TECHNICAL UNIVERSITY OF CRETE

DIPLOMA THESIS

Implementation of a Trust-Based Intrusion Detection System for IoT Networks

---

Author:
Sofia
KAFRITSA-GEORGANTA

Thesis Committee:
Prof. Sotiris IOANNIDIS
Prof. Apostolos DOLLAS
Prof. Michael PATERAKIS

A thesis submitted in fulfillment of the requirements for the diploma of Electrical and Computer Engineer

in the

School of Electrical and Computer Engineering
Microprocessor and Hardware Laboratory

July 31, 2024

TECHNICAL UNIVERSITY OF CRETE

Abstract

School of Electrical and Computer Engineering

Electrical and Computer Engineer

Implementation of a Trust-Based Intrusion Detection System for IoT Networks

by Sofia KAFRITSA-GEORGANTA

The Internet of Things (IoT) deployments are recognized as playing a pivotal role in the digital transformation era, with the potential to revolutionize all economic sectors and human activities. However, the widespread adoption of IoT solutions is impeded by significant security risks that necessitate advanced cyber defense mechanisms. Intrusion Detection Systems (IDS) are considered a crucial element of these security measures.

In this thesis, a specific type of IDS that revolves around the concept of trust is investigated. Given that IoT nodes can exhibit unpredictable behavior, which can resemble malicious activity and have similar detrimental effects on the overall IoT deployment, an IDS solution based on assessing the trustworthiness of nodes is developed. This solution is deployed in a distributed manner across the IoT network, effectively monitoring the network behaviors of nodes to compute various metrics and determine whether a node exhibits acceptable behavior or shows signs of unreliability or malicious activity.

These decisions are inherently dynamic and form a trust vector. Configurable thresholds enable the IDS to trigger alarms about potential offending nodes, while the computation of trust is continuously adaptive to a node's behavior. This allows trust to be withdrawn or re-established, preventing intermittent errors from having permanent effects on the IoT network. The overall solution is designed to have low performance and memory requirements, making it suitable for all types of IoT nodes.

TECHNICAL UNIVERSITY OF CRETE

Abstract

School of Electrical and Computer Engineering

Electrical and Computer Engineer

Implementation of a Trust-Based Intrusion Detection System for IoT Networks

by Sofia KAFRITSA-GEORGANTA

Οι εφαρμογές του Διαδικτύου των Πραγμάτων (IoT) διαδραματίζουν καθοριστικό ρόλο στην εποχή του ψηφιακού μετασχηματισμού, με τη δυνατότητα να φέρουν επανάσταση σε όλους τους οικονομικούς τομείς και τις ανθρώπινες δραστηριότητες. Ωστόσο, η ευρεία υιοθέτηση των λύσεων IoT παρεμποδίζεται από σημαντικούς κινδύνους ασφαλείας που απαιτούν προηγμένους μηχανισμούς κυβερνοάμυνας. Τα Συστήματα Ανίχνευσης Εισβολών (IDS) είναι ένα κρίσιμο στοιχείο αυτών των μέτρων ασφαλείας.

Σε αυτή τη διπλωματική εργασία, εξετάζεται ένας συγκεκριμένος τύπος IDS που περιστρέφεται γύρω από την έννοια της εμπιστοσύνης. Δεδομένου ότι οι κόμβοι του IoT μπορεί να εμφανίσουν απρόβλεπτη συμπεριφορά, η οποία μπορεί να μοιάζει με κακόβουλη δραστηριότητα και να έχει παρόμοια επιζήμια αποτελέσματα στην συνολική ανάπτυξη του IoT, αναπτύσσεται μια λύση IDS με βάση την αξιολόγηση της αξιοπιστίας των κόμβων. Η λύση αυτή αναπτύσσεται με κατανεμημένο τρόπο σε όλο το δίκτυο IoT, παρακολουθώντας αποτελεσματικά τις συμπεριφορές των κόμβων για να υπολογιστούν διάφορες μετρήσεις και να καθοριστεί εάν ένας κόμβος εμφανίζει αποδεκτή συμπεριφορά ή παρουσιάζει σημάδια αναξιοπιστίας ή κακόβουλης δραστηριότητας.

Αυτές οι αποφάσεις είναι εγγενώς δυναμικές και σχηματίζουν έναν φορέα εμπιστοσύνης. Τα ρυθμιζόμενα κατώφλια επιτρέπουν στο IDS να ενεργοποιεί συναγερμούς για πιθανούς παραβατικούς κόμβους, ενώ ο υπολογισμός της εμπιστοσύνης προσαρμόζεται συνεχώς στη συμπεριφορά ενός κόμβου. Αυτό επιτρέπει την ανάκληση ή την επαναφορά της εμπιστοσύνης, αποτρέποντας τα διαλείποντα σφάλματα να έχουν μόνιμες επιπτώσεις στο δίκτυο IoT. Η συνολική λύση σχεδιάζεται να έχει χαμηλές απαιτήσεις απόδοσης και μνήμης, καθιστώντας την κατάλληλη για όλους τους τύπους κόμβων IoT.

Acknowledgements

I am deeply grateful to my family and friends for their consistent support throughout my academic journey. I extend my heartfelt thanks to my supervisors, Andreas Brokalakis and Sotiris Ioannidis, whose invaluable guidance and assistance have been instrumental. Their expertise and unwavering support have significantly contributed to the development of my research.

