# IoT Network Intrusion Detection System

Chirayu Badgujar & Mukund Gohil

---

## Contents

# 1. Introduction

## 1.1. Overview

The Advanced IoT Network IDS (Intrusion Detection System) is a comprehensive platform designed to detect, analyze, and visualize network security threats in static and real-time. It combines multiple detection methodologies with an intuitive interface to provide both educational insights and practical security monitoring.

This system captures network traffic patterns and applies various detection algorithms to identify potential security threats, making it suitable for education, research, and security awareness training.

## 1.2. Purpose and Goals

The system serves several key purposes:

- **Educational Tool:** Demonstrates how different IDS detection methods work.
- **Research Platform:** Allows experimentation with different detection algorithms.
- **Security Awareness:** Visualizes attack patterns and normal network behavior.
- **Testing Environment:** Provides a safe environment to simulate cyber attacks and their detection.

Primary goals include:

- Real-time visualization of network traffic patterns.
- Implementation of multiple detection methodologies.
- Detailed logging and analysis of network events.
- Customizable simulation parameters.
- Support for both simulated and real network traffic.

## 1.3. System Architecture

The system follows a modular architecture with four primary components:

1. **Data Acquisition Layer:** Gathers network packets through simulation or real capture.
2. **Analysis Layer:** Contains detection models that identify potential threats.
3. **Visualization Layer:** Presents the data through an intuitive graphical interface.
4. **Notification Layer:** Manages alerts and responses to detected threats.

The system pipeline is as follows:

`[Data Acquisition]` → `[Feature Extraction]` → `[Detection Models]` → `[Decision Engine]` → `[Results Aggregation]` → `[Alert Generation]` → `[Notification System]` → `[User Interface]` → `[Visualization]` → `[Logs]` → `[Statistics]`

# 2. Core Components

## 2.1. PacketCapture

The `PacketCapture` class handles all aspects of network data acquisition, whether simulated or real.

**Key Features:**

- Support for both simulated and real packet capture.
- Feature extraction from raw packets.
- Realistic pattern generation for various protocols.
- Configurable attack simulation.
- Buffer management for packet history.

**Methods:**

- `get_next_packet()`: Retrieves the next packet.
- `get_feature_vector()`: Extracts numerical features.
- `_generate_sample_packet()`: Creates a simulated packet.
- `_extract_features()`: Extracts features from real packet.

**Usage Example:**

```
packet_capture = PacketCapture(use_sample_data=True)
packet = packet_capture.get_next_packet()
feature_vector = packet_capture.get_feature_vector(packet)
```

## 2.2. IDSModels

The `IDSModels` class encapsulates all detection algorithms and parameters.

**Key Features:**

- Signature-based, anomaly-based, and placeholder deep learning detection.
- Voting mechanism for ensemble decisions.

**Methods:**

- `predict(feature_vector)`: Runs detection models and returns results.
- `_initialize_models()`: Sets up initial training.

**Usage Example:**

```
ids_models = IDSModels()
predictions = ids_models.predict(feature_vector)
signature_result = predictions['signature']
anomaly_result = predictions['anomaly']
final_decision = predictions['final']
```

### 2.3. SimulationWindow

Main UI component managing the application.

**Key Features:**

- Tabbed interface for visualization, logs, stats, settings.
- Real-time visualization with PyQtGraph.
- Control panel for simulation parameters.

**Methods:**

- `update_simulation()`, `_update_visualizations()`
- `export_logs()`, `import_training_data()`

### 2.4. NotificationSystem

Handles alert generation and notification delivery.

**Key Features:**

- Threshold-based alerts.
- Visual notifications and logs.
- Extensible to email or SIEM.

**Methods:**

- `process_event(event)`: Triggers alert if needed.
- `_send_notification()`: Delivers alert.

## 3. Detection Methods

### 3.1. Signature-based Detection

This method identifies known attack patterns by comparing traffic against a signature database.

**Implementation:**

- Uses RandomForestClassifier from scikit-learn.
- Features: protocol, port, packet size.
- Trained on labeled normal and attack traffic.
- Returns binary classification.

**Advantages:**

- High precision.
- Low false positives.

- Fast.

**Limitations:**

- Cannot detect novel/zero-day attacks.
- Needs signature updates.
- Vulnerable to evasion.

## 3.2. Anomaly-based Detection

Identifies deviations from normal behavior using unsupervised learning.

