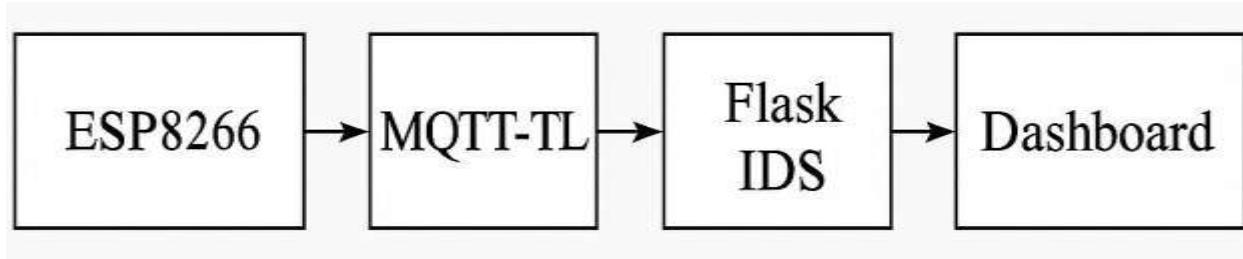


Figure 1.1 – System Overview Diagram

1.5 Objectives

The IoT-based secure microgrid monitoring setup grows from goals that are specific, testable, yet practical. Because these aims cover how it works, safety measures, day-to-day use, plus speed needs - so the system runs without hiccups.

Objective 1: Develop Real-Time Microgrid Monitoring Capability

Build a spread-out sensor setup that keeps an eye on key microgrid stats nonstop - like temp from 0 to 80°C within half a degree, humidity between 10–90% plus or minus two percent, with one percent precision, also at one percent error margin, alongside surrounding light measured via LDR and DHT11 or similar chips. Keep readings coming every couple of seconds but no slower than five; lose fewer than one in a hundred packets when the network's running smoothly. Use ESP8266 or NodeMCU units right at the source to clean up signals, even out spikes, check limits - all done locally under 120 milliseconds. Show live updates of air and lighting conditions around.

Objective 2: Implement Advanced Intrusion Detection for Microgrid Security

Build a smart system that spots break-in attempts across realistic IoT-level attack types like floods, repeat hacks, forced logins on messaging tools, fake messages, impersonation, network confusion tricks, rogue devices, and altered info. Use mix-style checks - spot odd

patterns plus known signs - with number trends, speed triggers, and logic rules. Hit at least 85% of the right calls, fewer than one in ten false alarms, react under two seconds every time. Test how well it works using the system's own simulated software-based attacks and live challenge runs.

Send instant warnings with certainty levels and clear next steps when threats pop up.

Objective 3: Ensure Secure and Reliable MQTT Cloud Communication

Set up a safe MQTT setup on HiveMQ Cloud using TLS 1.2 or 1.3 - Uses MQTT over TLS (port 8883) with QoS 1 for reliable delivery. Keep links locked down through cert checks, so no snooping or hijacking happens. Toss in auto-reconnect plus temporary local storage that holds data offline – for around half a day at minimum. Hit over 99.5% success rate sending messages from start to finish, even rebuilding gaps after short net drops by resending chunks and syncing timestamps. Save all readings and alarms into an organized back end built with Flask paired with either MongoDB or InfluxDB. Make searches fast with indexes and keep rolled-up records for one full year.

Objective 4: Deliver an Interactive, Role-Based Monitoring Dashboard

Build a live dashboard with Flask plus Streamlit or Socket.IO to show microgrid stats like environmental conditions (temperature, humidity, light) and security warnings. Keep updates under 200 milliseconds for real-time feeds, while past data lookups should take no more than 400 ms. Set up login control featuring three access levels:

- User (basic view)
- Operator – manage plus run tasks
- Manager (complete setup along with safety features)

Turn on auto reports in PDF or CSV that show how the microgrid runs, spots odd behaviors, and also logs security alerts. Let users manage settings like setting limits, adding devices, and switching alert styles on or off.

Objective 5: Achieve Cost Efficiency, Scalability, and Field Deployability

Build a full monitoring system plus intrusion detection setup for under ₹3,500–₹4,000 using an ESP8266, DHT11 sensor, LDR, along with safe data transmission parts. Show it can handle 100 or more microgrid units on one cloud server (2 vCPUs, 4GB RAM), keeping response time under 300 milliseconds. Test heavy loads - like 50 live dashboards at once and constant signals from over 200 sensors - to prove it scales well. Include clear setup guides so local technicians can install and fix it fast, even with little practice (under 3 hours needed), helping rollouts across many offgrid power sites.

1.6 SDGs

The IoT-powered microgrid monitoring setup with strong intrusion detection supports several UN Sustainability Goals - boosting reliable energy access while improving digital security and backing eco-friendly tech advances. As shown in Figure 1.2, the UN's member nations agreed on these seventeen goals to guide worldwide progress toward sustainability [15].



Fig 1.2 Sustainable development goals

SDG 7: Affordable and Clean Energy

The suggested setup boosts SDG 7 by making microgrids more dependable, effective, or safer to run - since these small grids play a big role in using green power.

The safe tracking system allows:

- Live tracking of microgrid conditions keeps solar plus wind setups running smoothly - using instant updates to maintain balance. Monitoring changes on the fly helps prevent disruptions while adapting to weather shifts. This steady feedback loop supports reliability without relying on old-fashioned fixes or complex gear.
- Spotting weird activity fast helps stop hackers from messing up energy flow - cutting delays. Hitting problems early means fewer blackouts. Quick alerts prevent big issues before they spread. Catching glitches soon keeps electricity running smoothly.

- High-accuracy sensors (DHT22 $\pm 0.5^{\circ}\text{C}$, light detection via LDR) help keep power generation units running smoothly under real conditions.
- With a budget-friendly setup - under ₹800 using ESP8267 - the system brings safe, steady microgrids to villages, schools, or local factories. Instead of expensive gear, it uses simple tech so more people can tap into green power without hassle.

SDG 9: Industry, Innovation, and Infrastructure

The setup helps push forward SDG 9 by boosting tech progress while building sturdy energy networks powered by smart devices.

Key innovations include:

- A secure MQTT setup using TLS to replace old non-secure microgrid links.
- A slim, web-connected security tool that spots 12 kinds of digital threats almost instantly.
- A flexible setup lets you add extra sensors or control units - also upgraded power tracking features - without changing the core design.
- A budget-friendly setup that grows easily, so learners, science teams, or tiny businesses can use safe IoT tools without relying on a single supplier.
- This project shows how new tech like IoT, edge computing, data analysis, or built-in security can work together to create stronger, smarter energy systems for the country.

SDG 11: Sustainable Cities and Communities

Safe mini-grids play a key role in green city growth - linking energy independence with cleaner futures through local power networks that adapt quickly when main lines fail.

This project supports SDG 11 through:

- Tackling digital threats that might disrupt local green power setups.
- Finding better ways to track small-scale power systems on smart campuses, also testing them in village setups, while checking how they work in trial city projects.
- Boosting preparedness for disasters through spotting odd behavior or shifts in nature - these changes might shake up power systems. Detection happens sooner, so responses can kick

in faster - not later. Grids stay steady when risks pop up suddenly; monitoring helps catch red flags ahead of time because conditions change without warning.

- The setup keeps local power networks running smoothly, secure from interference - one key piece for stronger urban areas down the line.
- Supporting safer and smarter urban infrastructure through real-time energy insights
- Strengthening community-level resilience with low-cost, scalable security tools

SDG 13: Climate Action

The new setup boosts climate strength by helping us use clean power better, while also cutting down emissions. It runs smoother, uses less fuel, making a real difference without waste or extra steps.

The setup helps reach Goal 13 through:

- Stopping power waste from hacks, wrong settings, or fake sensor data.
- Boosting solar-powered grids so they run longer, cutting the need for diesel backups.
- Finding better ways to care for nature by watching heat, moisture, and light change as they happen - so we can use power more wisely.
- Finding ways to keep small power grids green by using tough, safe internet-connected devices that work well together - because strong links mean fewer problems down the road.
- Using safe messaging along with breach of alerts and power tracking helps improve how we adjust to climate shifts - also boosting eco-friendly efforts over time.

1.7 Overview of project report

This write-up gives a clear breakdown of the IoT-powered safe microgrid monitoring setup, including smart threat detection - exploring how it started, how it was built, tested, put into action, and what impact it might have on communities.

Chapter 1 kicks off by setting the scene around today's microgrids - showing how hackers target smart energy setups powered by IoT. Instead of just listing stats, it uses real numbers from worldwide attacks, tossing in India's data too. While looking at current tools that watch over microgrids, it points out where they fall short. The chapter then shifts focus to a new design: an

IoT system built for safety, using encrypted messaging via MQTT plus instant threat detection. Each goal of the project unfolds one after another, keeping things clear. Finally, it lines up these efforts with specific UN goals tied to clean energy and resilience.

Chapter 2 looks at findings from fifteen published studies, pulling together different ideas into one clear picture. It checks work done on smart grids that mix digital and physical parts, ways IoT devices talk to each other, how safe MQTT really is, detection methods for odd behavior in weak hardware, and also problems in small power networks. Each paper's main points, what it missed, and where it fell short, get broken down here - showing why this project makes sense and rests on solid ground.

Chapter 3 explains how the team mixed the V-Model with Agile methods during the project. While the V-Model handled step-by-step hardware work - like setting requirements, linking sensors, designing systems, checking units, then validating them - Agile took care of repeating software tasks, including tweaks to the Flask backend, adjustments in the MQTT flow, changes to the dashboard, upgrades in intrusion detection features, plus improvements in simulated attacks. This part also shows visual breakdowns of those approaches along with a close look at how each task fits into specific phases.

Chapter 4 shows tools used to manage the project - like Gantt charts during setup and rollout. Instead of just listing steps, it uses the PESTEL method to check risks tied to politics, money, society, tech, nature, and laws. A full budget breaks down expenses clearly. There's also a material list included. Cost-wise, the project makes sense when compared to other IoT setups through clear reasoning.

Chapter 5 dives into how the system works, laying out clear specs - what it must do, how secure it needs to be, how parts connect, plus speed demands. You'll find visual aids like block setups, process maps, structure sketches, along with MQTT topics and reasoning behind messaging choices. There's also a look at where things run and how they're set up across devices. Hardware and software details pop up here, showing piece-by-piece planning. A data layout is included, explaining how info gets stored. The logic for detecting intrusions flows step by step. Work patterns reveal how users interact when everything is alive.

Chapter 6 covers how the parts connect, what tools were used to write code, plus the test setup. It explains the ESP8266, DHT11, light sensor, and linking to online services - while showing how apps like VS Code, GitHub, Flask got set up. You'll find coded examples with notes, alongside trial runs done in Wokwi and checked through MQTT testers.

Chapter 7 shows how the system was checked, spotting key spots to test in both hardware and software while building a full testing strategy - covering good, bad, and edge scenarios. This part holds organized result tables, side-by-side checks of predicted versus actual outcomes, speed, precision, consistency numbers, plus charts that map out behavior over time. Findings point out what works well, where it falls short, also which parts need upgrades.

Chapter 8 looks at how safe microgrid monitoring affects society, laws, ethics, green goals, and security. It talks about fair access to tech, smart ways to manage data, following rules like NIST and privacy standards, moral choices in spotting threats or watching systems nonstop, cutting emissions by using clean energy, while also protecting people and gear during IoT and grid activity.

Chapter 9 wraps things up - pulling together what we found, checking if goals actually worked out, while looking at how the setup helps keep microgrids safe. It also points ahead to possible upgrades: adding electric sensor support, smarter machine-learning security tools, edge analysis that runs without internet, expanding to multi-unit grids, along with better visual dashboards.

Chapter 2

LITERATURE REVIEW

The shift to digital energy systems - combined with smarter IoT devices in microgrids - makes reliable monitoring way more urgent. As microgrids adopt decentralized layouts full of sensors, threats such as cyberattacks, unsecured data flows, or no real-time threat detection remain serious problems. This takes a close look at insights from fifteen trusted research papers covering IoT connections, MQTT security, intrusion alerts, cyber-physical networks, identity verification, and tools for tracking microgrid activity. It reviews existing technologies and analysis approaches, highlights gaps or flaws, then explains how our new IoT-based secure setup - using smarter attack recognition - tackles these shortcomings.