Contents

List of Figures

List of Tables

List of Abbreviations

ACK	**ACK**nowledgement
AEC	**A**verage **E**nergy **C**onsumption
AI	**A**rtificial **I**ntelligence
ARP	**A**ddress **R**esolution **P**rotocol
CPU	**C**entral **P**atagram **P**rotocol
CNTD	**C**Clustered **N**eighbor based **T**rust **D**issemination
DoS	**D**enail of **S**ervice
EPFR	**E**nd-to-End **P**acket **F**orwarding **R**atio
FIN	**FIN**ish
HIDS	**H**ost **I**ntusion **D**etection **S**ystem
ICMP	**I**nternet **C**ontrol **M**essage **P**rotocol
IDS	**I**ntusion **D**etection **S**ystem
IO	**I**nput/**O**utput
IP	**I**nternet **P**rotocol
IoT	**I**nternet of **T**hings
JSON	**J**ava**S**cript **O**bject **N**otation
MitM	**M**an-in-the **M**iddle
NBTD	**N**eighbor **B**ased **T**rust **D**issemination
NIDS	**N**etwork **I**ntusion **D**etection **S**ystem
OSI	**O**pen **S**ystems **I**nterconnection
PDR	**P**acket **D**elivery **R**atio
PSH	**PuSH**
QoS	**Q**uality of **S**ervice
RST	**ReSeT**
SSL	**S**ecure **S**ockets **L**ayer
STD	**ST**andard **D**eviation
SVM	**S**upport **V**ector **M**achines
SYN	**SYN**chronize
TCP	**T**ransmission **C**ontrol **P**rotocol
TLS	**T**ransport **L**ayer **S**ecurity
TTD	**T**ree based **T**rust **D**issemination

TRM	**T**rust **M**anagement **M**odel
UDP	**U**ser **D**atagram **P**rotocol
VM	**V**irtual **M**achine

Dedicated to my family and friends…

Chapter 1

Introduction

1.1 Motivation

In the era of digital transformation, the Internet of Things (IoT) has emerged as a revolutionary force, integrating technology into various aspects of our daily lives. From smart cities to healthcare, agriculture, and manufacturing, IoT devices have brought about a new level of efficiency and connectivity. This revolution not only optimizes operational methodologies but also fosters a more data-driven approach to decision-making, enhancing the quality of life and paving the way for future innovations.

However, this widespread adoption has also introduced significant security challenges. The high level of connectivity among devices, greatly increases the threat surface, as their interlinked nature opens up numerous potential weaknesses that attackers can take advantage of. The intricate web of IoT ecosystems therefore presents a wider range of attack opportunities that demand advanced security measures to counter these diverse threats.

One of the primary steps towards providing an effective protection mechanism is the ability to detect malicious activity and attack efforts. Typically this is handled by Intrusion Detection Systems (IDS) and particularly for IoT environments, trust-based IDS have emerged as a promising solution. These systems employ "Trust" as a dynamic, comprehensive metric to evaluate and monitor the behavior and reliability of network nodes. This approach allows for a nuanced assessment of each node's trustworthiness by analyzing its historical behavior, communication patterns, and interactions within the network. By continuously updating trust scores, the system can identify and respond to anomalies indicative of potential security threats, ensuring a proactive and adaptive security posture tailored to the evolving landscape of IoT threats. This method leverages the inherent network relationships and

behaviors, providing a mechanism to detect and mitigate risks before they can cause significant damage.

1.2 Thesis Contributions

The goal of this thesis is to design and implement a distributed Trust-based Intrusion Detection System (IDS) within IoT networks. This system leverages trust metrics as a novel approach to enhance security measures by dynamically evaluating the trustworthiness of network entities. The present work aims to introduce a flexible and adaptive system capable of responding to network errors and potential security threats through modulated Trust assessments. Specifically, the system decreases trust scores in response to alert-triggering events and gradually restores Trust as normal behavior resumes. The keypoints of this work are the following:

- Network Monitoring Tool Implementation: A software tool tailored for monitoring network activities within IoT environments has been developed.
- Normal Sensor Behavior Evaluation: Typical operations of a sensor within the IoT network are analyzed to establish a baseline profile and identify key metrics. These metrics serve as a standard against which ongoing network activity is measured, enabling the identification of deviations indicative of potential security threats.
- Detection Tool and Trust Evaluation Algorithm: The core of the thesis involves creating an algorithm that detects abnormalities in network traffic or behavior. Upon detection, the system will adjust Trust scores accordingly, decreasing scores in the face of suspicious activity and incrementally increasing them when normal patterns resume. This mechanism ensures the system remains vigilant against threats while avoiding undue penalties for temporary or benign anomalies.

The system is designed to work in co-operation with the rest of the security tools developed within IntellIoT [1], a Pan-European Research and Innovation project supported by the European Commission, that fosters the development of humanised IoT and AI devices and systems.

1.3 Thesis Outline

- **Chapter 2 - Theoretical Background:** Chapter 2 provides the necessary theoretical background concepts related to the goal of this thesis.
- **Chapter 3 - Related Work:** Chapter 3 provides a brief overview of existing works in literature and this thesis' approach.
- **Chapter 4 - Implementation:** Chapter 4 provides the methodology of the thesis' approach
- **Chapter 5 - Evaluation:** Chapter 5 reports the results of our proposed implementation
- **Chapter 6 - Conclusions and Future Work:** Chapter 6 concludes the thesis and provides future work related to the approach

Chapter 2

Theoretical Background

2.1 Internet of Things (IoT)

The concept of the Internet of Things (IoT) as outlined by researchers in [2], [3] and [4], envisions a network environment where various physical objects like devices, buildings, individuals and smart entities are interconnected through the internet. This definition is further elaborated upon by [3] who describe IoT as a model of connectivity. In this context, nearly all digital and physical electronic devices are anticipated to be linked through the Internet Protocol (IP) allowing them to communicate and interact with each other forming an extensive network ecosystem. This revolutionary shift in connectivity highlighted in [5] and [6] is fostering the development of applications in fields such as education, healthcare, remote monitoring, transportation and surveillance despite the emergence of complex security issues resulting from the diverse structure of these interconnected systems.

