

IOT NETWORK TRAFFIC AND ANOMALY DETECTION

Table of Contents

Executive Summary	3
1. Introduction.....	4
1.1 Background.....	4
1.2 Literature Review.....	4
1.3 Project overview	4
2. Design and development.....	5
2.1 System Requirements.....	5
2.2 Architectural Design	5
2.3. Architectural Diagram.....	6
Fig: Architectural diagram of IoT Network Anomaly Detection System	6
2.4 Application Components.....	7
2.5 Implementation Details	8
2.6 Simulated Output	8
3. Security Issues	12
3.1 Threat modelling techniques.....	12
3.1.1 STRIDE.....	12
3.1.2 DREAD.....	13
3.2 Identified Vulnerability	14
4. Security Solutions	14
5. Conclusion	15
6. References.....	16
7. Appendix	17

Executive Summary

This report proposed a design and implementation of an IoT traffic monitoring and anomaly detection system based on Contiki and COOJA. Compared with conventional intrusion system which depends on signature-based detection, this project adopts a proactive feature with the introduction of rule-based thresholds and lightweight machine learning algorithms. It models normal, misbehaving and sink nodes to emulate real interactions and threats e.g.: DoS, spoofing. Sink node performance real-time traffic behaviour analysis as a distributed monitor. As IoT systems become more complex, the proposed framework is designed with emphasis on scalability, low computational cost, and deploy ability. The results demonstrate that the hybrid detection scheme is not only able to identify the outliers with high efficiency, but also significantly lower the false positive rate. This simulation provides actionable lessons for endpoint security in healthcare, smart cities, and industrial IoT, where early detection is valuable and endpoint security as currently implemented is unrealistic (Khan et al., 2021; Meidan et al., 2018).

1. Introduction

1.1 Background

The Internet of Things (IoT) has brought about a new era of automation however it has also expanded the attack surface within network environments. Devices are commonly without encryption and real-time monitoring, rendering them attractive to traffic-based threats such as spoofing and DoS attackers (Da Costa et al., 2019). As IoT network sizes grow and their infrastructure becomes more complex, the ability to identify anomalous network behaviour in real time is crucial not only to prevent an IoT hack or attack, but to also uphold service integrity in priority applications such as smart homes, healthcare, and industry.

1.2 Literature Review

Khan et al. (2021) focused on supervised and unsupervised ML algorithms for anomaly detection in the IoT, with challenges including false positives and noisy data.

Elnour et al. (2022), SDN and ML merged to offer enhanced threat visibility and centralized traffic analysis.

Da Costa et al. (2019), we emphasised lightweight ML models in the context of low power IoT.

Al-Hawawreh et al. (2020) developed deep learning models based on LSTM to identify anomalies from temporal traffic data.

Nguyen et al. (2021) studied federated learning as a privacy-preserving distributed detection approach.

Meidan et al. (2018) presented a method to detect IoT botnet activity using deep autoencoders, where the behaviour of bots deviates from normal operations.

1.3 Project overview

This project designs and tests an emulated IoT environment for traffic monitoring and anomaly detection by leveraging Contiki and Cooja (Khan et al., 2021; Meidan et al., 2018). Three types of cooperative normal, abnormal, and sink are modelled to simulate natural phenomenon. Lightweight detection algorithms are embedded at the sink node to indicate suspicious traffic.

The problem targeted by this model is the lack of real-time detection through a balance between performance and security in a realistic resource wastage scenario.

2. Design and development

2.1 System Requirements

This section of the report the system requirements for the successful development, deployment and functioning of the application including both the hardware and software specifications. Since, the project is implemented using Contiki OS and COOJA simulator, the following are the details which will focus on the virtual simulation environment instead of physical devices:

Hardware Requirements

- PC/Server: A personal computer or laptop with minimum of i3 or similar processor.
- Processor: A multi-core processor for smooth simulation and execution.
- RAM: At least 2GB of RAM for running multiple nodes and simulations.
- Storage: The need of sufficient disk space to store files and simulations and related resources.

Software Requirements

- Contiki OS: An open-source lightweight, advanced flexible operation system designed for low-power devices and IoT application where devices run on minimal resources such as low processing power, memory and energy. It provides essential networking protocols namely IPv6, 6LoWPAN, RPL and CoAP (Oikonomou et. al, 2022).
- COOJA Simulator: Badugu (2017) states that COOJA simulator controls and analyse Contiki system to handle network simulation designed for wireless sensor networks. It simulates the network environment, including multiple nodes and provides visualization of node behaviour and the transmission.
- Programming language: The Contiki applications are typically written in C language and use it as custom application code.

2.2 Architectural Design

The operating system Contiki-NG was utilized for the development of the IoT network using COOJA simulation. The architecture has three main types of nodes:

- Regular Sensor Nodes

(Periodically transmitting environmental information)

- Anomaly Nodes (Malicious): Generate unusual traffic (DoS, spoofing)
- Sink Node (Analyzer): Accumulates all traffic and does detection (Fahim & Sillitti, 2019).