**Implementation:**

- Uses IsolationForest.
- Trained on normal traffic.
- Converts anomaly scores to binary decisions.

**Advantages:**

- Detects novel/zero-day attacks.
- No signature database needed.
- Adaptive.

**Limitations:**

- Higher false positives.
- Needs accurate baseline.
- Depends on training data.

## 3.3. Deep Learning Detection

Placeholder for advanced sequential pattern models.

**Planned Implementation:**

- CNN or RNN for sequential analysis.
- Feature extraction from packet sequences.
- Returns probability of attack.

**Current Simulation:**

- Simulated probability influenced by other models.

### 3.4. Ensemble Decision Making

Combines all detection results using voting.

**Implementation:**

- Simple majority voting.
- Equal weight per model.
- Extensible to weighted voting.

**Advantages:**

- Reduces false positives.
- More robust.

## 4. User Interface Guide

### 4.1. Dashboard Overview

Displays system status and key metrics.

- Status Indicator (Active/Stopped)
- Packet Counter
- Alert Counter
- Last Alert Info

### 4.2. Visualization Tab

Graphical network traffic view.

- Scatter plot: port vs packet size
- Color-coded: Green (Normal), Red (Attack)
- Interactive: click for details

### 4.3. Event Logs Tab

Detailed event log view.

- Text log: Chronological
- Event table: ID, IPs, Port, Protocol, Decision
- Color-coded rows

### 4.4. Statistics Tab

Displays traffic and model performance stats.

- Time-series of attacks
- Attack type distribution (Bar chart)
- Model performance (TP/FP rates)

### 4.5. Settings Tab

System and model configuration.

- Select network interface (Simulated/Real)
- Enable/Disable models
- Adjust sensitivity
- Import/export training data/logs

### 4.6. Control Panel

Controls for simulation.

- Start/Stop buttons
- Speed slider (100ms–2000ms)
- Clear button (reset data)

## 5. Data Flow

### 5.1. Packet Acquisition

**Simulated Mode:**

- Generates realistic packets with normal/attack traits.
- Based on predefined patterns.

**Real Network Mode:**

- Uses Scapy for live packet capture.
- Extracts features from raw packets.

### 5.2. Feature Extraction

Converts raw packets into numerical vectors.

- Encoded protocol, port, size, etc.
- Converts categorical to numerical features.

### 5.3. Analysis Pipeline

- Feature vector passed to detection models.
- Models return individual decisions.
- Ensemble mechanism combines decisions.
- Final decision stored with event data.

### 5.4. Visualization Process

- Events added to history.
- Visual data extracted (port, size, status).
- Plots updated and styled.
- Statistics recalculated.

## 6. Model Training

### 6.1. Initial Training

- Uses pre-generated sample data.
- Signature model trained on full dataset.
- Anomaly model trained on normal data.
- Models ready on startup.

### 6.2. Importing Custom Training Data

- CSV file format: Protocol, Port, Size, Attack (0/1).
- User imports via Settings tab.
- Models retrained and updated.

### 6.3. Model Evaluation

- Tracks true/false positives.
- Accuracy based on simulation ground truth.
- Real mode requires feedback for accuracy.

# 7. Alert System

## 7.1. Alert Thresholds

- Configurable per attack type.
- Default: Port Scan (3), DDoS (1), Lateral Movement (2), Amplification (2), Unknown (5)

**Logic:**

- Count matching events.
- Trigger alert if threshold met.
- Reset counter after alert.

## 7.2. Notification Types

- Visual indicators, dialogs.
- Event logs with highlights.
- Extensible: Email, SIEM, Response triggers.

## 7.3. Alert Responses

- Log event details.
- Trigger external scripts.
- Interact with firewalls.
- Playbook automation.

# 8. Alert System

## 8.1. Alert Thresholds

- Configurable per attack type.
- Default: Port Scan (3), DDoS (1), Lateral Movement (2), Amplification (2), Unknown (5)

**Logic:**

- Count matching events.
- Trigger alert if threshold met.
- Reset counter after alert.

### 8.2. Notification Types

- Visual indicators, dialogs.
- Event logs with highlights.
- Extensible: Email, SIEM, Response triggers.

### 8.3. Alert Responses

- Log event details.
- Trigger external scripts.
- Interact with firewalls.
- Playbook automation.

## 9. Data Management

### 9.1. Exporting Logs

- Export via Settings tab.
- CSV format.
- Includes metadata, predictions, timestamps.