Drawing from the theoretical foundations discussed above, the practical core of IoT revolves around nodes. Nodes can be defined as physical objects integrated with sensors and actuators, within a service oriented framework. These units independently gather, transmit and process data to create an intelligent and responsive environment. The structure of the Internet of Things, which consists of layers such as the perception layer, network layer and application layer as outlined in [7] and demonstrated in 2.1, allows physical objects to be transformed into digital entities. These entities then interact through channels and network elements to support IoT applications and tackle security challenges inherent in a vast interconnected system.

The fusion of the virtual worlds via IoT not only improves efficiency and decision making in various sectors, but also underscores the crucial importance of robust network design and secure communication protocols. It is essential

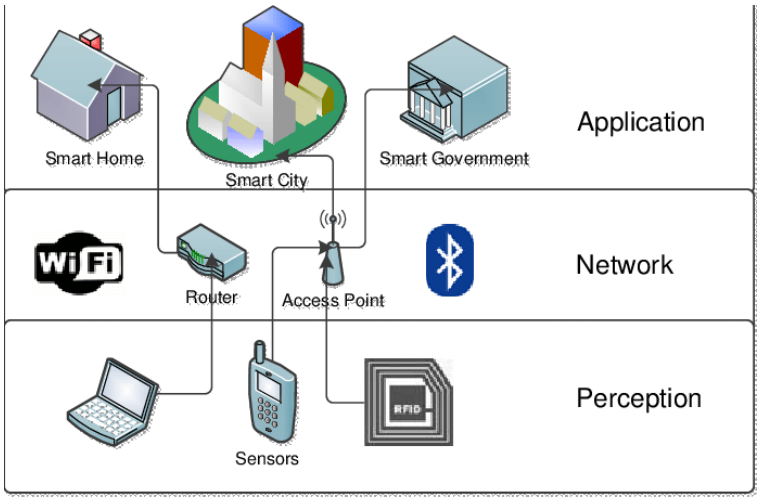

FIGURE 2.1: Three-layer IoT Architecture by [8]

to prioritize security and privacy as data travels between nodes within this framework due to the sensitive nature of the processed information. Furthermore, integrating trust mechanisms into networks highlights the importance of validating the integrity of nodes and the authenticity of data. This underlines the complex interplay between the deployment of technology and the development of policies, crucial for maintaining the integrity and reliability of IoT systems.

2.2 Intrusion Detection Systems

An Intrusion Detection System (IDS) is a security tool designed to monitor network or system activity for suspicious behavior or policy violations and produce reports to a management station. Its primary purpose is to detect unauthorized access, misuse, modification or denial of computer network and system resources. IDS solutions can be broadly classified into several types based on their detection methods and deployment strategies. For IoT environments, an IDS can be strategically placed in several locations to bolster security:

1. **Directly on IoT Devices** (Host Intrusion Detection Systems-HIDS): Installing IDS software on IoT devices allows for immediate detection of threats at the device level, although the limited processing power of some devices might restrict this option's feasibility.

2. **Within the IoT Network** (Network Intrusion Detection Systems-NIDS):

Integrating IDS within the IoT's network infrastructure enables the examination of traffic flows between devices, identifying potential intrusions or unauthorized access.

3. **Edge Computing Locations**: Implementing IDS at the network's edge - closer to where data is produced yet not on the devices themselves - combines efficient security monitoring with reduced response times.

4. **On Gateway Devices**: IoT gateways bridge IoT devices to the internet or other networks. IDS placed at these gateways can scrutinize the traffic entering and exiting the IoT environment, offering a critical checkpoint for security.

5. **Cloud Integration**: For IoT systems utilizing cloud computing for data processing and storage, cloud-integrated IDS can monitor for threats against cloud-hosted resources, utilizing the cloud's expansive processing capabilities for thorough traffic analysis.

Deciding on the placement of IDS within an environment, relies on various elements, such as the particular security needs, the architecture of the IoT system, the types of devices in use and the potential attack paths that require safeguarding. A successful approach frequently includes a layered strategy incorporating several IDS deployments throughout different areas of the ecosystem for thorough defense.

2.2.1 Types of Intrusion Detection Systems

Intrusion Detection Systems (IDS) are critical components within cybersecurity frameworks, aimed at detecting and responding to potential security breaches. IDS are categorized based on their detection methodologies and operational focus. An understanding of the different types of IDS facilitates the selection of the most appropriate system for specific network environments and security requirements. Presented below are various types of IDS and their unique characteristics, as mentioned in [9].

Signature-based Intrusion Detection Systems (SIDS)

Signature-based Intrusion Detection Systems (SIDS) use pattern matching techniques to identify known threats. These systems rely on a database of predefined attack signatures. When the system detects a sequence of commands or actions that match a stored signature, it triggers an alert. This

method is effective for recognizing known threats by comparing current activities against known patterns.