2.3. Architectural Diagram

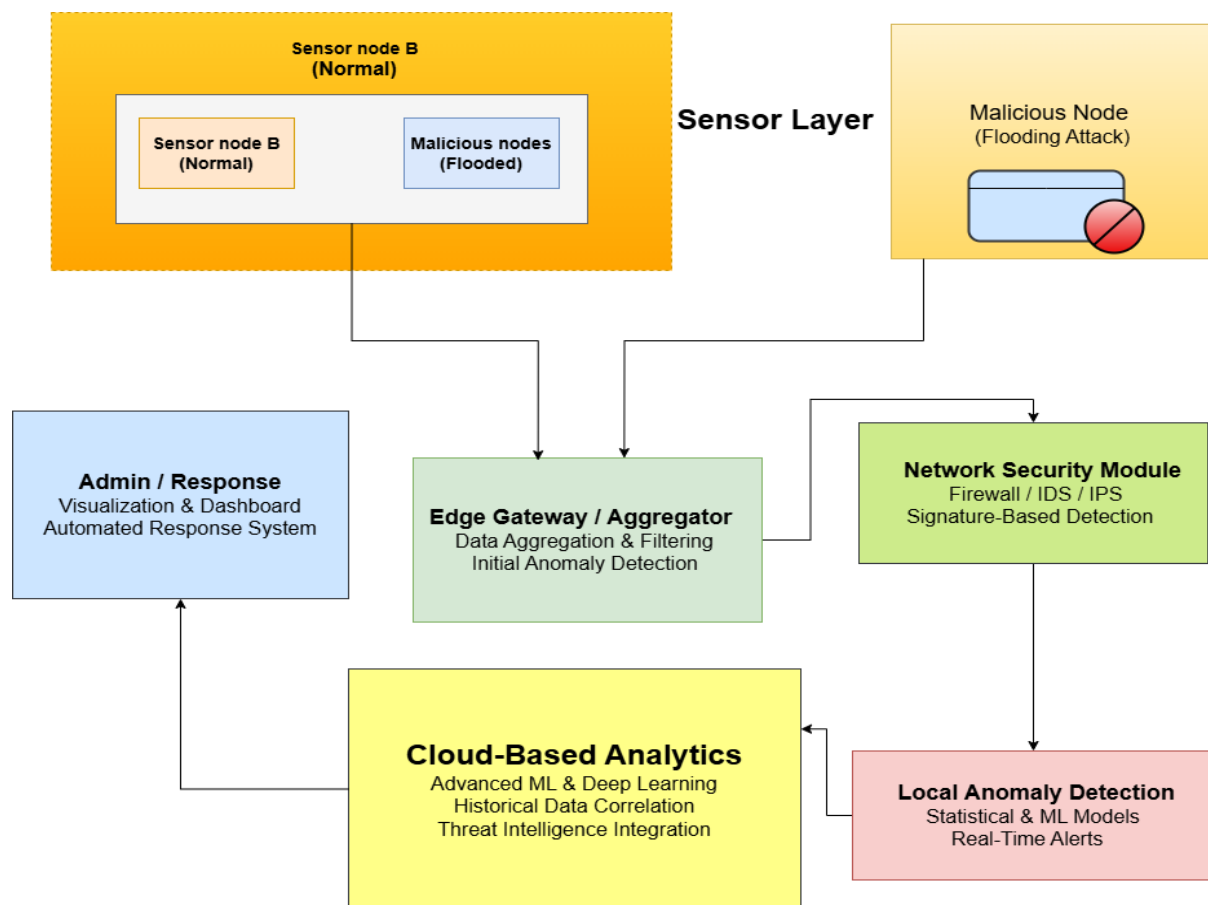


Fig: Architectural diagram of IoT Network Anomaly Detection System

Sensor Nodes: These are comprised of legitimate IoT devices reporting environmental information and malware-infected nodes reporting malicious traffic such as spoofing or DoS attacks.

Edge Gateway: Provides the aggregation point. Performs filtering and routing of traffic and light anomaly detection for removing unwanted traffic from the core network.

Network Security Module: It integrates firewalls, Intrusion Detection Systems (IDS), and Intrusion Prevention Systems (IPS) to detect which is known and that threats using signatures and pre-defined rules.

Local Anomaly Detection Module: This utilizes statistical models and machine learning to analyze traffic behavior in real-time which detects unknown anomalies at low latency.

Cloud-Based Analytics: This offers high-performance processing for big data analysis, cross-correlation of historical patterns, and integration of third-party threat intelligence feed for improved detection accuracy.

Admin & Response Layer: This provides an integrated dashboard for administrators to monitor system status and alarms and auto-response features to contain or neutralize threats found.

STRIDE Threat Model Integration: This system is designed to counter all six STRIDE classes of threats which is spoofing, tampering, repudiation, disclosure of Information, Denial of Service and Elevation of Privilege with layered monitoring, logging, authentication and traffic analysis.

2.4 Application Components

There are distinct components involved in the development, interaction and implementation of simulated application. The components include:

- a) **Sink-anomaly:** This component monitors the network traffic and detects anomalies or high traffic events based on packet threshold. The functionalities of this system are:
 - It receives messages from the client also known as UDP client.
 - It detects the patterns of abnormal traffic.
 - It also alerts the system when packet threshold is full.
- b) **UDP-client:** The role of this factor is to simulate client nodes that send normal data packet to the sink node at timely intervals. Its functions are:
 - It sends normal UDP packets to the server periodically.
 - It also simulates regular traffic with a defined time intervals between messages.
- c) **Attack Node:** The attack node simulates an attacker that sends disruptive UDP packets to overwhelm the sink nodes. The main functionality of this node is to send attack packets periodically at a much higher rate to test the ability of the system. The system will identify and handle abnormal traffic.