### 9.2. Importing Data

- Import via Settings tab.
- CSV format.
- Validates format, retrains models.

### 9.3. Data Retention

- Events stored in memory.
- Visualization: last 100 events.
- Clear Data resets memory.

**Extensions:**

- Add database for persistence.
- Implement rotation/compression.

# 10. Attack Types

## 10.1. Port Scanning

**Characteristics:**

- Multiple small packets to different ports.
- TCP SYN flags.
- Fast probing pattern.

**Detection:**

- Signature: SYN pattern to multiple ports.
- Anomaly: Unusual probing.
- Visual: Horizontal clusters.

## 10.2. DDoS Attacks

**Characteristics:**

- High traffic volume.
- Large packets, repeated dest ports.
- Many source IPs.

**Detection:**

- Signature: Volume-based.
- Anomaly: Traffic spike.
- Visual: Dense clusters.

## 10.3. Lateral Movement

**Characteristics:**

- Internal communications.
- Sensitive/management ports.

**Detection:**

- Signature: Unusual access pattern.
- Anomaly: Internal anomalies.
- Visual: Odd internal flow.

### 10.4. Amplification Attacks

**Characteristics:**

- UDP with services like DNS/NTP/SSDP.
- Small requests, large responses.

**Detection:**

- Signature: Known protocol patterns.
- Anomaly: Large response ratio.
- Visual: Tall packets on key ports.

## 11. Extension Guide

### 11.1. Adding New Detection Models

1. Create new model class.
2. Add to `IDSModels`.
3. Implement `predict()`.
4. Update voting mechanism.
5. Add UI controls.

**Example:**

```
self.heuristic_model = CustomHeuristicModel()
predictions['heuristic'] = self.heuristic_model.detect(vec)
votes = ... # use all predictions
```

### 11.2. Implementing New Visualizations

1. Create PyQtGraph widget.
2. Add to relevant tab.
3. Implement update logic.
4. Enable interaction.

**Example:**

```
self.flowGraph = pg.GraphItem()
self.plotWidget.addItem(self.flowGraph)
self.flowGraph.setData(pos=..., adj=..., symbolBrush=...)
```

### 11.3. Integration with External Systems

- SIEM: Send API events.
- Firewalls: Auto rules.
- Threat Intel: IP checks.
- Alerts: Email/SMS.

**Example:**

```
def send_to_siem(event):
    requests.post(siem_url, json=payload)
```

## 12. Troubleshooting

### 12.1. Common Issues

**Packet Capture Fails:**

- Check permissions.
- Verify interface mode.
- Test on active interface.

**High CPU Usage:**

- Lower update frequency.
- Periodically clear data.
- Reduce visualization load.

**Model Accuracy Problems:**

- Improve training data.
- Adjust thresholds.
- Enhance feature extraction.

### 12.2. Performance Considerations

**Memory:**

- Use deque with maxlen.
- Rotate or store in DB for long runs.

**Processing:**

- Batch feature extraction.

- Optimize plot refresh.

**UI Responsiveness:**

- Use separate threads.
- Rate limit updates.
- Async capture for live mode.

## 12.3. Debugging Tips

- Enable verbose logs.
- Visual inspect feature values.
- Compare model outputs.
- Check feature importance.

# 13. Requirements and Dependencies

## 13.1. Software Dependencies

**Python Libraries:**

- PyQt5
- PyQtGraph
- scikit-learn
- numpy
- pandas
- scapy (optional for real mode)

**Versions:**

- Python 3.6+
- PyQt5 5.12+
- scikit-learn 0.24+
- numpy 1.19+
- pandas 1.1+
- scapy 2.4+

## 13.2. Hardware Requirements

**Minimum:**

- CPU: Dual-core 2GHz
- RAM: 4GB
- Storage: 100MB
- Network: Any interface

**Recommended:**

- CPU: Quad-core 3GHz
- RAM: 8GB+
- Storage: 1GB
- Network: Gigabit Ethernet

## 13.3. Installation Guide

1. Install Python 3.6+
2. Install packages:

```
pip install PyQt5 pyqtgraph scikit-learn numpy pandas
pip install scapy  # optional
```

3. Run app:

```
python ids_simulation.py
```

4. For real capture:
   - Run as admin or sudo.
   - Configure firewall permissions.
5. (Optional) Create virtual env:

```
python -m venv ids_env
source ids_env/bin/activate  # Linux/macOS
ids_env\Scripts\activate     # Windows
pip install -r requirements.txt
```

# 14. System Architecture and Class Diagram

This section presents the Class Diagram, which models the structure of the IoT-Based Intrusion Detection System (IDS).