Anomaly-based Intrusion Detection Systems (AIDS)

Anomaly-based Intrusion Detection Systems (AIDS) create a model of normal behavior for a system and flag deviations from this model as potential threats. These systems monitor current activities and compare them to established models of normal behavior. Deviations from this normal behavior are flagged as potential threats. Techniques used include statistical models, machine learning, and knowledge-based methods. AIDS can identify unusual behaviors that do not conform to typical patterns.

Host-based Intrusion Detection Systems (HIDS)

Host-based Intrusion Detection Systems (HIDS) monitor data originating from the host system. These systems analyze operating system logs, application logs, and file system changes to detect potentially malicious activities. HIDS can monitor encrypted communications by analyzing user behavior. Installation on each host is required, and it consumes host resources.

Network-based Intrusion Detection Systems (NIDS)

Network-based Intrusion Detection Systems (NIDS) monitor network traffic to detect malicious activities. These systems analyze incoming and outgoing packets on a network. They are effective at identifying threats before they reach the target system by monitoring traffic patterns and behaviors. NIDS operate on the network level and can handle traffic from multiple systems, making them suitable for high-bandwidth networks.

Hybrid Intrusion Detection Systems

Hybrid Intrusion Detection Systems combine the techniques of SIDS and AIDS. These systems leverage the strengths of both signature-based and anomaly-based detection methods. By combining the two approaches, hybrid IDS can reduce false positives and improve the detection of unknown threats. They adapt to new attack patterns and provide a comprehensive security solution by using a combination of known signatures and behavioral analysis.

Trust-based Intrusion Detection System (TIDS)

A Trust-based Intrusion Detection System (IDS) in the context of the Internet of Things (IoT) refers to a security mechanism that utilizes trust evaluation metrics to identify and mitigate (in collaboration with other security components) potential security threats and malicious activities within a network. Unlike traditional IDS solutions that rely solely on signature or anomaly detection techniques, a trust-based IDS assesses the reliability and integrity of devices and their communications based on established trust relationships and behaviors observed over time. In the next subsection, we introduce the concept of trust.

2.3 Trust

Trust is a multidimensional concept, that plays a role in the realm of network security, representing the faith and assurance in the dependability, honesty and integrity of a system, network or entity. In trust-based intrusion detection systems, managing trust involves the establishment and reinforcement of trust connections among network nodes or entities, to efficiently identify malicious behavior. Trust mechanisms are employed to evaluate the behavior and reputation of nodes, assessing their trustworthiness to make decisions that enhance network security and protect against threats. Trust is crucial for maintaining data integrity and confidentiality, ensuring the availability of network resources, and securing communication pathways.

In [10] and [11] the authors undertake a comprehensive exploration of trust computation models, emphasizing its pivotal role in maintaining the security and reliability of IoT systems. They construct their classification on five essential stages, providing a structured approach to understand how trust is formed, computed, propagated, updated, and aggregated within IoT environments. The stages identified are:

1. **Trust Composition** - what elements to consider, when computing trust. This involves determining the basis on which trust scores are calculated.

   (a) Quality of Service (QoS) Trust
   This aspect refers to the belief that an IoT device is able to respond to a service request. It evaluates trustworthiness based on performance. Metrics such as competence, cooperativeness, and task

completion capability are pivotal, with trust being derived from the device's ability to meet or exceed predefined service quality standards. In [12] authors used end-to-end packet forwarding ratio, energy consumption, and packet delivery ratio to measure QoS trust.

(b) Social Trust
Social trust in IoT systems reflects the perceived trustworthiness based on interactions and relationships among devices. It incorporates elements such as reputation, recommendations, and historical interactions, assessing the influence of social networks and community standings on trust decisions. Essentially, social trust evaluates a device's commitment and goodwill to perform a service request, adding a layer of trust assessment beyond technical metrics.

2. **Trust Propagation** - how trust information is propagated to peers.

   (a) Distributed Schemes
   Trust information is shared among devices in a peer-to-peer fashion, allowing devices to independently calculate and update trust scores based on direct interactions and recommendations from other devices in the network.

   (b) Centralized Schemes
   Involves a central entity that collects, processes, and disseminates trust information among devices, acting as a single point of trust data management.

3. **Trust Aggregation** - the method by which individual pieces of trust evidence are combined to form an overall trust assessment.

   (a) Weighted Sum [13]
   A technique to combine multiple pieces of trust evidence, assigning different weights to each based on their relevance or credibility.

   (b) Belief Theory [14]
   Also known as Dempster-Shafer theory, it allows combining evidence from different sources to calculate a degree of belief (or trust) in a proposition.

(c) Subjective Logic [15]
A framework for modeling uncertainty and belief, enabling the calculation of trust as a subjective opinion based on evidence.

(d) Certain Logic [16]
valuates propositions under uncertainty, compatible with probabilistic approaches and subjective logic, allowing for a nuanced expression of trust certainty.

(e) Bayesian Inference with Belief Discounting [17]
Uses Bayesian statistics to update the trust level based on new evidence, incorporating mechanisms to discount beliefs in the face of uncertain or unreliable information.

(f) Fuzzy Logic [18]
Applies fuzzy logic principles to model trust with degrees of truth rather than binary true/false values, accommodating partial truth and uncertainty.

(g) Regression Analysis [19]
A statistical method to identify the relationships between trust and various factors, predicting trust behavior based on observed patterns.

4. **Trust Update** - when trust is updated.

   (a) Event-driven
   Trust levels are updated in response to specific events or interactions, allowing for dynamic adjustments based on recent activities.