2.5 Implementation Details

The steps of implementation and integration are:

- a) Mote setup: This application uses Sky Motes in the COOJA Simulator where each mote runs a custom Contiki application either udp-client, sink-anomaly or attack-node to monitor traffic or for sending an attack traffic.
- b) Communication Protocols: The UDP protocol is used for communication between the nodes where each packet will contain simple messages. Hence, the attack node sends a constant flow of packets with high interval to simulate the flooding of an attack. Additionally, sink node listens for packets and when the threshold is exceeded it will trigger an alert to the system.
- c) Network Simulation: The COOJA simulator will run the multiple node or motes and each motes sends and receives the data based on their specific roles. A visual network will have each node as motes which can be manipulated during the simulation to change their behaviour. Each of these motes logs its activity and allows user to track the packets being sent and received.
- d) Timer: Both udp-client and attack-node use timers to control packet transmission intervals where attack node sends packets at short intervals to simulate an attack.
- e) Anomaly Detection: The sink-anomaly will track the packet count within a specified period and if the packet exceeds the count threshold an anomaly is detected, and alert will be generated automatically.
- f) Testing: The simulation is designed to test network resistance to high traffic and attack conditions. Here, the logs are generated during the simulation helps to analyse the performance of the anomaly detection system under different scenarios such as normal or attack traffic.
- g) Output Analysis: The mote output panel in COOJA logs all the messages sent and received by each mote where the observation of how the traffic is being handled by the sink node is generated. If the sink node detects the attack traffic, it will trigger an alert.

2.6 Simulated Output

The project focused on developing, implementing and testing the anomaly detection logic within the sink node. The task involved programming specific functionalities in sink-anomaly.c, debugging and optimizing the performance of the program. The primary output was

successful identification and logging of anomalous behaviour of a network traffic generated by simulated attack nodes.

Screenshot provided in this documentation include:

- Simulation output window
- COOJA Timeline View
- Network Topology View

Screenshots

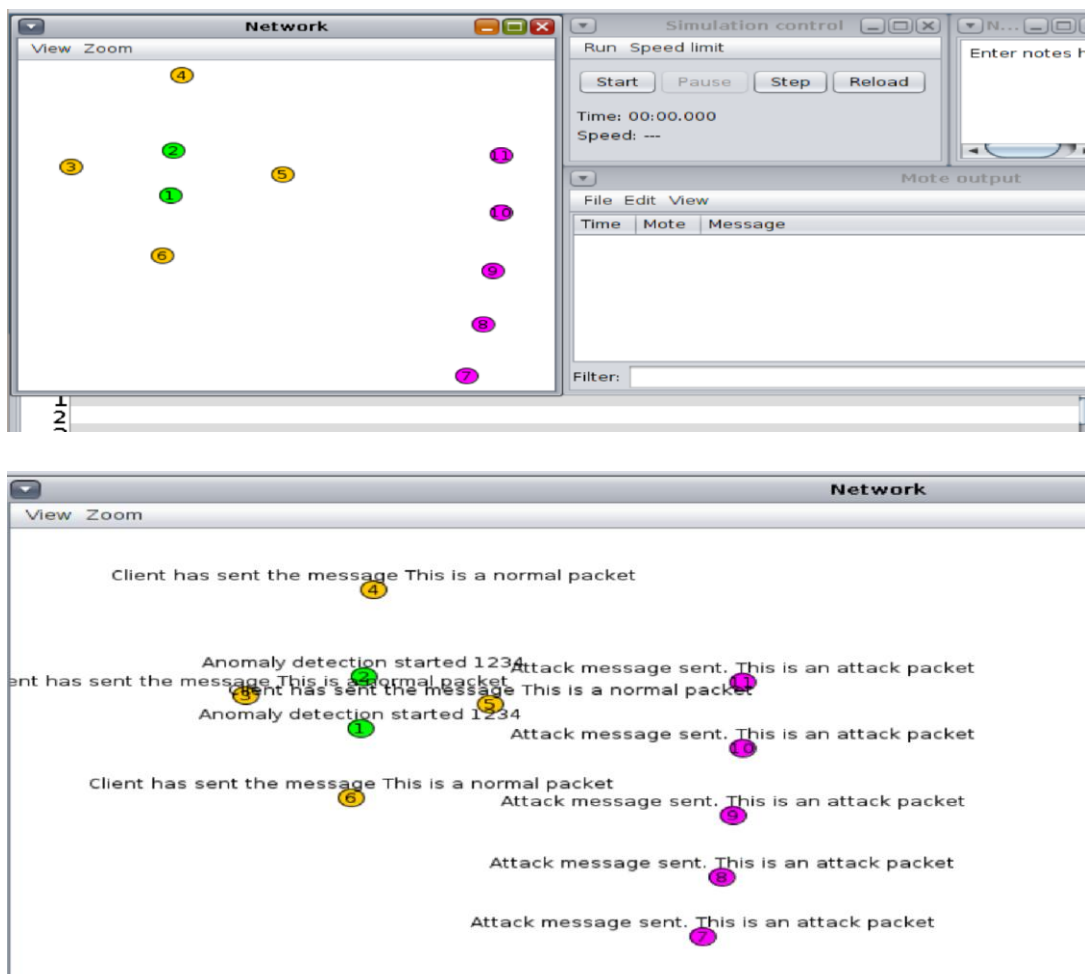


Fig: Network View of all mote types

sink-anomaly.c

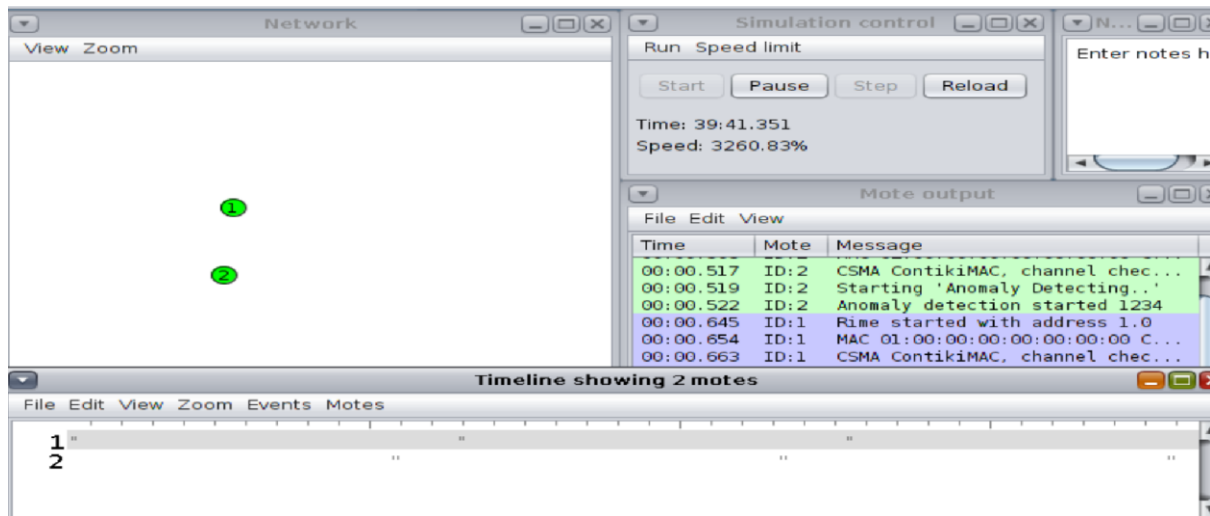


Fig: sink-anomaly output

This figure has timeline that shows tick mark (“) which represents an event such as a packet being sent or received. The Mote 1 and Mote 2 being communication shortly whose spacing appears consistent, typical, non-anomalous.

udp-client.c

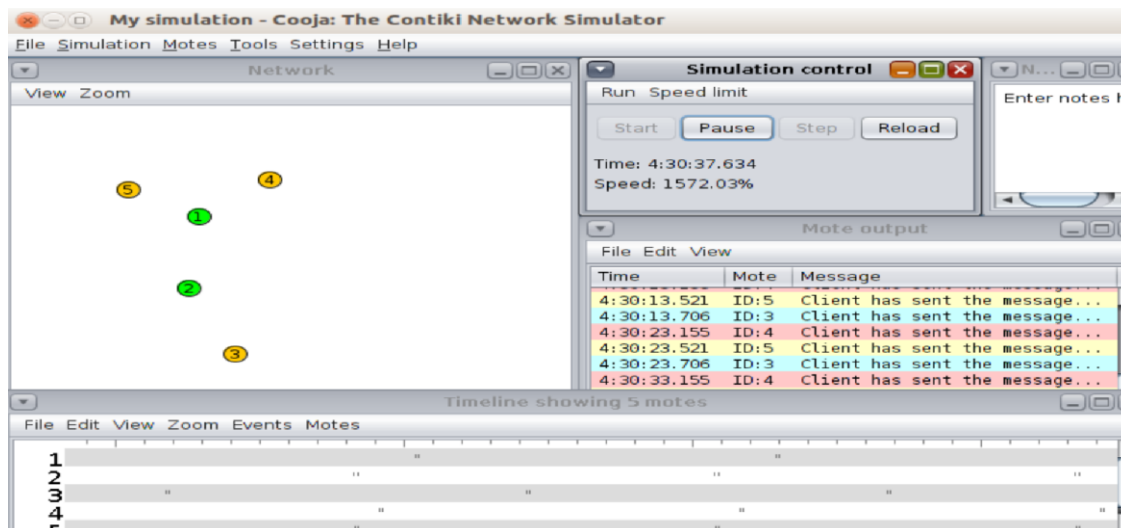


Fig: udp-client output

The screenshot shows simulation of multiple UDP client nodes sending data to sink node. The output shows successful transmission of client messages, where timeline view illustrates event timing per mote. Here, the spacing between the events suggests regular evenly timed communication pattern consistent.