## Class Diagram Overview

The Class Diagram provides a blueprint of the software architecture. It defines the system's primary classes, their attributes and methods, and the relationships between them.

- **Entities and Components:** Includes classes like `Sensor`, `Gateway`, `DataProcessor`, `AlertManager`, and `UserInterface`.
- **Relationships:** Illustrates inheritance, aggregation, and associations (e.g., a `Gateway` aggregates data from multiple `Sensors`).
- **Attributes and Methods:** Shows core fields (e.g., `sensorId`, `timestamp`, `dataValue`) and functions (e.g., `processData()`, `generateAlert()`).

## Key Components

- **Sensors:** Collect and send environmental or system data.
- **Gateways:** Receive data from sensors and transmit to processors.
- **DataProcessors:** Analyze data for anomalies or intrusions.
- **AlertManager:** Handles alert generation and logging.
- **UserInterface:** Displays metrics, logs, and alert information.

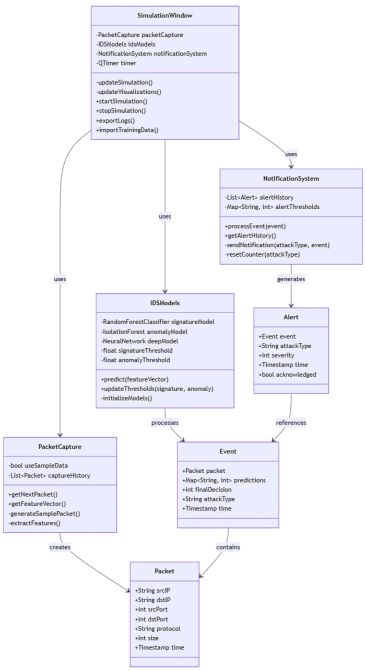


Figure 1: System Class Diagram

# 15. Data Flow Diagrams (DFDs)

## 15.1. Level 0 DFD: High-Level Overview

The Level 0 Data Flow Diagram represents the overall context of the system, identifying its interaction with external entities.

- **External Entities:** Administrators, IoT devices, third-party systems.
- **Main Process:** Single high-level IDS process managing inputs and outputs.
- **Data Stores and Flows:** Shows entry (sensor data), temporary storage (logs), and outputs (alerts).

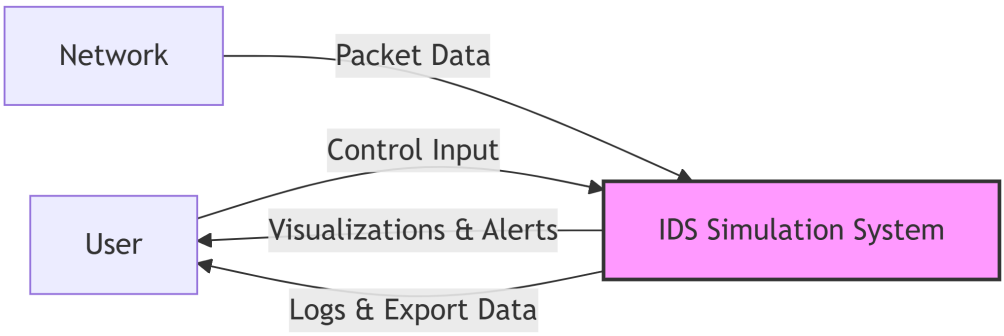


Figure 2: Level 0 DFD - Context Diagram

## 15.2. Level 1 DFD: Detailed Process Breakdown

The Level 1 DFD expands the core process into functional components:

- **Data Collection:** Receives input from sensors.
- **Pre-Processing:** Validates and cleans incoming data.
- **Analysis:** Uses pattern recognition to detect intrusions.
- **Alert Management:** Triggers and logs alerts.
- **Reporting:** Interfaces for administrators.