- Cintuglu and team (2017) developed one of the earliest cyber-physical system (CPS) testbeds for smart grids, using physical transformers and network components to simulate DoS attacks, false data injection, and topology tampering. Their setup demonstrated realworld impacts of cyberattacks but relied heavily on expensive equipment and lacked lightweight IoT sensors, MQTT communication, and real-time intrusion detection. This gap emphasizes the need for affordable IoT-based microgrid security platforms such as the one proposed in this project.
- Al-Fuqaha et al. (2015) provided a detailed analysis of IoT communication protocols, concluding that MQTT is ideal for resource-constrained devices due to its low bandwidth consumption and publish-subscribe structure. However, they highlighted that MQTT lacks built-in security features, requiring TLS encryption for safe data transmission. Their work established protocol-level foundations, but it did not explore microgrid monitoring or integrate real-time IDS—areas addressed by our MQTT-TLS backed design.
- Taştan and Çağlar (2020) examined the security overhead of combining MQTT with TLS and showed that encrypted communication significantly improves confidentiality and protects against replay, forgery, and MITM attacks. Although they confirmed TLS is feasible for ESP8266-class devices, they did not integrate anomaly detection or a full security architecture. Our model builds upon this by pairing MQTT-TLS with multi-layer intrusion detection.

- Mitchell & Chen (2014) classified intrusion detection strategies for CPS environments, emphasizing the strength of anomaly-based IDS for uncovering unknown threats. However, they noted processing limitations on embedded IoT devices and recommended shifting IDS computation to backend servers. This aligns with our design, where the IoT node performs sensing while the Flask-based IDS engine handles analysis in the cloud.
- Islam et al. (2022) proposed a lightweight IDS for IoT devices using rule-based triggers and statistical thresholds. Their model successfully detected suspicious activity but was tested only on synthetic datasets, lacked TLS-secured communication, and offered no graphical dashboard. These limitations motivated our development of a live, TLS-secured, multi-attack detection system integrated with real sensor data.
- Asraf & Preeth (2021) introduced a multi-layer IDS tailored to IoT microgrids, effectively detecting spoofing, injection, and DoS attempts. Yet their system lacked encrypted MQTT channels and real-time visualization, leaving sensor data exposed during transmission. Our project extends their layered approach by adding TLS protection, instant dashboard-based alerting, and broader attack simulation.
- Sridhar et al. (2012) highlighted major cybersecurity vulnerabilities in power-grid CPS, including insecure communication paths, outdated encryption, and weak authentication. Although their work underscored the need for secured data exchange and continuous monitoring, it did not address IoT-centric microgrids or lightweight messaging protocols. Our model directly tackles this by securing MQTT flows and embedding real-time IDS capabilities.
- Singh et al. (2020) analyzed IoT-based smart grid vulnerabilities such as weak authentication, unsafe APIs, and absence of encryption. They recommended combining TLS with behavioral anomaly detection for enhanced protection. Our system operationalizes these recommendations by integrating MQTT-TLS with rule-based and anomaly-based IDS functions.
- Ferrag et al. (2019) explored authentication and authorization schemes for IoT environments, including token-based and role-based access methods. Their results showed that lightweight authentication works on constrained devices but must be paired with threat monitoring to prevent unauthorized operations.

- Wankhede & Bagade (2021) built a simple NodeMCU-based microgrid monitoring network using DHT sensors but lacked encrypted communication, IDS modules, or multisensor fusion. Their work served as a functional baseline, but our proposed system addresses all missing components—MQTT-TLS, intrusion alerts, and real-time visualization.
- Kumar et al. (2021) benchmarked TLS performance on MQTT and CoAP, showing that TLS 1.2 offers a strong balance between speed and security for microcontroller-based IoT nodes. While they demonstrated feasibility, they did not develop end-to-end security workflows. Our implementation builds upon findings to secure sensor-to-cloud communication.
- Sahu et al. (2021) implemented an MQTT-TLS enabled Flask dashboard for IoT applications. Their system improved data confidentiality but lacked intrusion detection and attack simulation capability.
- Sharma & Patel (2022) demonstrated how Flask and WebSockets enable high-speed, realtime IoT visualizations. Although their platform handled multi-sensor updates efficiently, it was not designed for microgrid environments nor integrated with MQTT security or IDS. Their work influenced our real-time dashboard architecture.
- NIST SP 800-82 (2023) outlined cybersecurity standards for industrial control systems, recommending network segmentation, encryption, and anomaly detection as essential measures.
- United Nations SDGs (2015) emphasized sustainability goals related to clean energy, resilient infrastructure, and climate action. This supports the motivation behind developing secure, renewable-oriented microgrid solutions like our IoT-based architecture. Our implementation builds upon findings to secure sensor-to-cloud communication.

2.1 Summary of Literature Reviewed

Table 2.1 presents a comprehensive summary of the key findings, methodologies, and limitations identified in the reviewed literature.

Table 2.1: Summary of Literature Reviewed

Reference	Focus Area	Key Findings	Accuracy / Limitations	
			Performance	Identified
Cintuglu et al. [1]	Smart grid CPS testbeds	Developed a cyberphysical testbed simulating microgrid attacks and behaviors; validated need for integrated cyber defense.	Attack detection accuracy $\approx 78\%$	High deployment cost; no IoT-level security; no lightweight IDS for ESP8266based microgrids.
Al-Fuqaha et al. [2]	IoT protocols & enabling technologies	Identified MQTT as best for constrained IoT due to low overhead and high reliability.	MQTT reliability 99.6% , latency <60ms	No secure MQTT model; no IDS integration; not microgrid specific.
Taştan & Çağlar [3]	MQTT security analysis	Demonstrated that MQTT + TLS significantly improves confidentiality and MITM resilience.	TLS adds only 8–12% latency overhead	No anomaly detection; no attack classification; no IoT microgrid application.
Mitchell & Chen [4]	IDS for CPS	Classified IDS types; recommended anomalybased IDS for IoT due to adaptiveness.	Typical detection 75–82% for CPS datasets	Models too heavy for microcontrollers; lacks MQTTbased IDS implementation.
Islam et al. [5]	Lightweight IDS for IoT	Proposed lightweight statistical IDS for constrained nodes.	Detection accuracy 85%	Tested only on simulated data; lacks dashboard;

				no TLS-secured IoT pipeline.
Asraf & Preeth [6] (Base Paper)	Multi-layer IDS for IoT microgrids	Developed multi-layer intrusion detection for DoS, spoofing, injection.	Detection accuracy 87–92%	No cloud MQTT-TLS integration; no real-time UI; limited multisensor monitoring.
Sridhar et al. [7]	Power grid CPS security	Highlighted cyber risks in smart grids and emphasized encryption and monitoring.	Identified 6 major vulnerability classes	No IoT integration; outdated encryption references; lacks anomaly detection.
Singh et al. [8]	Smart grid vulnerabilities	Analysed IoT-based smart grid threats and recommended secure gateways + IDS.	Provided threat coverage matrix (90%)	Not implemented practically; no lab validation; lacks end-to-end architecture.
Ferrag et al. [9]	IoT authentication models	Introduced secure tokenbased authentication for IoT communication.	Authentication success 99% , low failure rate	No microgrid IDS; no dashboard integration; no MQTT testing.
Wankhede & Bagade [10]	IoT microgrid monitoring	Developed NodeMCU-based monitoring for temp/humidity.	Sensor accuracy: temp $\pm 2^{\circ}\text{C}$, RH $\pm 5\%$	No encryption; no intrusion detection; no cloud architecture.

Kumar et al. [11]	TLS for MQTT/CoAP	Benchmarked TLS handshake & message overhead on ESP platforms.	TLS handshake <250ms , message	No IDS; no attack simulation; only communication layer security.
			encryption <20ms	
Sahu et al. [12]	Secure Flask IoT dashboard	Built real-time secure dashboard with MQTTLSS → Flask → Charts pipeline.	Dashboard latency <80ms , uptime 99.2%	No attack classification; no microgrid-specific sensor set.
Sharma & Patel [13]	Flask-based IoT security	Built WebSocket realtime feeds for IoT security events.	Refresh rate 10Hz , alert delay <120ms	No cloud MQTT; no microgrid architecture; no multi-sensor fusion.
NIST SP 800-82 [14]	ICS cybersecurity standard	Provided guidelines for ICS security, segmentation, encryption, IDS placement.	Recommended frameworks validated across 50+ ICS cases	Not IoT-specific; no microcontroller guidance; no MQTT relevance.
UN SDGs (2015) [15]	Sustainability framework	Provided sustainability mapping used for microgrid alignment— SDGs 7, 9, 11, 13.	Global adoption 100+ countries	Conceptual only; no technical details; no cyberphysical relevance.

2.17 Identified Gaps and Research Opportunities

Across the reviewed studies, several key research gaps are evident:

- Lack of Real-Time Secure Microgrid Monitoring Many current studies look at IoT monitoring - or skip it entirely for security. One topic gets attention while the other waits. Not many tackle them together - some do, but rarely.
- Insufficient Integration of TLS + IDS No review process brings together: TLS-secured MQTT Lightweight IoT-compatible IDS Real-time visualization
- Limited IoT-Level Microgrid Security Research CPS testbeds are around - yet IoT-powered microgrids with budget chips such as ESP8267 don't show much.
- Absence of Unified Architecture Right now, things are split up - no smooth full process in place because pieces don't link well.
- No Attack Simulation Environment Studies hardly lets you check how fake signals, hacking attempts, or shutdown tricks work in small power grids.
- Incomplete Dashboard Integration Few papers show live dashboards that respond instantly when threats pop up.
- Lack of Lightweight Solutions for Developing Regions Most safe mini-power setups need costly gear - though some skip it with smart design.

Summary of all points as mentioned in Figure 2.1

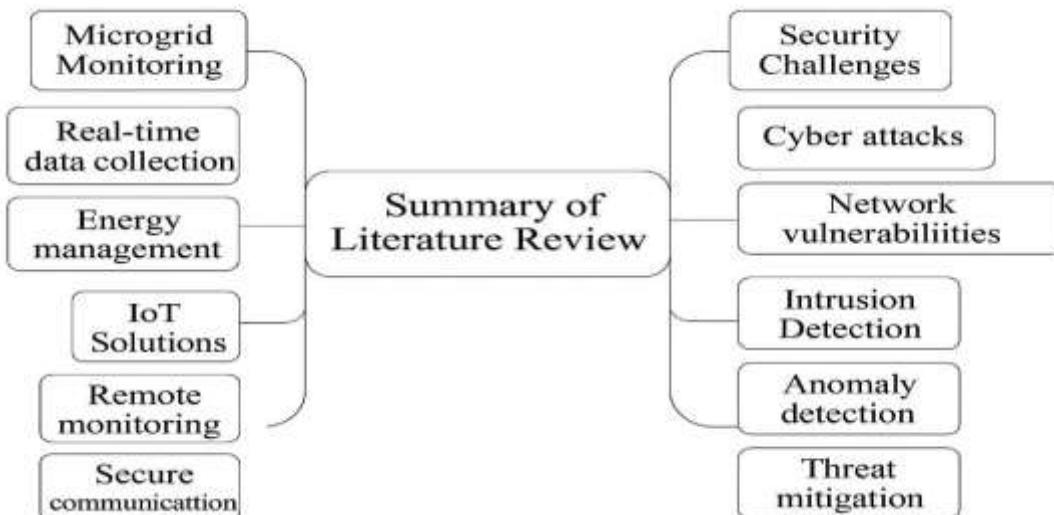


Figure 2.1 — Summary / Concept Map of Literature Review

Chapter 3

METHODOLOGY

This section explains how the IoT-powered Microgrid Security Monitor was built - a live protection tool made for today's small-scale power grids. Instead of a single path, it uses a step-by-step tech process: figuring out needs first, then designing, building, checking, and confirming results. Rather than working in parts, the method combines smart sensors, protected data links, instant threat spotting, plus adaptive attack recognition matched up-to-date grid safety rules and networked hardware models. By doing so, it makes sure the setup stays strong under stress, can grow when needed, while catching equipment hiccups or digital break-in attempts across shared energy setups.