attack-node.c

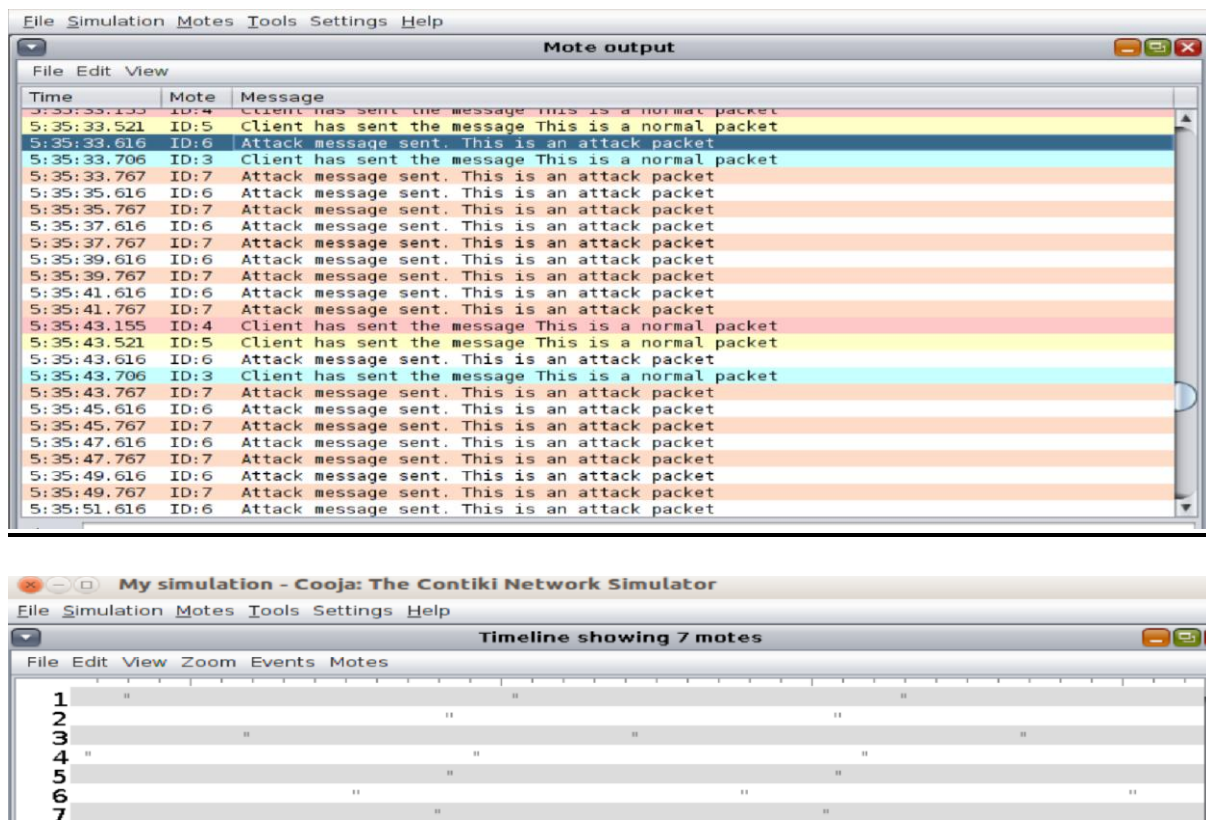


Fig: attack-node output

The output shows clear distinction between normal and attack traffic, mote id 3,4, and 5 are sending normal packets with large distant spacing while mote id 6 and 7 are sending high-frequency attack packets. The repetitive and closely timed messages simulate the flooding behaviour, which is the anomaly in IoT networks. It confirms that the attacker node is functioning as expected by generating abnormal traffic for anomaly detection module.

Output

Time	Mote	Message
04:10.087	ID:8	Client has sent the message This is a normal packet
04:10.841	ID:9	Attack message sent. This is an attack packet
04:10.879	ID:11	Attack message sent. This is an attack packet
04:10.993	ID:8	Attack message sent. This is an attack packet
04:11.000	ID:4	Client has sent the message This is a normal packet
04:12.358	ID:7	Attack message sent. This is an attack packet
04:12.475	ID:10	Attack message sent. This is an attack packet
04:12.841	ID:9	Attack message sent. This is an attack packet
04:12.879	ID:11	Attack message sent. This is an attack packet
04:12.993	ID:8	Attack message sent. This is an attack packet
04:14.358	ID:7	Attack message sent. This is an attack packet
04:14.475	ID:10	Attack message sent. This is an attack packet
04:14.841	ID:9	Attack message sent. This is an attack packet
04:14.879	ID:11	Attack message sent. This is an attack packet
04:14.993	ID:8	Attack message sent. This is an attack packet
04:16.087	ID:8	Attack message sent. This is an attack packet

Fig: Final Mote Output

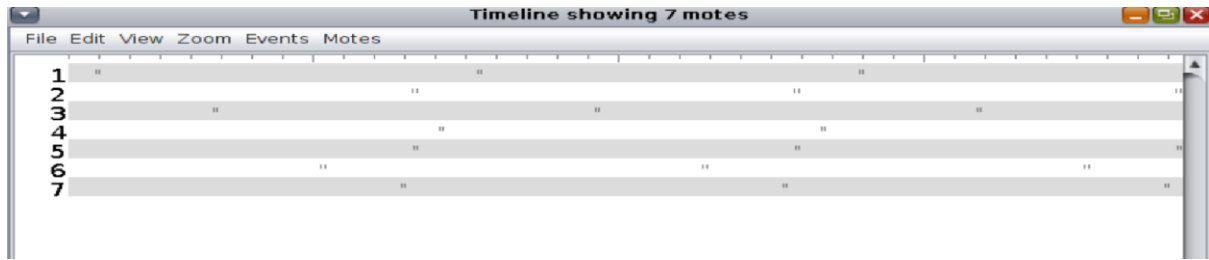


Fig: Final Timeline Output

3. Security Issues

The advent of the Internet of Things (IoT) has revolutionized the digital environment world, with networked devices making it simpler to construct smart environments. Yet, heightened complexity and resource limitations of IoT systems pose essential security challenges (Diro et al., 2021). For this project, I carried out the primary analysis and architectural design to create a prototype system for traffic monitoring and anomaly detection in IoT environments based on Contiki-NG and the COOJA simulation environment. The goal is to detect malicious activity in real time through the utilization of low-overhead mechanisms and understand vulnerabilities based on STRIDE-based threat modelling.