   (b) Time-driven
   Trust is updated at regular intervals, regardless of events, to reflect changes over time and decay in trust, if no new evidence is provided.

5. **Trust Formation** - how to form the overall trust out of multiple trust properties.

   (a) Single-trust Approach
   Focuses on a singular trust property or dimension to assess trustworthiness, simplifying the trust decision to a key aspect.

   (b) Multi-trust Approach
   Considers multiple dimensions of trust simultaneously, offering a

more comprehensive assessment by integrating various trust factors.

2.4 Attacks

In the context of the Internet of Things, an attack is any malicious activity that causes harm or downgrades the system. Attackers can disrupt the normal operation of the system, control IoT devices for malicious purposes, steal sensitive information or cause physical damage. According to [20], attacks can be categorized based on several characteristics, severity, target or access level. Layer-wise, attacks can be categorized as follows:

1. **Physical Layer**: Attacks here involve attacks directly affecting the hardware components of IoT devices.
2. **Data Link Layer**: This layer is susceptible to attacks that disrupt the direct data exchange over physical connections.
3. **Network Layer**: Targets the routing of data packets across the network.
4. **Transport Layer**: Focuses on disrupting the reliable delivery of data packets between devices.
5. **Application Layer**: Attacks at this level aim at the applications directly, exploiting vulnerabilities to execute malicious code, data breaches, or service disruptions.

2.4.1 Physical Layer Attacks

The physical layer serves as the bedrock of the IoT stack, interfacing directly with the hardware components and the transmission media. It is tasked with the binary transmission of data and governs how devices convert digital data into signals and vice versa. This layer is fundamental because any disruption here can cascade up the stack, compromising overall network integrity. According to [20], [21] and [22] some attacks on physical layer are:

- Jamming Attack: Jamming attacks disrupt communication by overwhelming the physical channels that IoT devices use, especially in wireless communications like Wi-Fi, Bluetooth, or cellular networks. These attacks are executed by emitting noise or stronger signals on the same frequency bands as the target devices, effectively drowning out legitimate communications. Attackers may continuously transmit on a specific

frequency to block device communications or exploit protocol weaknesses by disrupting the communication process required for establishing connections. The consequences of jamming can range from temporary communication blackouts to prolonged denial of service, affecting critical operations across various applications in healthcare, industrial settings, and smart infrastructure. These threats highlight the necessity for resilient communication strategies and protocols to mitigate the risks of signal interference and ensure continuous service availability in IoT networks.

- Tampering Attack: Tampering attacks specifically refer to unauthorized modifications of physical devices or their operational settings. These can include altering firmware, manipulating sensors, or physically damaging the device to impair its function. Tampering can also involve unauthorized changes that affect the device's output, leading to incorrect operations or malicious activities. For IoT environments, the risk is significant because devices are often deployed in accessible or remote locations, making them vulnerable to physical attacks. Ensuring physical security measures, using tamper-resistant and tamper-evident technologies, and monitoring device integrity are crucial defenses against these attacks.

2.4.2 Data Link Layer Attacks

The data link layer is the protocol layer that handles the moving of data into and out of a physical link in a network. It is responsible for node-to-node data transfer—a link between two directly connected nodes. It also detects and possibly corrects errors that may occur in the physical layer. Given its crucial role in error detection and frame synchronization, security breaches at this layer can lead to significant data integrity challenges. According to [20] and [23] here are some attacks on Data Link Layer:

- Collision: This is a DoS attack, where a node induces a collision in some small part of a transmitted packet. The packet will then fail the checksum check, because of the changes brought on by the collision, and the receiver node will then ask for a retransmission of the packet.
- Exhaustion and Unfairness: This attack is a collision attack taken a bit further. A malicious node may conduct a collision attack repeatedly in order to exhaust the power supply of the communicating nodes.

2.4.3 Network Layer Attacks

Occupying a central role in the Open Systems Intercommunication (OSI) [24] model, the network layer manages device addressing, tracks the location of devices on the network, and determines the best way to move data. This includes routing of packets from the source to the destination across multiple hops in the network. Attacks at this layer can therefore redirect or misdeliver packets, undermining the network's ability to effectively communicate. Some attacks, according to [20] and [25] affecting the Network Layer are:

- Spoofed or Altered Routing Information: In a spoofing and altering routing information attack, a malicious node manipulates routing data within the network to disrupt normal operations. By spoofing—falsifying data to impersonate other nodes—altering, or replaying routing information, an attacker can introduce severe network issues such as routing loops, create wormholes, or even form black holes where data packets are silently dropped. Such attacks can also lead to network partitioning, where segments of the network are isolated from each other, severely degrading network performance and reliability. This type of attack undermines the integrity of routing protocols and can lead to inefficient network operations, data loss, or unauthorized access to network traffic.

- Selective Forwarding: A selective forwarding attack involves a compromised node selectively dropping packets it is supposed to forward. This type of attack disrupts network communication by targeting specific data types or sources, significantly impacting data integrity and network reliability. The attack can evade detection by intermittently dropping packets, making it challenging to identify and isolate the malicious node. This tactic undermines the trustworthiness and effectiveness of the network's data transmission.

- Sybil Attack: A Sybil attack in network security occurs when a single node illegitimately takes on multiple identities or "Sybil nodes" to deceive network protocols. Particularly in the context of multipath routing protocols, this deceptive node can manipulate the protocol by making it appear as though multiple unique paths are available. However, all these paths actually converge through the Sybil node itself. This can compromise the integrity of data routing, potentially leading to traffic interception or a bottleneck, as all data is funneled through a single

point controlled by the attacker. This type of attack is especially harmful in decentralized systems where trust and identity verification are crucial.