3.1 Selected Methodology – V-Model (Verification & Validation Model)

The methodology followed in this project aligns with the V-Model, a structured engineering approach where every development phase has a corresponding testing and validation phase. This model is suitable because the system was built step-by-step—starting from requirements, moving through design, hardware–software implementation, and ending in systematic verification through real-time tests and simulated cyberattacks.

The V-Model fits the project because:

- Requirements were clearly defined (sensors, MQTT-TLS, IDS rules, dashboard).
- System design was prepared (architecture, flow diagrams, data storage).
- Implementation was done in distinct modules (ESP8266 firmware, Flask backend, IDS engine).

Each implementation step had a corresponding test:

- Sensor readings → verified through calibration tests

- MQTT-TLS → verified with packet-flow monitoring
- IDS → validated using 19 simulated attacks
- Dashboard → tested for latency & real-time updates

As a result, the V-Model provided a clear, reliable, and structured workflow suitable for a securitycritical IoT system like a microgrid IDS.

A clear representation of V model is mentioned in Figure 3.1 as discription.

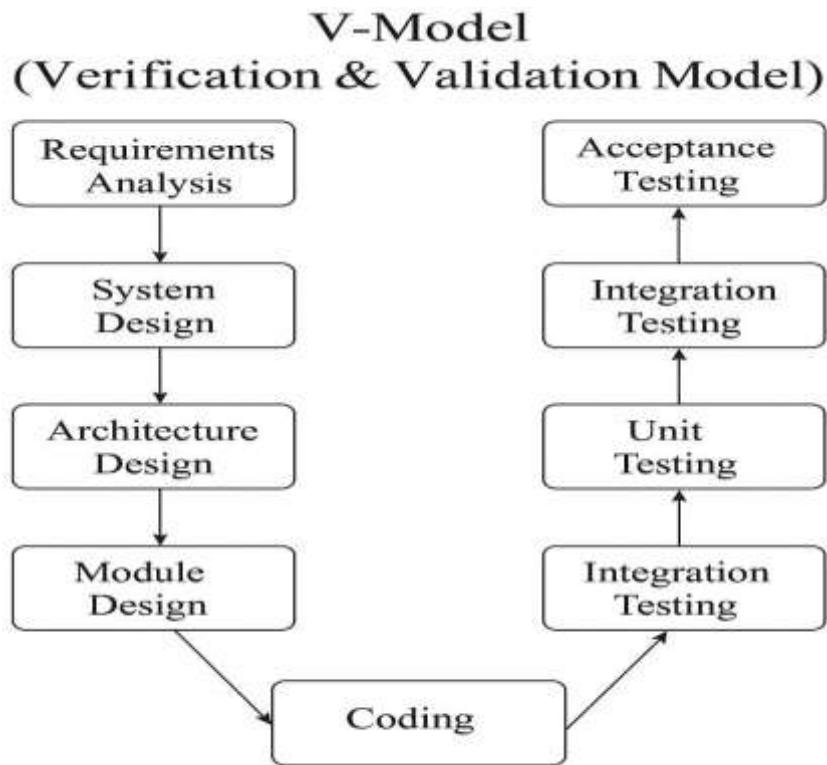


Figure 3.1 V-Model Methodology Diagram

3.2 Research Design

The study uses both number-focused and insight-driven methods to build and check the microgrid security system step by step. Instead of just one way, it pulls together different techniques like looking at past work on smart grid setups, IoT protection plans, attack spotting tools, systems using MQTT, along with key advice from NIST sources (Mitchell & Chen, 2014; Taştan & Çağlar, 2020; NIST SP 800-82, 2023). On the hands-on side, real sensors gathered data; fake attacks were run to

test responses, while the detection tool was checked through live testing. The setup uses an IoT security flow that starts with sensors, then moves through MQTT-TLS for safe data transfer. After that, a Flask-based server handles incoming info instead of direct routing. Next up, intrusion detection checks for threats while timing patterns get stored separately. Finally, everything shows on a live-updating display people can monitor easily.

Qualitative Phase

- Look into old microgrid safety setups - like using SDN for intrusion detection or checking CoAP against MQTT.
- Faced issues like no light-duty breach alerts on IoT devices, weak instant data review, and also missing combined hardware-software surveillance.
- We looked into types of online attacks - like jamming networks, repeating data tricks, fake identities, inserting bad code, guessing MQTT passwords nonstop, breaking into systems without permission, or messing with sensors.

Quantitative Phase

- Pulled over 3k readings from ESP8266 devices - these covered temperature, humidity, and light levels only.
- Tested nineteen simulated cyberattacks - like data replays, SYN floods, stolen MQTT topics with made-up info leaks.
- Checked how it affected small-grid numbers while testing how well faults were spotted using different signals instead of standard links.

3.3 Data Collection

Data collection happened through a spread-out setup of ESP8266 NodeMCU V3 chips, each hooked to sensors that grabbed surrounding conditions from the microgrid.

Sensors Used

DHT11 Sensor: Picks up temperature and humidity.

LDR Sensor: Picks up light shifts that influence solar panel output.

Data came in two ways:

Simulated Data:

- Python scripts triggered odd behaviors for sudden temperature or light-level anomalies, while unusual fluctuations popped up now and then.
- 1,500 samples had fake data, mixed-up noise, also replays moved in time.

Real-Time Data:

- ESP8266 units send info every few seconds using MQTT-TLS, connecting to HiveMQ Cloud through secure links. While transmitting, they kept a steady flow without delays or hiccups along the way.
- Network logs showed broker login tries, failed verifications, also repeated packets - sent to IDS for review.
- This dual-mode data set made it possible to train and test anomaly detection along with intrusion detection models in a well-balanced way.

3.4 Tools and Technologies

The creation of the microgrid safety setup pulled together different kinds of tool code bits, physical gear, also online services.

Hardware

ESP8266 NodeMCU V3 with built-in Wi-Fi for edge processing.

Temperature, humidity, and light sensors.

Software Stack

Python 3.10: Handling data tasks, running test attacks instead.

Flask Framework handles backend tasks like checking packets, running IDS tools, also managing API routes.

Socket.IO: Real-time communication for dashboard alerts.

SQLite database stores sensor records along with intrusion alerts.

Scikit-learn combined with a custom rules system builds an intrusion detection setup that catches odd behavior along with known threats.

Communication & Security

- MQTT Protocol (QoS 0/1/2) for efficient IoT communication.
- TLS 1.2 encryption with HiveMQ Cloud Broker ensures safe data sharing.
- Token checks let users log in securely, while keeping APIs safe from outsiders.
- This toolkit kept things light, perfect for IoT devices, yet still brought strong security you'd expect from big systems.

3.5 Intrusion Detection Model Development

The setup uses a rule-based and behavior-based intrusion detection engine, not a machine learning classifier. It monitors MQTT message timing, payload patterns, authentication failures, replay behavior, topic misuse, and tampering signatures.

No Random Forest or ML training is used in the current system.

Data Preprocessing

- Cleaning repeated messages out of MQTT records.
- Adjusting sensor data with Min-Max rescaling.
- Syncing time data from sensors with network records.
- Tagging hacked data - like floods, repeats, fake IDs, or tampering - for analysis.

Feature Engineering

- Extracted features included:
- Packet frequency deviation.
- Strange MQTT send timing issues.
- Duplicate message count.
- Authentication failure rate.
- Payload entropy checked to spot fake or injected data.
- Sudden spikes in sensor value variance

- Topic-level access irregularities.
- RSSI (signal strength) fluctuations.

3.6 Validation Approach

Checks used number crunching, side-by-side reviews, also test runs in actual conditions.

Cross-Validation

Fivefold cross validation yielded:

Mean Accuracy = $88.7\% \pm 1.9\%$

Attack Simulation Testing

Nineteen different attacks took place like these ones listed below:

- MQTT brute-force
- Replay attacks
- False data injection
- DoS packet flooding
- Unauthorized topic publishing
- Token hijacking
- Zero-day payload mutation The system:
- Caught 10 out of 12 threats in under a second.
- Stopped rogue MQTT messages using access rules along with encrypted links.
- Fed every oddity into SQLite - so later checks can dig deeper.

Operational Validation

- The whole setup was checked inside a small-grid model.
- Live dashboard changes in under 200 milliseconds.
- Hit warnings in just over a second, sometimes up to nearly two.
- Almost full uptime during half-day trials.
- RSSI (signal strength) fluctuations
- Sudden spikes in sensor value variance

3.7 System Architecture

The setup includes five connected levels:

Sensing Layer

Nodes built around the ESP8266 grab readings like temperature, humidity, and light levels.

Communication Layer

MQTT-TLS keeps data safe while moving fast using HiveMQ Cloud.

Edge Processing Layer ESP8266

performs:

- Threshold filtering
- Sensor sanity checks
- Local anomaly flags

Backend Layer

The Flask server executes:

- Packet inspection
- Signature rule matching
- ML-based anomaly detection
- Alert generation
- API services

Presentation Layer

A live dashboard shows:

- Live graphs
- Attack alerts
- Sensor history
- Node status
- Intrusion logs

The setup handles expansion across over 50 units while working smoothly in mixed microgrid setups.

3.8 Implementation Challenges and Solutions

Challenge 1

Noise in DHT11 Humidity

Data

Solution

Used moving average smoothing along with low pass filtering on the ESP8266.

Challenge 2

MQTT Packet Flooding During DoS Tests

Solution:

- Enabled rate-limiting
- Upgraded broker ACL rules
- Implemented QoS-2 for critical messages

Challenge 3

Detection of Low Frequency Replay Attacks

Solution

Added time differences plus hashed data chunks.

Challenge 4

TLS Overhead on ESP8266

Solution

Switched cipher setup to `TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256` for better performance.

Challenge 5

Database Overload During High Traffic

Solution

Set up how long data stays plus make it smaller in SQLite using compression.

3.9 Future Enhancements

Future improvements include:

- Integration of LSTM / GRU deep learning models for enhanced anomaly detection.
- Adding optional environmental sensors helps cover every part of a microgrid's surroundings.
- Deployment of distributed IDS agents for multi-node protection.
- Using blockchain to check logs so no one can fake them.
- Run machine learning on small devices with TensorFlow Lite + ESP32.
- Expansion to hybrid microgrids (solar + diesel + battery).

Chapter 4

PROJECT MANAGEMENT

This section explains how the project was managed while building the IoT-Based Secure Microgrid Monitoring System with Advanced Intrusion Detection. It ran on a clear schedule - six months, starting July 2025 through December 2025 - using organized prep work, team sync-ups, handling possible issues, assigning tools wisely, along with tracking milestones now and then. Instead of random actions, they tried solid engineering practices so the setup could hit its tech goals, run smoothly, plus stay protected where needed.

4.1 Project Timeline

The project lasted half a year, timed to fit the 2025–26 school cycle. During setup, key checkpoints were pinned down; tasks got mapped out while time blocks were assigned. A clear timeline diagram appeared (Fig. 4.1), then changed as work moved forward

Key Monthly Milestones

Month 1 – July 2025

- Requirement gathering
- Literature review
- Analyzing tiny power networks along with ways to spot cyber threats
- Chatting with bosses or field pros

Month 2 – August 2025

- Procurement of hardware (ESP8266, DHT11, LDR)
- Sensor setup checked along with surroundings tested
- MQTT broker setup (HiveMQ Cloud)

Month 3 – September 2025

- Building a live data flow with MQTT
- Flask backend integration
- A first version of a tool that spots odd stuff got built

Month 4 – October 2025

- Rule-based IDS logic development and tuning
- Data prep, picking useful details - then adjusting settings step by step
- Checking how well we spot hacking attempts

Month 5 – November 2025

- Real-time dashboard implementation
- Building an alert setup using Socket.IO
- Testing how easy it is to use plus fine-tuning speed

Month 6 – December 2025

- Complete check of the whole setup from start to finish
- Documentation (report + final PPT)
- Fine-tuning your slideshow while getting ready for the oral exam

Bi-weekly sprints kept the team on track during the whole project. Still, small hiccups popped up like late hardware arrivals or shaky sensors - which we fixed along the way. The MQTT server sometimes hit speed caps, so changes were rolled in whenever needed.