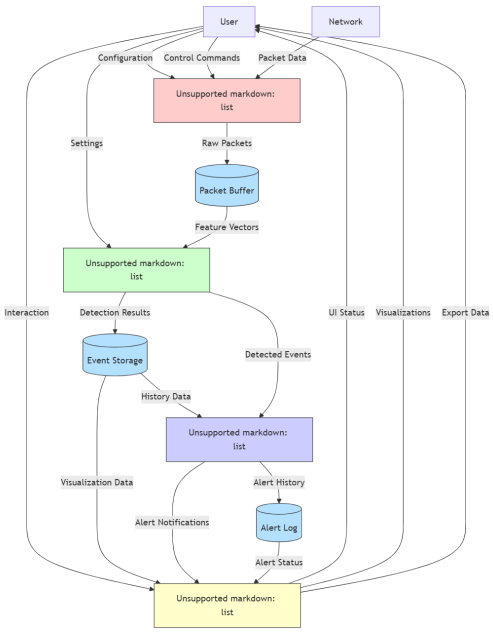


Figure 3: Level 1 DFD - Functional Breakdown

# 16. IoT-Based IDS Workflow Diagram

The workflow diagram outlines the step-by-step process used by the IoT IDS to detect and respond to intrusions in real-time.

## Workflow Steps

1. **Data Acquisition:** Sensors collect real-time data.
2. **Data Transmission:** Data is sent to gateways via wireless or wired networks.
3. **Initial Filtering:** Gateways reduce noise and filter irrelevant packets.
4. **Centralized Analysis:** Aggregated data is analyzed using ML or signature-based algorithms.
5. **Alert Generation:** Alerts are raised upon detecting suspicious activity.
6. **Response and Logging:** Automated mitigation and event logging.
7. **Administrator Notification:** Final alerts are sent to administrators.

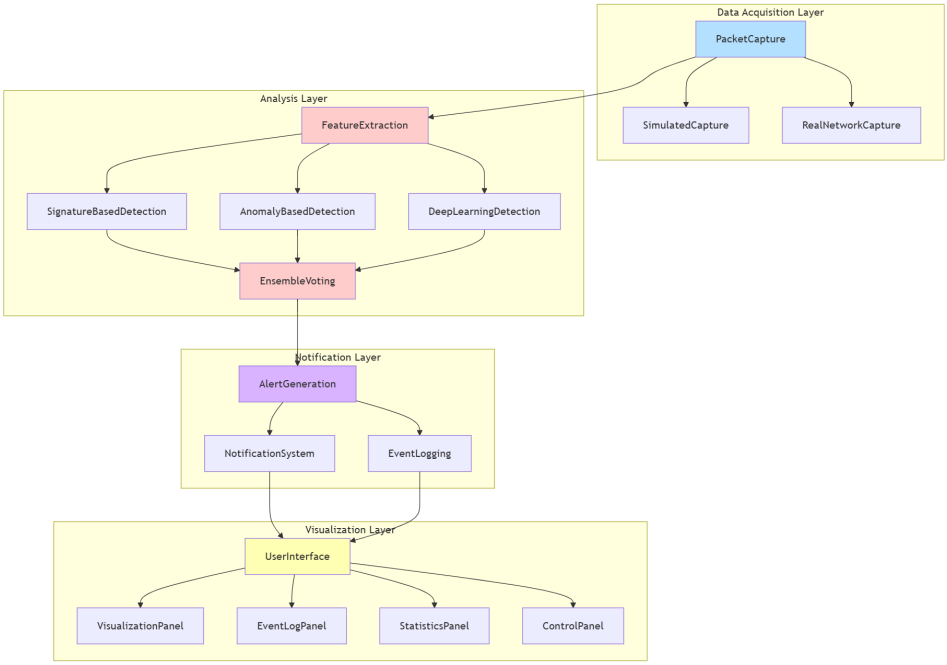


Figure 4: IoT IDS Workflow Diagram

# 17. Network Topology

This section illustrates the logical and physical layout of the IoT-based IDS network.

## Architecture of Connections

- **Device Layout:** Interconnectivity among sensors, gateways, central servers, and admin consoles.
- **Redundancy and Segmentation:** Ensures failover paths and security through segmentation.
- **Security Zones:** Deployment of firewalls and intrusion prevention systems (IPS).
- **Communication Protocols:** Use of MQTT, HTTPS, and TCP/IP for data transmission.