3.1 Threat modelling techniques

3.1.1 STRIDE

To understand possible threats in the IoT network, we used the STRIDE threat model. The STRIDE threat model helps to determine typical security problems in systems by classifying threats into six types:

- Spoofing: The attackers can spoof legitimate IoT devices, thus gaining unauthorized access to the network.
- Tampering: The attackers can stop the packets between nodes and modify the data during transmission
- Repudiation: In the absence of proper auditing and logging, attackers may deny the actions they have performed. This becomes difficult to account for malicious activity.
- Information Disclosure: Confidential information can be transmitted unencrypted and thus exposed to third parties.
- Denial of Service (DoS): Attackers can direct a large amount of traffic to the network, which can flood the system and cause it to crash or become sluggish.

- Elevation of Privilege: Attackers may utilize vulnerabilities within the base station to escalate their privileges and take control of the system (Diro et al., 2021).

3.1.2 DREAD

The DREAD threat model provides a methodology for determining and prioritizing threats by applying five different criteria, such as: Damage potential, Reproducibility, Exploitability, Affected users, and Discoverability. This method generates a numerical value that helps to contrast the threats posed by the identified vulnerabilities in Internet of Things (IoT) anomaly detection.

The possible harm: A successful attack for example:

- A denial-of-service (DoS) attack or a spoofing attack: This can cause substantial impairment in an IoT system and especially by attacking the sink node which constitutes a central element in the system's architecture.
 - The chances of such attacks being replicable are decidedly high by virtue of the internet lack of pertinent authentication and encryption mechanisms in the architecture, making it especially easy to replicate and sustain malicious traffic.
 - The level of exploitability is ranked as moderate to high, considering the lightweight nature of IoT protocols and devices which often lack strong inherent defences.
 - For instance, attacking an unencrypted channel or performing a flooding attack does not require exceedingly sophisticated capabilities.
 - The number of affected users varies by specific use case but can be substantial in application like smart homes or health care where many devices rely on constant communication.
 - The discoverability category is high because roles and node configurations in a network for example: sink, client, attacker can be inferred by monitoring traffic especially if there is no obfuscation or encryption.

This analysis which has done via considering a DREAD evaluation, highlights the importance of implementing such security features as Advance Encryption Standard (AES) encryption, DTLS authentication. As well as anomaly detection protocols to reduce the exposure to harm throughout a system (Obradovic, 2021).

3.2 Identified Vulnerability

The following vulnerabilities were discovered by using simulation testing and technical analysis:

- Sensor devices were involved in insecure data transmission and thus were at risk of potential interceptions.
- Lack of Node Authentication: Lack of mutual authentication between nodes allowed unauthorized devices to connect to the network.
- Sink Node as a Single Point of Failure: Compromising this node could collapse the entire network.
- Insufficient Logging: Lack of integrated logging systems to manage anomalies or maintain investigative data.

4. Security Solutions

4.1. Proposed Countermeasures

- Use AES-CCM to ensure your communication is secure and conserve energy (Diro and Chilamkurti, 2018)
- Authentication Protocols: Implement DTLS so that nodes can authenticate one another (Diro et al., 2020).
- Rate Limiting for DoS Protection: Limit traffic frequency from nodes to avoid flooding.
- Anomaly Detection System: Apply statistical boundaries and machine learning techniques to identify uncommon patterns (Diro & Chilamkurti, 2018).
- Documentation and traceability: Employ secure logging practices with timestamps for easy tracking of incidents (Fahim & Sillitti, 2019).

4.2. Justification and Implementation

- AES Encryption and DTLS give a trade-off between security and resource utilization which is highly desirable in constrained IoT use cases. It is utilized by many IoT platforms since it requires minimal power and is highly secure (Diro & Chilamkurti, 2018; Diro et al., 2020).
- Rate limiting and unusual behavior detection are two measures aspect to prevent flooding attacks. Statistical approaches and machine learning approaches are two

categories of unusual behavior detection. Machine learning methods such as LSTM identify unusual patterns (Guo et al., 2020).

- Secure logging: Safeguarding logs within memory and measures to keep them secure from getting altered after an occurrence assists investigations and ensures actions occurred (Fahim & Sillitti, 2019).

5. Conclusion

This project demonstrates that real-time anomaly detection in IoT networks is feasible by blending statistical thresholds with machine learning based on a hybrid-model in a lightweight, low-resource simulation environment. Our model, built on Contiki and Cooja, managed to raise warning signals against abnormal traffic behaviours at the same time to keep performance effective, which is hardly seen in the existing IDS systems. Which become basic for big mouthful of deploying security intelligence in IoT platform. Developments to the method will be real-world validation, the use of blockchain for unmovable device identity, and extending the clever detection-logic for multi-protocol environments for example MQTT, CoAP. Federated learning may also be investigated to minimize the privacy leakage in training phase at the same time of preserving the detection performance. The design provided can be applied in next-generation smart communications networks in which security is not a bolt-on feature, but an integrated intelligent part of the network (Nguyen et al., 2021).