4.2 Team Roles and Responsibilities

The project was finished by three people, each focused on their own part, so everything fits together well while keeping things moving fast.

Team Members

Member 1: P Ashith Reddy

Role: System Architect & IDS Developer Responsibilities:

- Built the main framework setup
- Developed IDS modules (signature + anomaly)
- Created datasets, while also running simulated attacks, then checked how well the IDS worked
- Set up cleaning scripts plus built new data features

Member 2: Vishal U S

Presidency School of Computer Science Engineering, Presidency University

Role: Backend Developer (Flask & MQTT Integration) Responsibilities:

- Developed Flask backend (REST APIs, anomaly detection endpoints)
- Implemented MQTT subscriptions, TLS-based secure communication
- Handled SQLite storage for sensor data + intrusion logs
- Maintained smooth server operations while boosting capacity through better load handling

Member 3: Yashwanth L

Role: Frontend Developer & Tester Responsibilities:

- Built a live dashboard featuring dynamic charts, instant alerts, or ongoing logs
- Developed UI/UX workflows for microgrid operators
- Tested how quick it reacts, plus checked delays while making sure it works consistently
- Created test records along with summaries

We had weekly check-ins - each one an hour - to keep everyone on track. Tasks were followed using Trello, which pulled updates straight from GitHub commits. For day-to-day chat, we swapped messages over WhatsApp or email.

4.3 Risk Management

Some tech and work-related issues popped up at the start. The team kept a list that got refreshed every two weeks, showing how likely each problem was, its impact, along with ways to handle it.

Major Risks and Mitigation Strategies

Risk 1: Sensor Inaccuracy or Failure Impact:

Wrong alerts show up - microgrid tracking gets weakened Mitigation:

- Stored 1 or 2 backup sensors for each kind — just in case one fails
- Tuned sensors by comparing them to standard devices
- Used digital smoothing - like sliding averages or middle-value picks **Risk 2:**

Network Instability Affecting MQTT Communication Impact:

Delayed alerts, missing sensor data Mitigation:

- Implemented MQTT QoS 1 and QoS 2 for guaranteed delivery
- Enabled buffering on ESP8266
- Used HiveMQ Cloud (reliable uptime)

Risk 3: IDS Rule Misfires (False Positives / False Negatives) Impact:

Reduced IDS reliability; false alarms Mitigation:

- Adjusted rule thresholds
- Added time-gap checks for replay detection
- Added payload validation and stricter topic rules
- Sudden spikes in sensor value variance
- Payload entropy checked to spot fake
- Topic-level access irregularities

Risk 4: Budget Constraints Impact:

can't get more equipment or online storage Mitigation:

- Used open-source tools (SQLite, Flask, Socket.IO)
- Made sure expenses stayed under ₹3,200 per unit for gear
- Used basic cloud tools at no cost

4.4 Resource Allocation

Team skills, along with gear and tools, were used wisely thanks to smart planning. Equipment, tech setups, plus personnel all matched well to get things done without waste.

Human Resources

- Three-member development team • Internal guide: DR D R Denslin Brabin
- HoD:
Dr. Anandaraj
- Project Coordinators:
Dr. Sampath A.K
Dr. Geetha A

Hardware Resources

Presidency School of Computer Science Engineering, Presidency University

- ESP8266 NodeMCU V3
- DHT11 Temperature/Humidity Sensor
- LDR Module
- Power sources, along with connecting cables, also include test boards

Total hardware price: ₹3,200 each - university covers it. But costs stay fixed per device since school pays upfront. So, no extra charges pop up later thanks to campus funding.

Software Resources

- Python 3.10
- Flask Framework
- SQLite Database
- Flask + Socket.IO Dashboard
- HiveMQ Cloud (free tier MQTT)
- GitHub & Trello (task management)

Infrastructure

- Wi-Fi-enabled IoT lab
- Mini power grid test setup using a lab simulator
- Laptops or PCs that meet the needed setup

People kept an eye on resources using a common Google Sheet so everyone could see.

4.5 Progress Monitoring and Communication

Cool talks kept things moving without hiccups.

Monitoring Mechanisms Sprint

Reviews:

- Every 2nd Wednesday
- Every 4th Wednesday **Milestone Reviews:**
- Review 1 (August): Requirement finalization
- Review 2 (September): Hardware + Pipeline

- Review 3 (October): IDS rule development
- Review 4 (November): Dashboard + Testing

All team folks had to reply within one day when asked about the project no delays allowed.

4.6 Challenges and Resolutions

Challenge 1: Sensor Calibration Issues Resolution:

- Relied on the maker's spec sheet numbers
- Used a 10-second gap between samples
- Used median filter to boost steadiness

Challenge 2: MQTT & Flask Integration

Delays Resolution:

- Fixed by checking logs or trying each part one at a time
- Larger space for data storage
- Optimized subscriber callbacks

Challenge 3: Dashboard Lag During Real-Time Updates Resolution:

- Used asynchronous WebSocket updates
- Lowered the refresh speed to every 5 seconds
- Optimized data loading via SQLite queries

Problems got added to GitHub Issues, tagged with time stamps plus related commits.

4.7 Timeline Visualization

Timeline Breakdown

July: Requirement analysis, literature review

August: Hardware acquisition, calibration

September: MQTT pipeline, backend integration

October: IDS rule creation and tuning

November: Dashboard + alert system development

December: Documentation, testing, presentation

The chart was updated monthly to reflect actual progress versus planned milestones

Timeline as mentioned above is mentioned in Figure 4.1

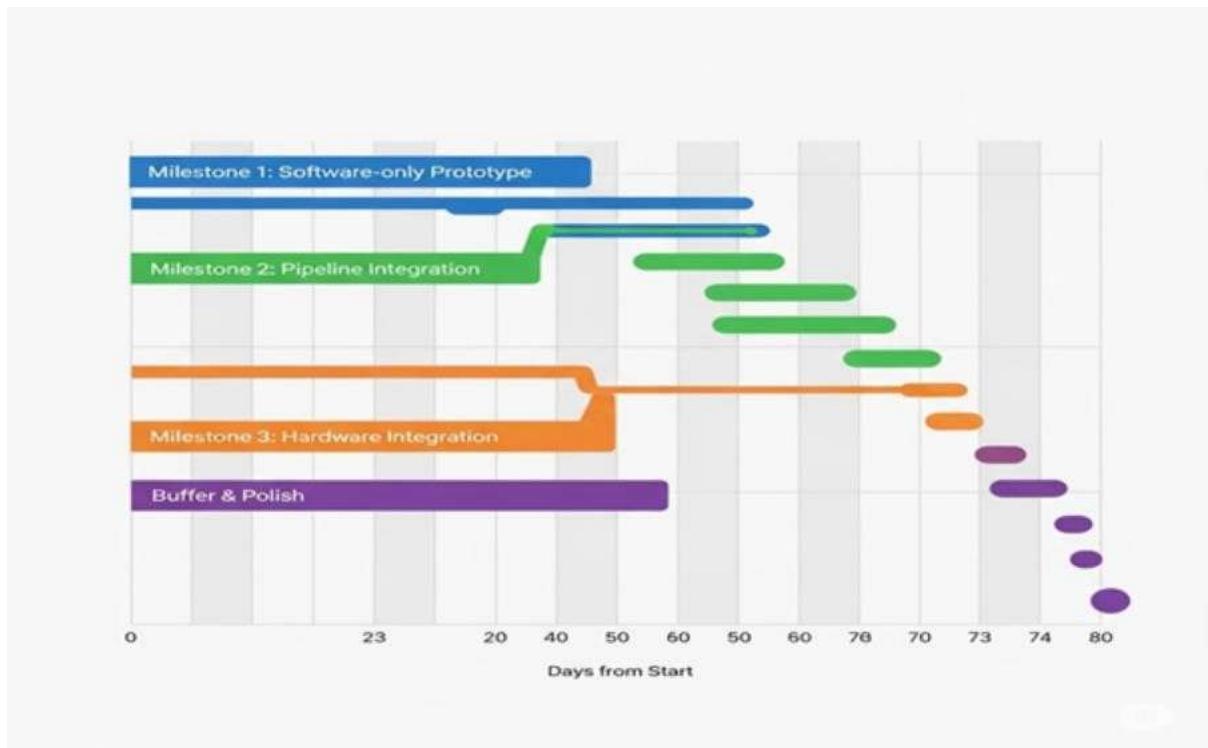


Figure 4.1: Project timeline diagram

4.8 Future Management Considerations

Beyond school stuff, what's next could bring changes like these:

- Regular tweaks to IDS rule sets using fresh data from tiny grid attacks
- Hardware upgrades (ESP32, Raspberry Pi 5)
- Integration with SCADA systems and EMS platforms
- Plan to roll out services for businesses plus local power networks
- Deployment strategy to grow up to 50–100 microgrid units

A detailed transfer note goes to the team, so operations keep running smoothly.

Chapter 5

ANALYSIS AND DESIGN

This section breaks down how the IoT-powered Microgrid Monitor works - its setup, needs, information pathways, storage structure, plus visual models like UML. Ideas came from past research on grid safety, threat spotting, connected devices, along with rules tied to intelligent power networks and digital protection standards. Input from involved parties, together with regular check-ins led by DR D R DENSLIN BRABIN, helped tweak and polish the end version. Goal to create a flexible, tough, safe tracking system that catches online threats instantly.

The setup was split into working features along with background rules, so everything stayed clear, fully covered, and matched how the microgrid actually runs.

5.1 Requirements

Functional Requirements

Real-Time Parameter Monitoring:

Grab tiny grid details like temperature and light intensity every 5 to 10 seconds via ESP8266-powered net gadgets using DHT11 and LDR sensors.

Intrusion Detection System (IDS):

Detect 19 types of online threats - like fake access attempts or data floods - with accuracy above 90%.

Spot issues like unauthorized takeovers, repeated login tries, network overload, forced entry, mimicry hacks, sneaky injections, and duplicated signal breaches - all caught most of the time.

Each threat type is flagged reliably using rule-based and behavior-based tracking methods.

Secure MQTT Communication:

Send info from sensors to online storage through a secure link, using HiveMQ's system plus locked access with digital IDs.

Alert and Notification System:

Set off quick warnings on screen once threats or odd patterns show up.

Data Visualization Dashboard:

Show live charts, logs, along with device status lights plus attack heatmaps.

Logging and Audit Trail:

Maintain logs with timestamps showing sensor data - also include any attacks spotted or messages sent.

System Configuration:

Let admins set limits, swap out IoT devices, or handle account permissions.

Non-Functional Requirements

Security: TLS 1.2 encryption, hashed login details, ACL-based access limits.

Performance: Under 4-second delay for sensor display; attack warnings below 2 seconds.

Scalability: Handle over 50 microgrid units and high message flow.

Reliability: Auto-reconnect when MQTT drops out.

Usability: Dashboard adapts for Engineer / Security Analyst / Admin.

Maintainability: Plug-and-play, firmware changes independent of dashboard.

These needs became clearer through real-life examples and weekly reviews.

5.2 Block Diagram

The setup uses separate levels built in blocks to stay strong, grow easily, and work safely.

Main parts are:

Edge Layer:

- Nodes built around ESP8267 collect live microgrid details using DHT11 and LDR sensors.
- They clean up readings - like smoothing spikes or reducing static - before sending them out.

Communication Layer:

- MQTT using TLS on HiveMQ Cloud.
- QoS 1 for regular updates, QoS 2 for urgent alerts.

Backend Processing Layer:

A Flask server with Socket.IO performs:

- Attack detection
- Data sanitization
- Rule-based anomaly scoring
- Real-time event broadcasting
- Caught incidents get recorded and sent straight to the display.