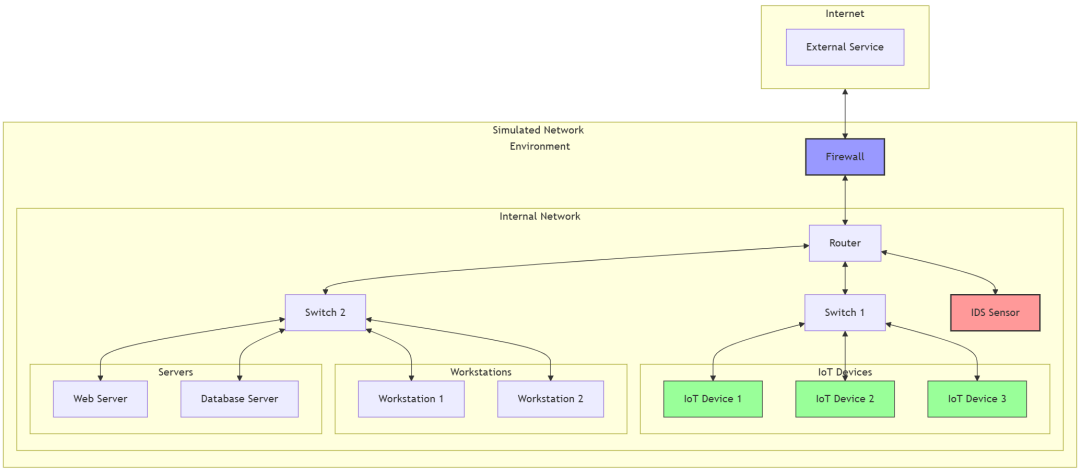


Figure 5: IoT IDS Network Topology

# 18. Network Topology

This section illustrates the logical and physical layout of the IoT-based IDS network.

## Architecture of Connections

- **Device Layout:** Interconnectivity among sensors, gateways, central servers, and admin consoles.
- **Redundancy and Segmentation:** Ensures failover paths and security through segmentation.
- **Security Zones:** Deployment of firewalls and intrusion prevention systems (IPS).
- **Communication Protocols:** Use of MQTT, HTTPS, and TCP/IP for data transmission.

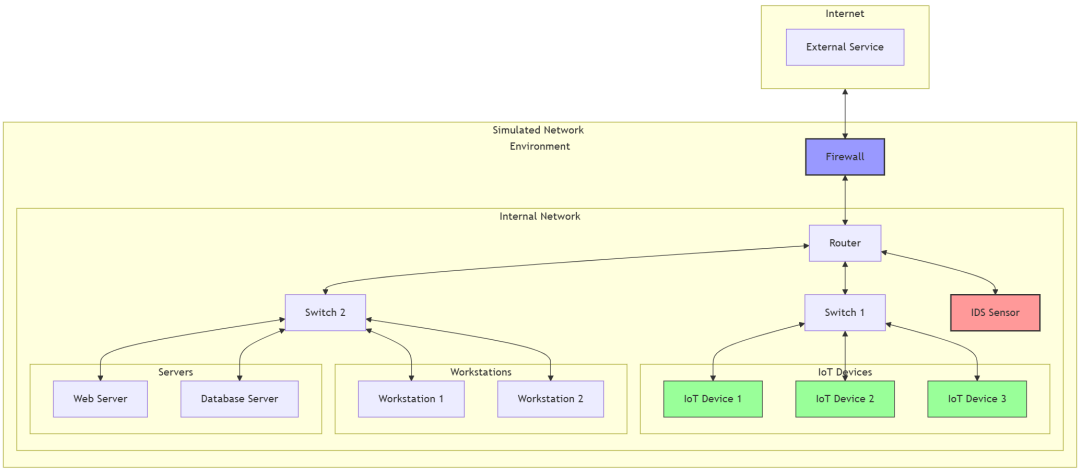


Figure 6: IoT IDS Network Topology

# 19. Conclusion

This documentation provides a comprehensive architectural and operational view of the IoT-based Intrusion Detection System. By integrating diagrams and descriptions, it enables a full understanding of the system's design, functionality, and implementation.

## Diagram Summary

- **Class Diagram:** Defines software components and relationships.
- **DFDs (Level 0 and 1):** Illustrate data flow from inputs to alerts.
- **Workflow Diagram:** Shows real-time operational logic.
- **Network Topology:** Highlights deployment and security structure.

## Final Thoughts

The coordinated use of these elements ensures that the IoT IDS is:

- Secure
- Scalable
- Efficient
- Visually traceable and well-documented

Future improvements may include diagram annotations, integration with threat intelligence feeds, and real-time visualization dashboards for system administrators.