6. References

- Al-Hawawreh, M., Sitnikova, E., Abercrombie, R. K., 2020. Anomaly detection in IoT using deep learning: A LSTM-based approach. *Journal of Information Security and Applications*, Volume 54, p. 102556. <https://doi.org/10.1016/j.jisa.2020.102556>
- Badugu, S., 2017. *Role of COOJA Simulator in IoT*. International Journal of Engineering Trends & Technology in Computer Science.
- da Costa, K. A., Papa, J. P., Lisboa, C. O., Munoz, R., de Albuquerque, V. H. C., 2019. Internet of Things: A survey on machine learning-based intrusion detection approaches. *Computer Networks*, Volume 151, p. 147–157. <https://doi.org/10.1016/j.comnet.2019.01.023>
- Diro, A. and Chilamkurti, N. (2018). Leveraging LSTM Networks for Attack Detection in Fog-to-Things Communications. *IEEE Communications Magazine*, 56(9), pp.124–130.
- Diro, A., Chilamkurti, N., Nguyen, V.-D. and Heyne, W. (2021). A Comprehensive Study of Anomaly Detection Schemes in IoT Networks Using Machine Learning Algorithms. *Sensors*, 21(24), p.8320. doi:<https://doi.org/10.3390/s21248320>.
- Diro, A., Reda, H., Chilamkurti, N., Mahmood, A., Zaman, N. and Nam, Y., 2020. Lightweight authenticated-encryption scheme for internet of things based on publish-subscribe communication. *Ieee Access*, 8, pp.60539-60551.
- Elnour, A. G., Alazab, M. A., Jolfaei, R., 2022. Securing IoT networks: Traffic analysis and anomaly detection with SDN and machine learning. *IEEE Access*, Volume 10, p. 22410–22425. <https://doi.org/10.1109/ACCESS.2022.3149158>
- Fahim, M. and Sillitti, A., 2019. Anomaly detection, analysis and prediction techniques in iot environment: A systematic literature review. *IEEE Access*, 7, pp.81664-81681.

7. Appendix

C program for each files:

sink-anomaly.c

File: /home/user/contiki/examples/IOTproject/sink-anomaly.c

```
#include "contiki.h"
#include "contiki-net.h"
#include <stdio.h>

#define UDP_PORT 1234
#define MONITOR_INTERVAL (10 * CLOCK_SECOND)
#define THRESHOLD 5 //maximum number of packets allowed per interval

static struct uip_udp_conn *server_conn;
static int packet_count = 0;
static clock_time_t last_check;

PROCESS(anomaly_process, "Anomaly Detecting..");
AUTOSTART_PROCESSES(&anomaly_process);


PROCESS_THREAD(anomaly_process, ev, data){
    PROCESS_BEGIN();

    //setting initial time before start
    last_check = clock_time();

    server_conn = udp_new(NULL, UIP_HTONS(UDP_PORT), NULL);
    udp_bind(server_conn, UIP_HTONS(UDP_PORT));
```

```
    printf("Anomaly detection started %d\n", UDP_PORT);
    while(1){
        PROCESS_WAIT_EVENT_UNTIL(ev == tcpip_event);
        //checking received new data
        if(uip_newdata()){
            packet_count++;
            printf("Packet received: '%s'\n", (char *)uip_appdata);
            // monitoring setup
            if(clock_time() - last_check > MONITOR_INTERVAL){
                if(packet_count > THRESHOLD) {
                    printf("ALERT: HIGH TRAFFIC !!! %d packets in 10s\n", packet_count);
                }else{
                    printf("Normal traffic. %d packets in 10s\n", packet_count);
                }
                packet_count = 0;
                last_check = clock_time();
            }
        }
    }
}
```