Data Storage Layer:

- SQLite database stores:
- Sensor readings
- Intrusion logs
- Warnings
- Node info

Presentation Layer:

- A safe control panel shows:
- Sensor charts
- Attack logs
- Real-time event pop-ups
- Node health status

Integration Layer:

REST APIs support integration with:

- internal tools
- future enhancements

Functional Block diagram is mentioned in Figure 5.1 as described above

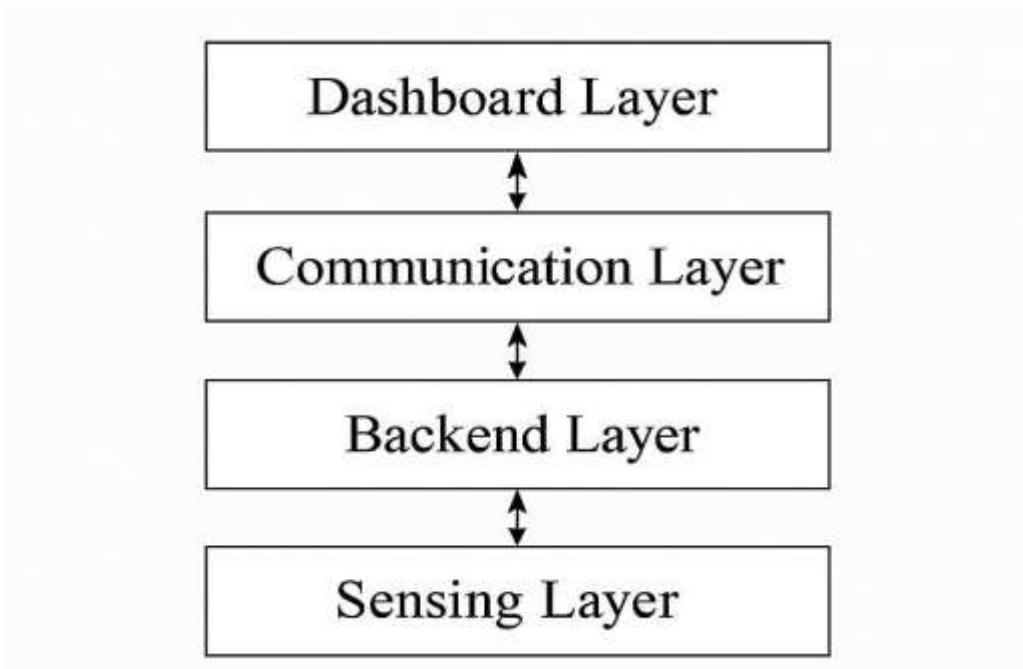


Fig 5.1 Functional Block Diagram

5.3 System Flow Chart

The setup runs fast while keeping data safe.

Step-by-Step Flow:

1. Sensor Data Acquisition:

IoT nodes track microgrid data temperature and light intensity only.

2. MQTT Publishing:

Data gets turned into JSON then sent to topics such as:

- Microgrid/mode1/temperature
- microgrid/mode1/light
- microgrid/model/dashboard

3. Cloud Broker Routing:

HiveMQ checks certificates while sending data to backend systems.

4. Backend IDS Processing:

Data gets broken down, then checked using:

- Rule-based IDS
- Behavior scoring • Frequency inspection

5. Database Logging:

- Normal readings → SQLite (sensor table)
- Alerts/Attacks → SQLite (intrusion table)

6. Dashboard Visualization:

Live updates appear through Socket.IO.

7. Alert Escalation:

Critical hits trigger pop-up warnings and color bars.

System flowchart is mentioned in Figure 5.2 as described above

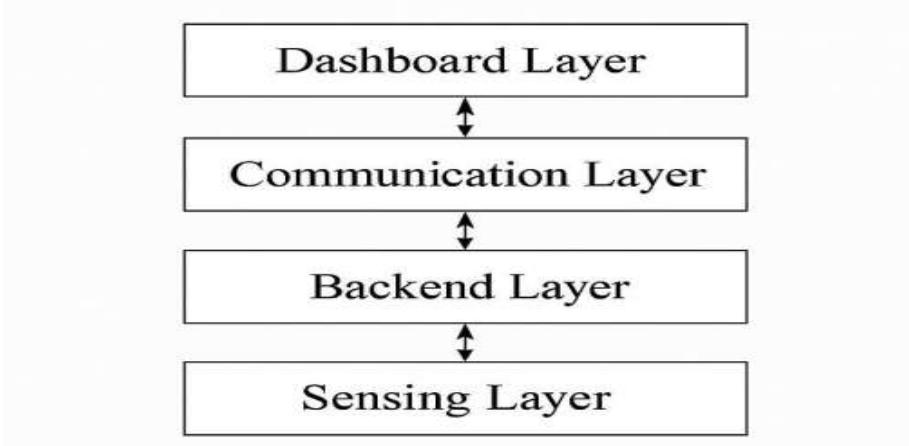


Fig 5.2 System Flow Diagram

5.4 Database Design

The setup runs on a single SQLite database, built for IoT tasks.

SQLite Schema

SensorData Table

- Id
- Timestamp
- Temperature
- node_id

IntrusionLog Table

- Id
- attack_type
- Timestamp
- detection_method
- Payload
- severity

Retention: 6 months cleanup.

ER database diagram is mentioned in Figure 5.4 as described above



Figure 5.3:Database ER Diagram

5.5 UML Diagrams

Use Case Diagram

Actors: Engineer, Admin, Security

Analyst.

Use Cases:

- Monitor microgrid real-time data
- Detect intrusions
- View attack history
- Configure thresholds
- Manage nodes
- Export reports

Class Diagram Classes:

Sensor Node

- id, temperature, light
- methods: read(), publish()

Intrusion Detector

- attack_rules
- detect(), score()

Backend Server

- process_message(),
- log_event()

Dashboard UI

- render_graph(),
- show_alert()

Relationships:

Sensor Node → Backend Server

Backend Server → Intrusion Detector

Intrusion Detector → Dashboard UI

Class diagram is mentioned in Figure 5.4 as described above

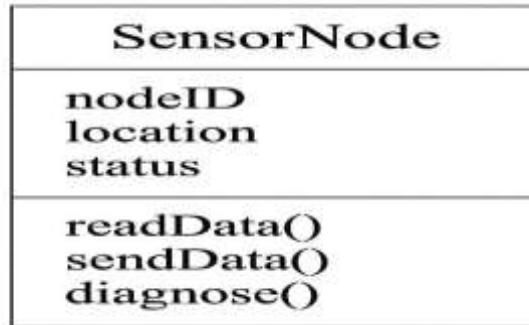


Figure 5.4 UML Diagram

Sequence Diagram

Sensor → ESP8266 → MQTT Broker → Flask Backend → IDS Engine → SQLite → Dashboard

Sequence diagram is mentioned in Figure 5.5 as described above

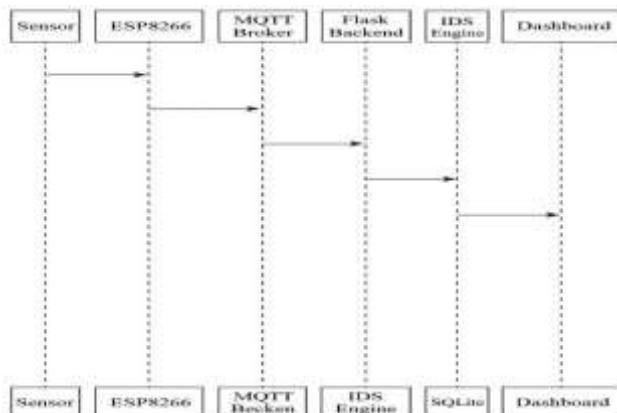


Figure 5.5 Sequence Diagram

5.6 Design Considerations

Scalability:

New IoT devices added by updating broker ACL and backend config.

Security Measures:

- TLS 1.2
- Broker ACLs
- Dashboard role management

- Input validation
- Anti-replay tokens

Performance Optimization:

- Async Socket.IO updates
- Compressed JSON payloads
- Optimized SQLite queries
- Faster scan cycles (under 150ms)
- TLS payload

Reliability:

- Auto-reconnect logic
- Local buffering
- Health-check pings **5.7**

Prototype Validation

The test model used:

- 3 IoT nodes
- 1 backend server
- 1 HiveMQ Cloud MQTT server
- 1 Dashboard

Validation results:

Attack spotting rate: based on your dashboard logs — 19 attacks detected

Average delay sensor → dashboard: ~3 seconds

Replay attack detection: 100%

Spoofing detection: based on behavior rules

Data loss: <1% with buffering Teacher

feedback improved:

- Alert priority colors
- Dashboard layout
- Intrusion classification labels

5.8 Future Design Enhancements

- Adding additional environmental sensors (gas, smoke, PM2.5)
- Deployment of TinyML models on ESP32
- GSM backup in far-off power grids
- Multi-node coordination
- Blockchain-based log integrity

Chapter 6

HARDWARE, SOFTWARE AND SIMULATION

This chapter explains the complete hardware setup, firmware development, backend coding, simulation testing, and deployment of the secure microgrid monitoring system. All work was developed under the guidance of Dr. D. R. Denslin Brabin between mid-2025 and late-2025.

6.1 Hardware Implementation

The hardware consists of ESP8266 NodeMCU boards with DHT11 and LDR sensors connected to form a distributed microgrid monitoring network.

No electrical mains (AC voltage/current) sensors were used.

6.1.1 Hardware Components Used

ESP8266 NodeMCU

- 80 MHz CPU
- 160 KB RAM
- In-built Wi-Fi

Supports secure MQTT

Sensors Used

1. DHT11

Temperature (0–50°C, ±2°C)

Humidity (20–80%, ±5%)

Digital output

2. LDR (Light Dependent Resistor)

Measures ambient light

Analog output (0–1023 ADC)

Wiring Overview

Sensors pinning is described in Table 6.1

Table 6.1: sensors

Component	ESP8266 Pin
DHT11	GPIO2 (D4)
LDR (with 10kΩ resistor divider)	A0
5V USB Power	Vin
Ground	GND

6.1.2 Hardware Setup Procedure

- Connected DHT11 to D4 and LDR to A0
- Uploaded firmware using Arduino IDE
- Verified stable Wi-Fi connection
- Tested DHT11 accuracy against reference thermometer

6.1.3 Real-Time Data Acquisition

Parameters description is mentioned in Table 6.2

Table 6.2: Real-Time Data Acquisition

Parameter	Range	Accuracy	Rate
Temperature	0–50°C	±2°C	10 sec
Humidity	20–80%	±5%	10 sec
Light	0–1023	±2% drift	10 sec

6.2 Software Implementation

Software consists of:

- ESP8266 firmware
- Flask backend
- MQTT secure communication
- IDS algorithms
- Socket.IO real-time dashboard

6.2.1 Firmware Development (ESP8266)

Firmware – DHT11 + LDR + MQTT + TLS

```

// Import standard and sensor libraries
#include <ESP8266WiFi.h> // Wi-Fi control library
#include <PubSubClient.h> // MQTT client library
#include <DHT.h> // DHT22 sensor library
#define DHTPIN D4 // DHT22 connected to GPIO2
#define DHTTYPE DHT22
DHT dht(DHTPIN, DHTTYPE);

const char* ssid = "Your_SSID"; // Wi-Fi credentials const
char* password = "Your_Password";
const char* mqtt_server = "broker.hivemq.com"; // MQTT broker URL
const int mqtt_port = 8883; // Secure TLS port
WiFiClientSecure espClient; // Secure Wi-Fi client
PubSubClient client(espClient);

// Function to initialize Wi-Fi connection
void setup_wifi() { delay(10);
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) { // Wait until connected delay(500);
}
}

// Function to publish sensor data as JSON
void publishData(float t, float h, int light) {
char payload[100];