- Wormholes: A wormhole attack in network security mirrors its cosmic counterpart by creating a secretive shortcut that links two distant points within the network. In this attack, data packets, including routing information and acknowledgments, are secretly transported through this hidden channel. This manipulation misleads nodes into believing they are neighbors with nodes that are physically far apart, causing confusion in routing protocols that depend on measuring accurate distances between nodes. This type of attack can underpin various malicious activities, such as eavesdropping, disrupting service by withholding data packets, or altering information in the packets before they reach their destination.

2.4.4 Transport Layer Attacks

The transport layer is responsible for end-to-end communication over a network. It provides reliable, transparent transfer of data between end points, and it manages error correction, data flow control and congestion control. Attacks on the transport layer can target various protocols, with the most common being the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP). UDP, due to its connectionless nature, is susceptible to amplification attacks, where small requests trigger larger responses, overwhelming the target with traffic. TCP is often exploited in attacks like **Denial of Service (DoS)** [26] and **Flooding** attacks. A Denial of Service attack is a security threat that aims to interrupt or suspend the services of a host connected to the internet. This interruption can be achieved through various methods, but the primary goal is always to make a machine or network resource unavailable to its intended users. DoS attacks can target any layer in the OSI model, not just the transport layer, affecting everything from physical network components to application-level services. A flooding attack is a subtype of DoS attacks that specifically involves overwhelming a network or service with excessive traffic. This kind of attack saturates the bandwidth of the victim's network and the processing capacity of its infrastructure, effectively blocking legitimate traffic from being processed and causing denial of service.

The characteristics of these attacks can vary, but they typically include:

- High Rate of Large Packets: In many DoS attacks, the aggressor sends a high volume of large packets to maximize bandwidth usage and quickly consume the target's network resources. Large packets can also exploit buffer overflow vulnerabilities, potentially allowing attackers to execute arbitrary code or disrupt service operations.
- High Rate of Small Packets: Alternatively, attackers may send a high volume of small packets. This strategy aims to exhaust server resources like CPU and memory by increasing the number of packets the server must process, rather than consuming bandwidth. Additionally, small packets increase the overhead due to a higher number of packet headers being processed, further straining the network infrastructure.
- Protocol Exploitation: Some attacks specifically exploit features or weaknesses in the TCP protocol, such as the way it handles session initiation or termination.

Understanding these general characteristics helps in identifying and mitigating these threats. Specific types of **TCP-based flooding** attacks include:

- SYN Flood Attack
  A SYN flood attack exploits the TCP three-way handshake by sending a high volume of SYN packets, typically from spoofed IP addresses, to a target server. The server, expecting to complete legitimate connection requests, responds with SYN-ACK packets and waits for a final ACK that never arrives. This can exhaust server resources due to an accumulation of half-open connections. Advanced variations of this attack might utilize slow-rate tactics to avoid triggering rate-based network defense mechanisms, thereby remaining under the radar for extended periods.
- FIN Flood Attack
  In a FIN flood attack, numerous TCP packets with the FIN flag set are sent unexpectedly to a server. This misuse of the TCP flag that typically signals the end of a connection can confuse the target, leading to unnecessary and resource-intensive operations to close non-existent sessions. The attack specifically aims to disrupt the normal termination process, leading to a rapid depletion of system resources.
- RST Flood Attack
  In an RST flood, the attacker sends numerous TCP packets with the RST flag, intended to abruptly close established connections. This attack

can cause immediate disruption of ongoing sessions, leading to a rapid denial of service.

- ACK Flood Attack
  In an ACK flood, the attacker sends a large number of ACK packets to the target. This can confuse the target's network by making it appear as though it is receiving acknowledgments for packets it never sent, which can consume bandwidth and processing power.

The above attacks can be categorized based on the connection flooding and connection termination attacks. Connection flooding attacks exploit the process by which TCP establishes connections between hosts. These attacks typically involve overwhelming a server with an excessive number of incomplete or bogus connection requests, which consume significant server resources and can lead to service unavailability. The most notorious type of connection flooding is the SYN flood, where attackers exploit the TCP handshake mechanism to create a large number of half-open connections that drain the server's resources. On the other side, connection termination attacks disrupt the normal termination process of TCP connections. By sending premature or unsolicited connection termination packets (FIN or RST), attackers can cause unexpected closure of active sessions, leading to data loss or service disruption. Such attacks leverage the normal mechanisms of TCP connection closure to confuse the target system, often resulting in denial of service due to resource exhaustion as systems try to clean up nonexistent or already-closed sessions.

Other DoS Attacks are:

- TCP Window Size Attack: By manipulating the window size in the TCP header, attackers can control the flow of data. A maliciously set zero window size stops all data transmission, while erratic changes can destabilize a connection, leading to timeouts and dropped connections.
- Zero Window Attack: Advertises a window size of zero, effectively telling the sender to stop sending data and holding the connection open indefinitely.
- Land Attack: Sends a spoofed SYN packet to a target machine with the source IP and port set to the target's IP and port, causing the machine to reply to itself and potentially hang.