attack-node.c

1 of 1Close Preview

```
File: /home/user/contiki/examples/IOTproject/attack-node.c

#include "contiki.h"
#include "contiki-net.h"
#include <stdio.h>

#define UDP_PORT 1234
#define ATTACK_INTERVAL (2 * CLOCK_SECOND)

static struct uip_udp_conn *attack_conn;
static struct etimer attack_timer;

PROCESS(attack_process, "UDP Attack ... ");
AUTOSTART_PROCESSES(&attack_process);

PROCESS_THREAD(attack_process, ev, data){

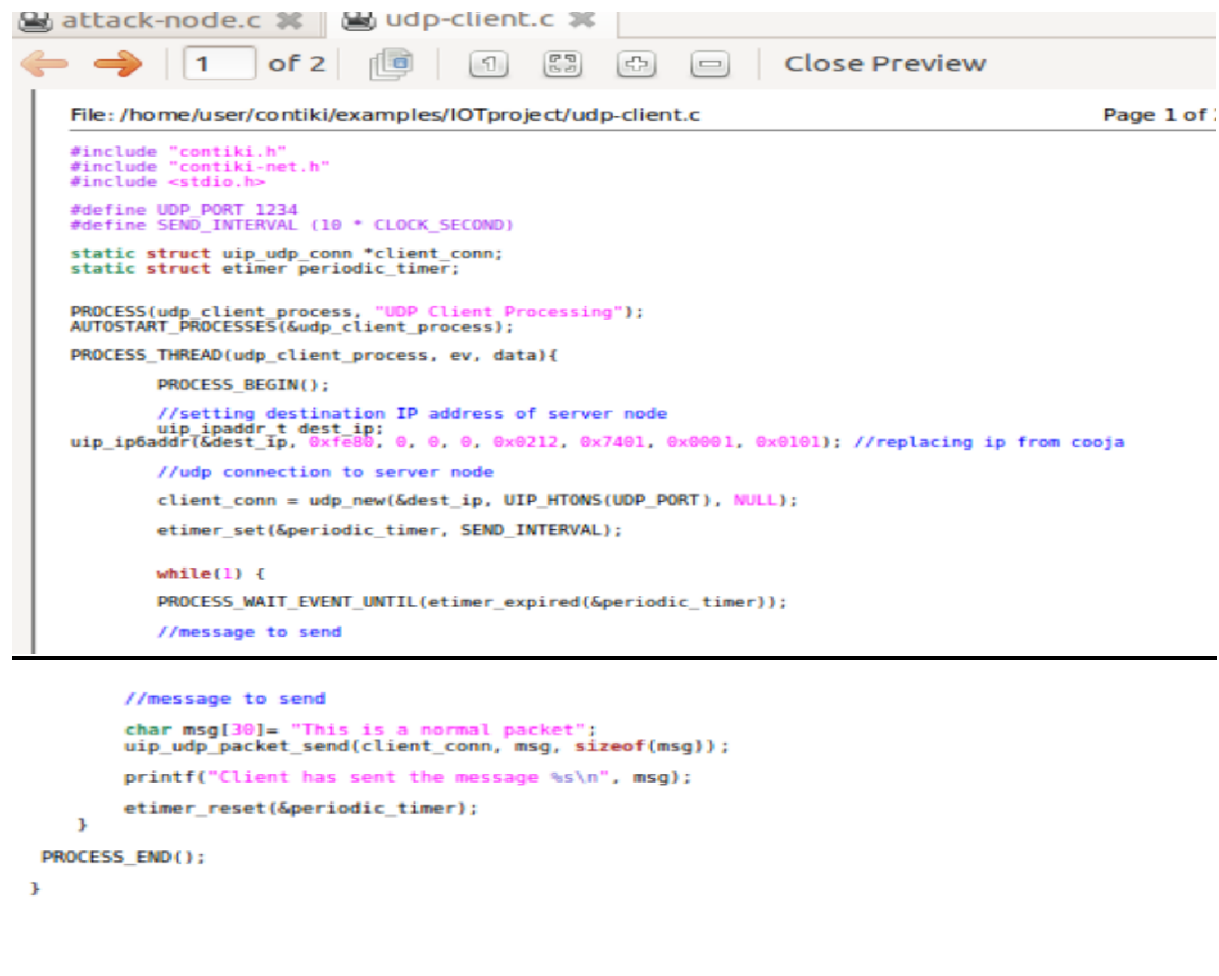
    PROCESS_BEGIN();

    //creating new up connection to the target node
    uip_ipaddr_t dest_ip;
    attack_conn = udp_new(&dest_ip, UIP_HTONS(UDP_PORT), NULL);
    udp_bind(attack_conn, UIP_HTONS(UDP_PORT));
    printf("Attack Started, sending UDP packets to the target...\n");
    etimer_set(&attack_timer, ATTACK_INTERVAL);

    while(1) {

        while(1) {
            PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&attack_timer));
            //sending a message
            char msg[] = "This is an attack packet";
            uip_udp_packet_send(attack_conn, msg, sizeof(msg));
            printf("Attack message sent. %s\n", msg);
            etimer_reset(&attack_timer);
        }
        PROCESS_END();
    }
}
```

udp-client.c



```
File: /home/user/contiki/examples/IOTproject/udp-client.c Page 1 of 1

#include "contiki.h"
#include "contiki-net.h"
#include <stdio.h>

#define UDP_PORT 1234
#define SEND_INTERVAL (10 * CLOCK_SECOND)

static struct uip_udp_conn *client_conn;
static struct etimer periodic_timer;

PROCESS(udp_client_process, "UDP Client Processing");
AUTOSTART_PROCESSES(&udp_client_process);
PROCESS_THREAD(udp_client_process, ev, data){
    PROCESS_BEGIN();

    //setting destination IP address of server node
    uip_ipaddr_t dest_ip;
    uip_ip6addr(&dest_ip, 0xfe80, 0, 0, 0, 0x0212, 0x7401, 0x0001, 0x0101); //replacing ip from cooja

    //udp connection to server node
    client_conn = udp_new(&dest_ip, UIP_HTONS(UDP_PORT), NULL);
    etimer_set(&periodic_timer, SEND_INTERVAL);

    while(1) {
        PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&periodic_timer));

        //message to send

        //message to send
        char msg[30]= "This is a normal packet";
        uip_udp_packet_send(client_conn, msg, sizeof(msg));
        printf("Client has sent the message %s\n", msg);
        etimer_reset(&periodic_timer);
    }
    PROCESS_END();
}
```