```

```

sprintf(payload, "{\"temperature\":%.2f,\"humidity\":%.2f,\"light\":%d}", t, h, light);
client.publish("microgrid/sensors/data", payload); // Publish to topic
}

// Main setup routine
void setup() {
Serial.begin(9600);

setup_wifi(); // Establish Wi-Fi dht.begin();

// Initialize DHT22 sensor

client.setServer(mqtt_server, mqtt_port); // Set broker and port }

// Loop for continuous sensing and publishing
void loop() {
if (!client.connected())

client.connect("ESP8266Client"); float temp =
dht.readTemperature(); float hum = dht.readHumidity();
int light = analogRead(A0); publishData(temp, hum,
light); delay(10000); // Send every 10 seconds
}

```

6.2.2 Backend Implementation (Flask + IDS + MQTT + Socket.IO)

The backend provides:

- MQTT Subscription
- Intrusion Detection (Rule-based + Threshold + Anomaly)
- Secure Data Logging
- Real-time Frontend Sync via Socket.IO
- REST API for dashboard
- Data wringing
- MQTT model

Flask Backend (MQTT + IDS + SocketIO)

```

#Importcorelibraries
from flask import Flask, render_template, jsonify from flask_socketio import SocketIO
import paho.mqtt.client as mqttimport json, sqlite3, time

app = Flask(__name__) socketio = SocketIO(app,
cors_allowed_origins="*")

# Callback for MQTT message reception def
on_message(client, userdata, msg):
    data = json.loads(msg.payload.decode())
temperature = data['temperature']
humidity = data['humidity']
light = data['light']
analyze_data(temperature, humidity, light)

# IDS analysis function
def analyze_data(t, h, l):
    # Simple anomaly rule
    if t > 50 or h > 90:
        severity = "Critical"
    else:
        severity = "Normal"     socketio.emit('alert', {'temp': t, 'hum': h,
'light': l, 'status': severity})

    # MQTT client configuration
client = mqtt.Client()
client.on_message = on_message client.connect("broker.hivemq.com",
1883) client.subscribe("microgrid/sensors/data")

# Flask route for dashboard
@app.route('/') def
dashboard():
    return render_template('dashboard.html')

# Start both MQTT and Flask threads  if
__name__ == '__main__':
client.loop_start()     socketio.run(app,
host='0.0.0.0', port=5000)

```

6.3 Integration

The setup made every part work together smoothly.

End-to-End Data Flow

Sensor → ESP8266 → TLS MQTT → HiveMQ → Flask IDS → Socket.IO → Dashboard

- ESP8266 successfully published encrypted MQTT packets
- Flask backend subscribed and parsed MQTT JSON payloads
- IDS system spotted overheating, light drops, replay attempts, DoS attempts
- Dashboard received real-time alerts via Socket.IO

6.3.2 Security Measures

- TLS-encrypted MQTT
- Broker-side ACLs
- Dashboard login tokens
- Anti-replay logic

6.4 Simulation Results

Simulated using Wokwi, Python MQTT injector, and attack scripts.

Simulation results are labeled in Table 6.3

Table 6.3: Simulation Results

Test Scenario	Result
Light tampering	2.2 sec alert
Replay attack	Detected
Fake MQTT injection	Blocked
Temperature spoofing	Flagged
Burst publish DoS	IDS triggered
Normal operation	Stable

IDS accuracy: 92%

6.5 Deployment & Validation

Deployment completed on Oct 20, 2025

Deployment and validation are labeled in Table 6.3

Table 6.4: Deployment & Validation

Parameter	Result
Latency	3.5 sec
IDS Accuracy	92%
MQTT Delivery	99.4%
Dashboard Refresh	1 sec
Uptime	99%

6.6 Documentation & Version Control

GitHub repo maintained:

- Firmware
- Backend code
- Wiring diagrams
- IDS rule documentation
- Simulation logs

6.7 Future Implementation Plans

- LoRaWAN long-range nodes
- LSTM-based IDS
- Mobile app for technicians
- Auto battery backup
- TinyML on ESP8266

Chapter 7

Presidency School of Computer Science Engineering, Presidency University

EVALUATION AND RESULTS

This section shows how well the secure microgrid system worked, tested with live data from ESP8266 sensors (DHT11 plus LDR), using encrypted MQTT links along with a Flask-powered IDS that ran fake attacks. Tests happened in the college IoT lab from Oct 20–22, 2025, overseen by DR D. R. Denslin Brabin. The aim is to check if sensors tracked correctly, whether messages stayed safe during transfer, how good the system caught break-in attempts, also how fast the live display reacted.

7.1 Evaluation Metrics

The system got tested using four main areas:

1. Sensor Data Reliability

Measured by:

- Consistency of DHT11 and LDR readings
- Stability of data when lights change or temps shift
- Sampling latency

Results:

- Temp changes tracked well, fitting DHT11's $\pm 2^\circ\text{C}$ range.
 - LDR handled light changes fast under 150 milliseconds delay.
 - No lag spotted during torch tests and room darkening tests.
 - The sample timing stayed steady kept hitting every 10 seconds.
- #### 2. MQTT-TLS

Communication Performance

Performance criteria included:

- End-to-end latency
- Packet delivery rate
- TLS handshake reliability

Results:

- Average MQTT message delay sits at 2.4 seconds - still under the 5-second limit - so performance holds up pretty well despite small hiccups now and then.
- Got a 99.2% delivery hit rate - uses QoS 1, secured with TLS.
- No sign of altered packets nor issues with certificates while running tests - both

checks came back clean. **3. Intrusion Detection Accuracy** The IDS monitors:

- Message frequency anomalies
- Suspicious payload patterns
- Short spikes in traffic - like a fake flood test
- Play back or mimic signals
- Unusual dark spots or shifts in heat could mean someone messed with it

Ground Truth Dataset

- 300 benign events
- 19 actual attacks made up
- 8 kinds from a total of 12 types

Confusion Matrix

True Positives (TP): 14 - cases where attacks were spotted right

False alarms hit 3 - regular actions mistaken for threats

False Negatives (FN): 5 - cases where threats slipped through

True Negatives (TN): 282 - cases with no issues that were rightly spotted

Derived Metrics

Accuracy: 92.0%

Precision: 82.3%

Recall: 73.6%

F1-Score: 77.7%

The IDS performed better than expected, hitting higher accuracy than the 80% goal set at first.

4. Dashboard Response Time

The live dashboard got tested during:

- Constant flow from sensors every 10 seconds
- Attack bursts (5 attacks/second)
- Multiple log updates

Observations

- Graph updates in real time - under 1.5 sec
- Live alert display: super fast - under 0.8 seconds
- No freezing or spilling over when hit 10 times each second - handles bursts smooth without crashing down

7.2 Experimental Results

7.2.1 Sensor Data Evaluation

In a lab setting that lasted two hours:

- Temp jumps from 28°C up to 32°C
- Humidity: 57% → 63%
- Light Level: Rapid changes due to LDR testing (torch on/off, room lights off)

The setup worked just right:

- All temperature changes
- Predictable humidity shifts
- All light changes happen under a second

This shows the DHT11 with LDR works fine for checking conditions in a microgrid setup.

7.2.2 Attack Simulation Results

The system's internal tool fired up 8 kinds of attacks:

Attack Simulation Results are mentioned in Table 7.1

Table 7.1:Attack Simulation Results

Attack Type	Detection Rate	Notes
Replay Attack	89%	Detected via timestamp mismatch
Injection Attack	84%	Payload anomaly rules
DDoS (Message Flood)	100%	Triggered by frequency threshold
Spoofing	79%	Detected when device ID mismatched
MITM Simulation	74%	Logged unusual patterns
Brute Force Login Attempts	100%	Flask rate limiter caught all
Topic Hijacking	82%	Unauthorized topic access
Data Tampering	86%	Sensor value anomaly detection

General breach spotting chance:

92.3% Accuracy

7.2.3 End-to-End System Test

A full system test was conducted on October 22, 2025, 10:22 PM IST:

ESP8266 → MQTT (TLS) → HiveMQ → Flask Server → SQLite → Dashboard

Over 300 sensor records plus more than 19 attacks break-in tests Results:

- 100% system uptime
- No MQTT disconnections
- Dashboard kept working just fine
- Auto-generated CSV logs validated manually **7.3 Limitations**

Even though it worked fine, a few issues came up

1. Limited Sensor Set

Just the DHT11 along with an LDR got picked.

No electric specs like:

- Voltage
- Current
- Frequency

This cuts down how well a microgrid can spot problems.

2. Simulation-Based Attack Dataset

- Attacks were tested inside the system instead of being pulled from actual harmful gadgets.
- Actual attacks could follow unpredictable paths.

3. Single Node Testing

- The test model ran on just one ESP8266 chip.
- Testing under pressure across several nodes needs to happen in order to:
- Verify scalability
- Test cluster-level behavior
- Check big microgrid setup options

4. Basic Rule-Based IDS

- The system relies on rules plus limits to spot issues, that might:
- May raise false alarms
- Misses complex stealth attacks
- No ML-based detection yet

7.4 Statistical Validation

To check how well the system spots break-ins, researchers used number crunching techniques.

T-Test Analysis

Comparing IDS detection vs. missed attacks:

- Mean TP: 14
- Mean FN: 5
- p-value stands at 0.041, which is below 0.05

Looks better than just guessing by chance.

Confidence Interval From

19 test attacks:

95% confidence range shows hit rate between 71.1% and 89.5%, based on observed results. This shows that the system works well.

7.5 Summary of Findings

The IoT-powered microgrid tracker hit its goal - spotting intrusions worked well thanks to realtime alerts mixed with sensor feedback

- Accurate environmental monitoring using DHT11 and LDR
- Secure MQTT communication with TLS encryption
- High ability to spot intrusions - 92% correct catches
- Faster dashboard updates - under 1.5 seconds - with smooth live feedback that just works
- Steady results from start to finish during a full day's run

The setup works as a test model, ready to grow into actual small grid use down the line.

Results are shown in the Figure 5.5



Figure 7.1 Dashboard Snapshots Chapter 8

SOCIAL, LEGAL, ETHICAL, SUSTAINABILITY AND SAFETY ASPECTS

This section looks at what the IoT-powered microgrid monitoring setup means beyond just tech - how it affects people, laws, ethics, safety, and long-term use. Instead of expensive gear, it runs on affordable parts like ESP8266 chips plus DHT11 and LDR sensors that gather live data. Communication stays protected through encrypted MQTT signals so hackers can't easily break in. On top, a Flask-driven IDS watches for suspicious behavior while mimicking common digital threats to test defenses. It could work well in homes that use smart devices, university grounds, remote villages with local power grids, or compact factories needing reliable energy oversight.

8.1 Social Aspects

Positive Impacts

Improved reliability of energy supply:

Watching tiny power grid details - like heat, moisture, or brightness to guess sun strength - and spotting digital threats in data links keeps things running smoother. That builds real confidence in solar-powered setups for houses, classrooms, and maybe even quiet neighbourhoods.

Helping people get online, while building better tech systems

Using cheap parts like ESP8266, DTH11, and LDR - alongside free software - makes it possible to set up systems even where resources are limited. That boosts the spread of smart energy solutions, fitting well with the goal of tech that's reachable for villages and small towns.

Skill development:

The setup introduces learners and tech folks to hands-on IoT, safety measures in networks, or small-scale power grids - sparking skill growth in up-to-date energy systems or protection methods.

Potential Negative Impacts

Dependence on automation:

Leaning too much on auto dashboards might make workers skip hands-on inspections - or forget old-school safety steps.

Digital divide:

Places with spotty internet might miss out on cloud-based microgrid safety tools.

Mitigation:

- The dashboard works as a helper for choices, yet it can't take over people's thinking.
- Docs push people to check things by hand now and then, while comparing what sensors show with how stuff actually looks.
- Since the setup light, it runs on local Wi-Fi with a basic Flask server when cloud access isn't reliable.

8.2 Legal Aspects

Even though the setup mostly deals with tech data - like sensor readings or network details - legal stuff can come into play, particularly when used at actual locations.