- TCP Urgent Pointer Attack: Exploits the urgent pointer field by setting it to an unexpected high value along with URG flags, potentially causing buffer overflows or unexpected application behavior

Furthermore, **Scanning and Probing Attacks** utilize crafted packets to gather information about a network or its components. These attacks typically scan the network to discover open ports, services, and vulnerabilities that can be exploited for further intrusion or attack. This reconnaissance is critical for attackers to plan more destructive actions:

- TCP Port Scanning
  This technique involves sending messages to multiple ports on a server to determine what services are active. Port scanning itself is often used for benign purposes such as network management, but in the hands of an attacker, it is a powerful reconnaissance tool. Methods include:
  - TCP SYN Scan: Sending SYN packets and noting which ports respond with a SYN-ACK, indicating an open port.
  - TCP FIN Scan: Involves sending FIN packets to ports, where no response typically indicates an open port, exploiting a specific behavior of certain TCP/IP stacks.
  - NULL Scan: Sends a packet with no flags set (a rare condition), which can confuse or bypass certain types of network defenses that expect at least one flag to be set on TCP packets.
- Christmas Tree Packet Attack
  Named for having every option in the TCP header set (turning on all the "lights"), Christmas tree packets are used for probing network responses. These packets have the URG, PUSH, and FIN flags set, which can trigger unusual responses or behaviors from different network devices or firewalls. This helps an attacker understand how a network might respond to more malicious intrusions.

Another type of attack are **Session Hijacking Attacks**. Session hijacking is a type of network security attack in which an attacker takes over a web session by stealing or manipulating a valid session token or exploiting vulnerabilities in the session management implementation. This kind of attack exploits the stateful connection aspect of session management that relies on token-based authentication or other identifiers that confirm the identity of the parties involved in a communication. The goal is to impersonate a user and carry

out unauthorized activities, such as stealing data or executing unauthorized transactions. Two attacks belonging in this type are the following:

- TCP Session Hijacking: A security breach where an attacker takes control of a TCP session between two endpoints by intercepting or using stolen session tokens. The attacker manipulates sequence numbers in TCP packets—vital for maintaining the correct order of data packets—to inject unauthorized packets that seem legitimate. This can lead to severe issues such as data theft and unauthorized actions performed under the guise of a legitimate user. To mitigate such attacks, it is essential to use encrypted communication protocols like TLS/SSL, employ strong session management techniques, and conduct anomaly detection to identify unusual session activities.
- Man-in-the-Middle (MitM) Attack: This attack involves an attacker secretly altering or intercepting communications between two parties who believe they are directly in contact with each other. This is often achieved through techniques like ARP spoofing or compromising network equipment to intercept all messages sent between the victims. The consequences can be drastic, including eavesdropping, data manipulation, and financial fraud. Defenses against MitM attacks include the use of end-to-end encryption, rigorous validation of network certificates, and heightened awareness regarding the security of network connections, especially on public or unfamiliar networks.

Furthermore, other **Network Manipulation Attacks** as mentioned in [27] and [28], can be:

- Replay Attack
  Replay Attacks involve the interception and retransmission of valid data packets to create unauthorized transactions or disrupt the sequence of events in network communication. Replay attacks may not directly cause denial of service but can confuse or disrupt applications by repeating specific actions or messages.
- Jellyfish Attack
  Similar to replay attacks but specifically targeted at disrupting TCP sessions by intermittently injecting old or duplicate packets into the flow, causing confusion and delays in the network, potentially leading to timeouts and disruptions.

2.4.5 Application Layer Attacks

At the pinnacle of the OSI model, the application layer is the closest to the end-user and provides network services directly to applications. It includes everything from email to file transfer and web browsing, acting as the window through which users interact with the network. It encompasses a wide array of protocols that define how data is presented and communicated, making it a ripe target for exploits aiming to manipulate user interactions and data. As mentioned in [20] and [29] attacks on Application Layer can be:

- Clone Attack: A clone attack in IoT involves an attacker creating a malicious replica of a legitimate node and then reintroducing it to the network. This cloned device is equipped with stolen valid credentials or cryptographic keys, allowing it to masquerade as a genuine part of the network. The attack is particularly dangerous because it enables the attacker to surreptitiously intercept or manipulate sensitive information, conduct unauthorized activities, or disrupt network operations from within. In IoT ecosystems, where devices often operate autonomously and interact with minimal human oversight, such attacks can spread quickly and unnoticed, posing severe security challenges to both privacy and network integrity.
- Selective Message Forwarding: Selective message forwarding, often referred to as a "gray hole" attack, is a type of network attack specific to multi-hop networks like those in IoT environments. In this attack, a compromised node selectively forwards messages instead of dropping all packets, which is typical in a "black hole" attack. The malicious node may choose to drop packets based on certain criteria, such as the origin or content of the message, which can disrupt network communication selectively. This can lead to incomplete or unreliable data being transmitted across the network, which can be particularly damaging in IoT applications where the accuracy and timeliness of data are critical. For example, in a smart home scenario, sensors might fail to report important events like security breaches or system failures, leading to potential safety issues. Detecting and mitigating this type of attack involves enhanced network monitoring and security measures to identify and manage malicious nodes effectively.

Chapter 3

Related Work and Thesis Approach

In this section, the current research landscape on Trust-based Intrusion Detection Systems in Internet of Things (IoT) networks will be delved into. The goal is to outline the key themes in this area, offering an overview of recent work. Through this exploration, the differentiation of our study will be sought by highlighting the methods and perspectives employed. The context of our research within existing knowledge is established, and our unique approaches to addressing trust management challenges in IoT networks are highlighted.