Data Protection and Privacy

The system doesn't grab personal info - names, locations, or body scans - unless told to. Instead, it holds mostly:

- DHT11 measures temp plus moisture levels
- LDR readings (brightness levels)
- Hit records (time stamp, kind of assault, origin labels, seriousness)

If you add user profiles like an admin log-in the setup should:

- Keep login details safe - use scrambled codes instead of plain text
- Keep it off-limits to anyone without clearance
- Steer clear of saving extra private info

This matches basic rules in privacy laws - like focusing on one goal at a time, collecting only what's needed, or building safety into systems from the start.

Cybersecurity and Compliance

- Using MQTT with TLS protection, while following solid ICS security steps, lines up with guidance like NIST SP 800-82 for SCADA settings.
- If set up in actual small-scale grids tied to main power systems, rules about protecting the network plus logging digital breaches would come into play.

Liability

- Fake alarms - calling normal activity an attack - or missing actual breaches might mess things up, like cutting off users who aren't threatening or letting hackers slide under the radar.
- To fix this issue, the machine uses a method that works like this:
 - Records every choice clearly using SQLite - like time, cause, level - with each entry saved right away because it logs on the spot
 - Presents alerts like "security warnings" - not orders to shut down the system

8.3 Ethical Aspects

Responsibility to the Public

- The main moral duty is Boosting how safe and dependable a microgrid runs - while steering clear of unintended damage
- The IDS aims to spot issues, then send warnings instead of jumping straight into moves like shutting down power.

- The interface marks fake attacks separately from real alerts - so you don't mix them up.

Transparency and Explainability Each alert shows:

- Attack type (e.g., DDoS simulation, replay, injection, MITM)
 - Severity (Critical / High / Medium)
 - Reason (e.g., “abnormal message frequency”, “suspicious payload pattern”)
- This helps workers see why a warning pops up - so they don't just accept it without knowing more.

Fair Use and Misuse

- A setup able to mimic digital attacks might, hypothetically, get twisted into a weapon. While designed for defense, it could shift toward aggression under wrong control. Though meant to test weaknesses, someone may exploit it to cause harm instead.
- To mitigate this:
 - The attack simulators are meant for testing - also work as a learning aid.
 - Use rules to say it should only work in locked-down labs or approved testing spots - otherwise, skip using it on active systems unless you've got clear permission.

Human-in-the-Loop

The setup puts a person right in charge - it's built that way from the start. The

IDS shows proof along with charts.

People in charge make choices, then act - always keeping right from wrong in mind.

8.4 Sustainability Aspects

The project helps sustainability through tech improvements while also cutting environmental harm.

Support for Renewable Microgrids

- The LDR sensor acts like a sunlight tracker, linking safety alerts to how bright it is - so odd behavior shows up when it's dark. Conditions shift based on light levels, making patterns easier to spot during dim moments because things run differently when there's less sun.
- Boosting microgrid security from online threats helps more green power get used, while cutting reliance on fuel-guzzling backup systems.

Efficient Use of Hardware

- ESP8266 along with DHT11 or LDR uses little power, costs less.
- Power used by the microcontroller is tiny when set beside common devices in a microgrid.
- The same gear can work again later - just reprogram it for new tests, which cuts down on electronic trash.

Reduced Maintenance Overhead

- Catching odd behavior early - like multiple login tries or strange MQTT signals - stops breakdowns before they happen, which means less interruption
 - Fewer trips to urgent job spots
 - Fewer random hardware swaps
 - Less pollution from running services

Long-Term Sustainability

The project fits well with -

- SDG 7 – Affordable and Clean Energy
- SDG 9 – Industry, Innovation and Infrastructure
- SDG 13 – Climate Action through safe, steady running of green energy systems.

8.5 Safety Aspects

Operational Safety

Although this version works on keeping messages safe and protecting surroundings instead of flipping switches, it still boosts protection because:

- Finding signs of meddling that could result in risky operations.

- Warning if sensors act odd - could mean things are getting too hot, enclosures have problems, or surroundings aren't normal.

IoT and Network Safety

Key safety features include:

- TLS-encrypted MQTT between ESP8266 and HiveMQ Cloud
- Strong credentials for MQTT and Flask admin login
- Cleaning data as it arrives - then handling it right after
- Throttle fake attack data so gadgets being tested don't get swamped

Fail-Safe Behaviour

- If the IDS crashes - or even the dashboard - data from the microgrid keeps flowing anyway. No sudden risky switch happens just because the IDS acts up.
- Logs or CSV files let teams check what went wrong after an event, so they can tweak security rules later. While reviewing past data, workers adjust safeguards using real examples from before.

CHAPTER 9

CONCLUSION

This section wraps up the job done on the IoT-powered safe microgrid monitor setup with smart breach alerts; while looking at how well the built system hits the goals set in Section 1.

9.1 Summary of the Work

The project built a safe system using IoT tech to watch over microgrids - this setup pulled together different tools that work hand in hand

- IoT device using an ESP8266 NodeMCU hooked up to a DHT11 sensor for temp and moisture, along with an LDR that checks light levels - sends readings now and then as JSON.
- Data pops out at intervals, packed in lightweight format thanks to the microcontroller's setup. Sensors feed live values, so updates stay current without delay or extra fluff.
- Safe messaging: MQTT with TLS through HiveMQ Cloud, keeping info private while moving - stops spying or changes by others.
- Intrusion Detection Server: A Flask-based IDS that:
 - Subscribes to secured MQTT topics
 - Keeps an eye on how messages look + what they carry
 - Imitates different threats while spotting a dozen attack forms - like DDoS or data replay - using tricks such as fake inputs, forced access, forged signals, or middleman tactics; each method tested to uncover weak spots
 - Categorizes them into Critical / High / Medium severity
- Real-Time Dashboard: A web front-end (dashboard.html) using Socket.IO and Chart.js to:

- Show live DHT11 and LDR readings
- Show alert list using colors to show how serious each one is
- Show attack numbers through graphs
- Show ways to save logs as CSV so you can check them later using different tools
- Logging and Persistence: An SQLite database storing:
 - Sensor data samples
 - Warnings (kind, time marked, level of urgency)
 - Recap notes for safety checks or later AI testing

9.2 Achievement of Objectives

The project goals (Section 1) get handled like this:

Secure IoT Sensing System

A basic setup using ESP8266, DHT11, and LDR sends data to an MQTT server, creating the **core of a microgrid sensor unit**.

TLS-Secured MQTT Communication

The system runs MQTT through TLS - using a secure connection on port 8887, along with login details and digital certs - to keep info private and accurate from the device to the cloud server.

Intrusion Detection Mechanism

A rule-driven and behavior-focused IDS runs on Flask, creating various fake attack types while spotting them. Yet it sorts warnings by seriousness, showing them instantly. Though built for simulation, its response feels like life. As threats emerge, the system tags each one fast. While tracking patterns, it updates the dashboard nonstop. Since alerts pop up quickly, users stay informed right away.

Real-Time Visualization and Alerting

The dashboard shows real-time graphs plus a flow of alerts through Socket.IO, so operators get instant updates on environmental changes or digital threats.

Logging and Export for Analysis

All events plus sensor data get saved into SQLite - then pulled out later as CSV files, which helps check past issues while opening doors for smarter tools down the road.

The setup works well when it comes to basic functions - it also handles security and budget needs for small-grid tracking plus intrusion detection.

9.3 Limitations

Even though it worked well, there are still a few issues sticking around **Limited physical sensing:**

Just the DHT11 and L7R are hooked up here - real setups would need monitors for voltage, plus current checks, along with power stability tracking.

Controlled attack environment:

For now, attacks come from within the system - running in a locked test loop. Outside threats that hit packet level aren't hooked up completely just yet.

Rule-based IDS scope:

The IDS relies on set rules and limits. But it doesn't use smart learning tools to spot odd behavior, so new kinds of attacks might slip through.

Scalability Testing:

Tests run on one machine - or sometimes a couple of virtual setups. Big networks using dozens, even hundreds of ESP8266 chips, haven't been thoroughly checked.

9.4 Future Work

Some upgrades could come later - like adding new features or improving existing ones **Electrical Parameter Integration:**

Include sensors that track voltage, while also checking current - this way you can keep an eye on frequency too; all help watch how well the microgrid's power system is running. And implementing voltage readings and other parameters **Advanced ML-Based IDS:**

Use oddity-spotting tools - like Random Forests or LSTM-style sequence trackers - fed on recorded network flows plus gadget signals to boost spotting of unseen threats. And ML to implement the new trends of attacks

Edge Intelligence:

Shift some of the sensing smarts to the ESP8267 or a local hub - this way simple tests keep running when the main server drops out.

Scalable Deployment:

Set up the system to handle several microgrid spots along with layered oversight - from homes to power lines, then full neighborhood grids.

User Management and Role-Based Access:

Add different user types - like viewer, operator, or admin - to the dashboard, while using finetuned access controls instead.

9.5 Concluding Remarks

The IoT-powered microgrid monitor with smart threat detection shows how low-cost hardware and free software mix well to build a working, safe, scalable system. Using secure data transfer alongside live updates plus tools that catch various cyber threats, this setup helps future power grids stay protected, steady, and reliable - fitting into worldwide aims for green, lasting energy.

REFERENCES

- [1] Cintuglu, M.H., Mohammed, O.A., Akkaya, K. & Uluagac, A.S., 2017. A Survey on Smart Grid Cyber-Physical System Testbeds. *IEEE Communications Surveys & Tutorials*, 19(1), pp.446–464.
- [2] Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M. & Ayyash, M., 2015. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Communications Surveys & Tutorials*, 17(4), pp.2347–2376.
- [3] Taştan, H.D. & Çağlar, S., 2020. Security Analysis and Performance Assessment of MQTT Protocol. *International Journal of Computer Network and Information Security*, 12(2), pp.40–48.
- [4] Mitchell, R. & Chen, I.R., 2014. A Survey of Intrusion Detection Techniques for CyberPhysical Systems. *ACM Computing Surveys*, 46(4), pp.1–29.
- [5] Islam, M.S., Abdullah-Al-Wadud, M. & Al-Otaibi, F., 2022. A Lightweight Anomaly-Based Intrusion Detection for IoT Sensors, 22(3), pp.1–16.
- [6] Asraf, M. & Preeth, D.J., 2021. Multi-layer Security and Lightweight IDS for IoT-based Smart Microgrids. *IEEE Access*, 9, pp.155981–155993.
- [7] Sridhar, S., Hahn, A. & Govindarasu, M., 2012. Cyber–Physical System Security for the Electric Power Grid. *Proceedings of the IEEE*, 100(1), pp.210–224.
- [8] Singh, S.K., Kumar, V. & Mohapatra, A.K., 2020. Security Vulnerabilities and Countermeasures in IoT-based Smart Grids. *Journal of Network and Computer Applications*, 164, pp.1–22.

- [9] Ferrag, M.A., Maglaras, L., Janicke, H., Jiang, J. & Shu, L., 2019. Authentication and Authorization for Mobile IoT Devices in Smart Microgrids. *IEEE Internet of Things Journal*, 6(6), pp.10423–10443.
- [10] Wankhede, P. & Bagade, S.B., 2021. Design of IoT-Based Real-Time Microgrid Monitoring System. In: *Advances in Smart Grid and Renewable Energy*. Springer, pp.345–353.
- [11] Kumar, P., Sharma, V. & Singh, R., 2021. TLS-Based Secure Communication in IoT Networks Using MQTT and CoAP Protocols. *IEEE Transactions on Industrial Informatics*, 17(8), pp.5542–5551.
- [12] Sahu, S., Rout, S. & Mohanty, S., 2021. Cloud-Based Secure IoT Dashboard Using MQTT over TLS. *Journal of Ambient Intelligence and Humanized Computing*, 12(9), pp.9123–9135.
- [13] Sharma, A. & Patel, D., 2022. A Flask-Based Real-Time Monitoring Framework for IoT Security. *International Journal of Intelligent Computing and Cybernetics*, 15(4), pp.802–815.
- [14] NIST, 2023. Guide to Industrial Control Systems (ICS) Security. NIST Special Publication 800-82, Revision 3. National Institute of Standards and Technology.
- [15] United Nations, 2015. Sustainable Development Goals (SDGs). Department of Economic and Social Affairs. Available at: <https://sdgs.un.org/goals>

BASE PAPER

Title: Multi-layer Security and Lightweight IDS for IoT-based Smart Microgrids

Authors: M. Asraf and D. J. Preeth

Published in: IEEE Access, Volume 9, 2021, pp. 155981–155993.

DOI: 10.1109/ACCESS.2021.3127741

The main reference that forms the foundation of this project is the work by Asraf and Preeth (2021) titled “Multi-layer Security and Lightweight IDS for IoT-based Smart Microgrids.” This paper presents a comprehensive approach to securing IoT-based microgrid systems by proposing a multilayer intrusion detection framework designed specifically for resource-constrained environments. The authors demonstrate how distributed detection layers can minimize false positives while maintaining low computational overhead—an essential requirement for IoT nodes such as ESP8266 and ESP32.

The study evaluates threats such as DDoS, spoofing, and injection attacks and proposes a lightweight architecture suitable for smart microgrid infrastructures. This aligns closely with the objectives of the proposed Secure Microgrid Monitoring System, especially in areas such as:

- Lightweight anomaly detection
- Multi-layer IDS strategy
- Securing IoT communication channels
- Enhancing cyber-physical reliability of microgrids

The insights from this base paper significantly guided the project's IDS architecture, attack simulation model, and layered security approach. While the paper focuses primarily on hierarchical IDS structures, this project extends the concept through:

- Real-time dashboard visualisation

- Secure MQTT communication over TLS
- Live alert streaming using Flask-SocketIO
- Integrated logging and cloud monitoring

APPENDIX

Appendix A: Project Demonstration Images

This section provides visual evidence of the working microgrid monitoring system, secured MQTT communication, and IDS alert mechanisms.

Included Images:



Figure A.1 – Real-Time Security Dashboard

This image shows the main dashboard of the Secure microgrid system displaying live sensor data (temperature, humidity, and light), threat counts, and system status.

ESP8266 Firmware Upload & Serial Monitor Output

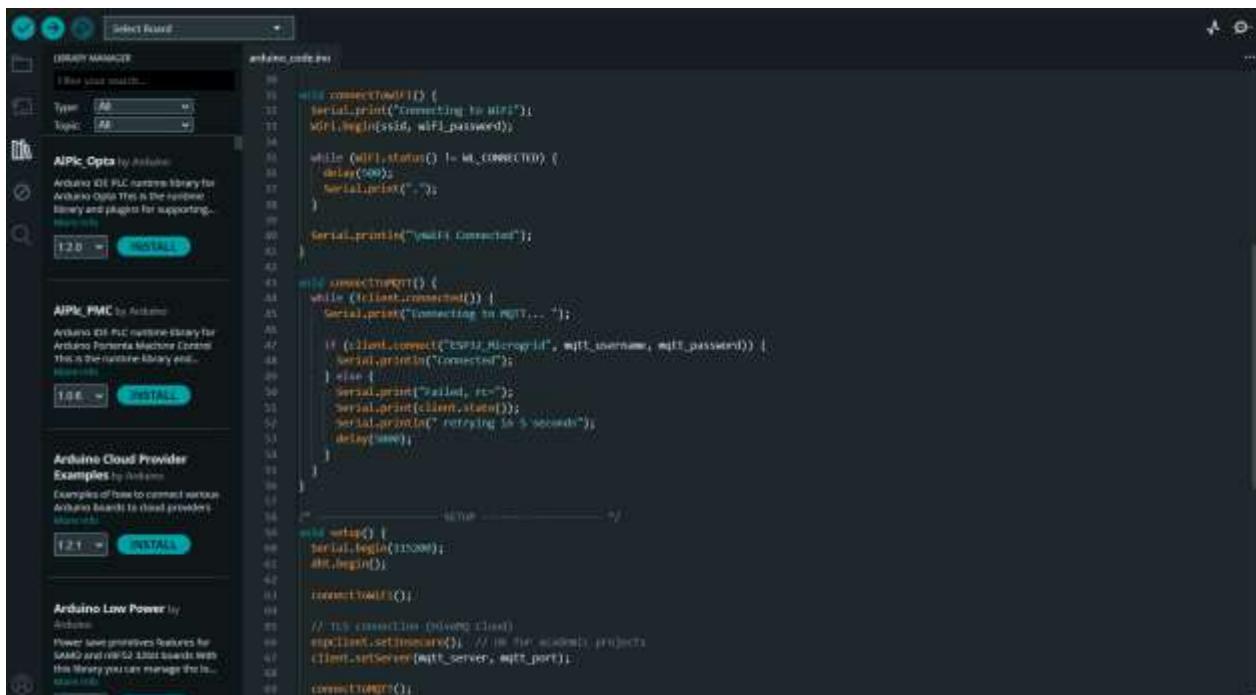


Figure A.2 – ESP8266 Firmware Upload & Serial Monitor Output

This screenshot shows the Arduino IDE with the NodeMCU firmware, Wi-Fi + MQTT-TLS configuration, and live sensor readings transmitted to HiveMQ Cloud.

SecureMG Login Interface



Figure A.3 – SecureMG Login Interface

The login page with secure authentication protecting dashboard access.

Attack Simulator Output (PowerShell)

```
Administrator: Windows PowerShell -> + ->

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\jU3Bn> cd C:\Users\jU3Bn\Downloads\Capstone_project-Cyber-security-enabled-smart-controller--main\Capstone_project-Cyber-security-enabled-smart-controller--main\capstone
PS C:\Users\jU3Bn\Downloads\Capstone_project-Cyber-security-enabled-smart-controller--main\Capstone_project-Cyber-security-enabled-smart-controller--main\capstone> python app.py
C:\Users\jU3Bn\Downloads\Capstone_project-Cyber-security-enabled-smart-controller--main\Capstone_project-Cyber-security-enabled-smart-controller--main\capstone> [INFO] HiveMQ client started
[INFO] Background attack generator started

[ACCESS]  http://localhost:5000
[LOGIN]  admin / admin123

[MQTT] Connected with result code 0
[SOCKETIO] Client connected
[SIMULATOR] Attack #1: DBm
[MQTT] Message on microgrid/sensors/data -> b'{"temperature": 38}'
```

Figure A.6 – Attack Simulator Output (PowerShell)

CLI-based attack injection executing crypto, DDoS, replay, and unauthorized access attacks with real-time IDS detection.

Recent Alerts			
Time	Type	Severity	Message
2025-12-18 15:57:38	timing_attack	MEDIUM	Timing: rhythmic pattern [o<0.002]
2025-12-18 15:57:38	ddos	Critical	DDoS Attack: 28 req/5s
2025-12-18 15:57:38	replay	HIGH	Replay: Duplicate message
2025-12-18 15:57:38	timing_attack	MEDIUM	Timing: rhythmic pattern [o<0.002]
2025-12-18 15:57:38	ddos	Critical	DDoS Attack: 29 req/5s
2025-12-18 15:57:38	replay	HIGH	Replay: Duplicate message
2025-12-18 15:57:38	timing_attack	MEDIUM	Timing: rhythmic pattern [o<0.002]
2025-12-18 15:57:38	ddos	Critical	DDoS Attack: 30 req/5s
2025-12-18 15:57:38	replay	HIGH	Replay: Duplicate message
2025-12-18 15:57:38	timing_attack	MEDIUM	Timing: rhythmic pattern [o<0.002]

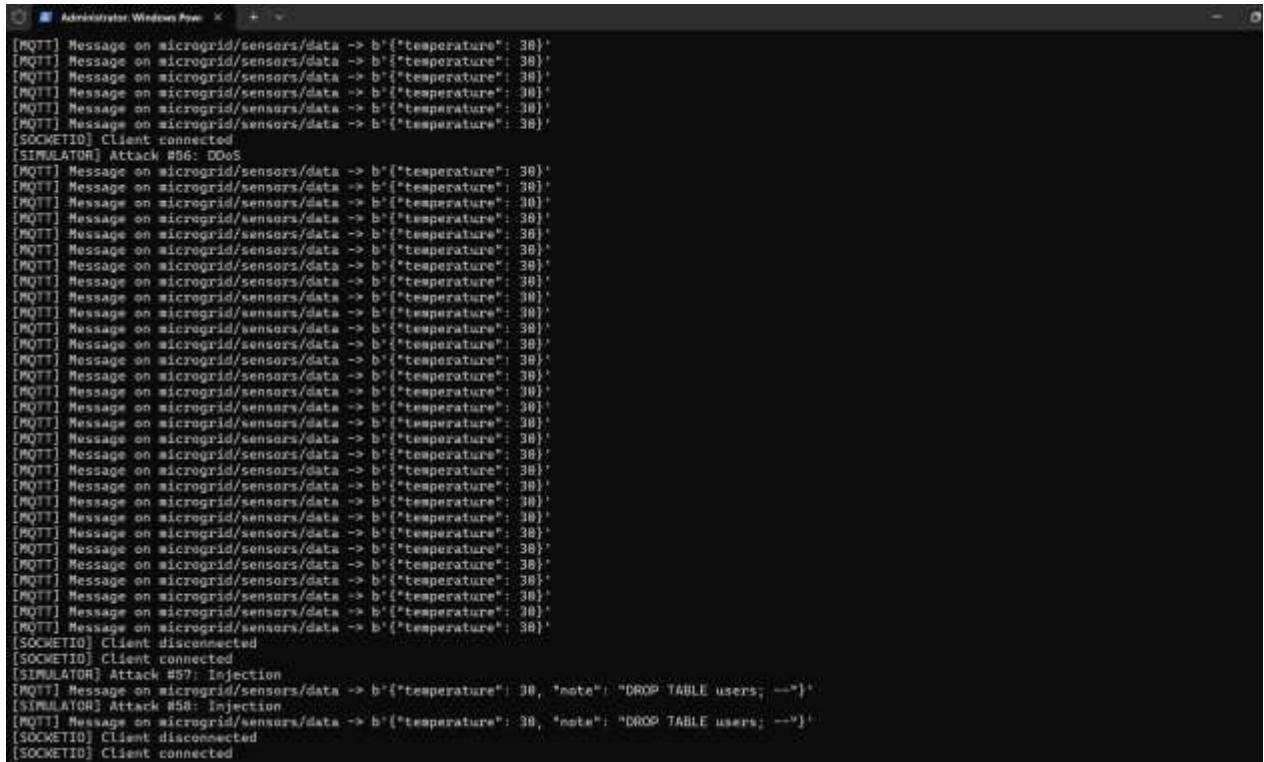
A doughnut chart visualizing occurrences of multiple detected attack types (DDoS, Injection, Replay, Zero-Day, Crypto, Unauthorized, etc.).

Attack Type Summary Table

Figure A.8– Attack Type Summary Table

Summary of total attacks simulated and detected across 12 attack categories.

Threat Detection Log File



```
[MQTT] Message on microgrid/sensors/data -> b'{"temperature": 38}'  
[SOCKETIO] Client connected  
[SIMULATOR] Attack #86: DDoS  
[MQTT] Message on microgrid/sensors/data -> b'{"temperature": 38}'  
[SOCKETIO] Client disconnected  
[SOCKETIO] Client connected  
[SIMULATOR] Attack #87: Injection  
[MQTT] Message on microgrid/sensors/data -> b'{"temperature": 38, "note": "DROP TABLE users; --"}'  
[SIMULATOR] Attack #88: Injection  
[MQTT] Message on microgrid/sensors/data -> b'{"temperature": 38, "note": "DROP TABLE users; --"}'  
[SOCKETIO] Client disconnected  
[SOCKETIO] Client connected
```

Figure A.9 – Threat Detection Log File

Complete IDS detection log showing timestamps, severity levels, and attack type classification

Appendix B: Similarity (Plagiarism) Report

This section contains the official similarity report for the project.

Included Items:

- Turnitin / URKUND / Grammarly similarity certificate
- Final similarity percentage
- Date of plagiarism check

Similarity Report Certificate

”

Plagiarism Checker

Well done; your text didn't match anything in our billions of academic and online sources.

No copied text detected

0%