3.1 Trust-Based IDS on distributed systems

In [30], the authors explore the implementation of a trust-based Intrusion Detection System (IDS) designed for IoT networks with an emphasis on distributed systems. These systems are tailored for devices that have limited processing power and energy resources. The authors focus on securing IoT devices against common network-based attacks such as denial of service, a significant threat given the interconnected nature of these devices.

The research uses a healthcare IoT application to demonstrate the network architecture, highlighting the practicality of their approach. In this architecture, IoT devices operate in an ad-hoc network, transmitting patient data through multiple devices to a border router. This router then sends the data onward to external servers via the Internet. The authors argue for the need for security solutions that are both lightweight and energy-efficient to protect these devices without draining their limited resources excessively.

The proposed intrusion detection mechanism is distributed, which enables IoT devices to efficiently build and manage trust regarding their neighbors.

This system uses Subjective Logic [15] as a framework for trust management, allowing for the dynamic updating of trust based on device interactions.

The trust evaluator component allows devices to monitor the behavior of their neighbors and adjust trust values accordingly. Trust is quantified in opinion triangles that include belief (trust), disbelief (distrust), and uncertainty. These values are adjusted based on direct observations and interactions, where positive and negative experiences directly affect the levels of trust or distrust.

Direct monitoring is crucial for trust adjustment. Devices perform specific checks such as forwarding checks to observe data packet handling, ranking checks to verify node ranks, and version number checks to ensure protocol compliance. These checks help identify potential malicious behaviors and adjust trust metrics based on observed actions.

Furthermore, trust values are not kept isolated but are aggregated either by border routers or cluster heads. This aggregation helps form a comprehensive reputation score for each node, facilitating the identification and isolation of consistently malicious nodes.

To manage trust within the network, the paper introduces three distinct algorithms:

- Neighbor Based Trust Dissemination (NBTD) operates on a centralized model where the border router actively calculates and manages the trust values across the network.
- Clustered Neighbor Based Trust Dissemination (CNTD) takes a distributed approach, with local cluster heads managing trust assessments before aggregating them.
- Tree Based Trust Dissemination (TTD), similar to CNTD, reduces network overhead by limiting monitoring to parent nodes and thus potentially missing malicious activities in leaf nodes without children.

The effectiveness of these algorithms was tested through simulations involving a network of 1000 nodes. The findings indicated that NBTD and CNTD were particularly effective in detecting intruders, more so than TTD, which struggles with nodes that do not have children. However, issues such as false positives and negatives were noted, especially in scenarios involving malicious cluster heads. TTD was shown to have the lowest network load due to

its limited monitoring scope, whereas NBTD recorded the highest due to its centralized data handling.

In [31] a different approach based on Fuzzy Logic is employed. Fuzzy logic is a fundamental component of the TRM-IoT (Trust Management Model Based on Fuzzy Reputation for Internet of Things), adept at managing the inherent ambiguities and incomplete data that characterize IoT networks. This model champions a decentralized approach in which sensors or sensor-embedded devices directly communicate and mutually assess trustworthiness based on observed behaviors. Diverging from the constraints of traditional binary logic, which restricts values to true or false, fuzzy logic allows for a continuum of values between 0 and 1. This flexibility enables a more detailed and dynamic representation of trust and reputation, making it highly suitable for the complex and evolving nature of IoT environments.

In the TRM-IoT model, several trust evaluation metrics are employed to gauge and manage the trust and reputation of nodes within the network. One of the primary metrics used is the End-to-End Packet Forwarding Ratio (EPFR), which measures the ratio of packets successfully received by destination nodes relative to those sent by source nodes. This metric is crucial as it reflects a node's reliability in transmitting data across the network. Another significant metric is the Average Energy Consumption (AEC), which tracks the energy expended by nodes for sending, receiving, and processing data. Given the limited power resources typical in IoT setups, AEC is a vital indicator of a node's efficiency and sustainability. Additionally, the Packet Delivery Ratio (PDR) is used to assess the proportion of packets successfully delivered compared to the total sent, highlighting the impact of node behavior on network reliability, especially in scenarios involving packet loss due to intentional dropping or node misbehavior.

The benefits of employing fuzzy logic in the TRM-IoT model are manifold. It provides robust tolerance to imprecise and noisy data, which is essential in the variable and heterogeneous environments characteristic of IoT. The flexibility inherent in fuzzy logic allows for adaptive and responsive metric definitions and interpretations, enhancing the network's ability to manage trust and reputation under changing conditions. Furthermore, the granularity afforded by fuzzy logic facilitates detailed distinctions in trust levels, empowering the network to make informed, security-enhancing decisions regarding node interactions and data transmissions. This dual-layer evaluation system, integrating both local and global trust assessments, ensures

comprehensive coverage of trustworthiness across the IoT network, reflecting both direct and community-based reputations.

3.2 Machine Learning Based Trust Computational Model

In contrast to the methods used in the aforementioned work, in [32] the authors address the complexities of establishing robust trust mechanisms in IoT systems through a novel computational model that integrates machine learning techniques.

Traditional trust management systems often fall short in addressing the dynamic and heterogeneous nature of IoT environments. To counter these challenges, the authors in [32] propose a robust trust management framework that categorizes trust into three primary metrics: Knowledge, Experience, and Reputation. Each of these metrics encompasses specific trust attributes which are crucial for evaluating the trustworthiness of entities within the network: knowledge focuses on the historical data and direct observations of interactions, providing a baseline of trust based on past behavior, experience derives from direct feedback and past interactions, emphasizing the outcomes and reliability of recent engagements and reputation aggregates community feedback and is indicative of the broader community's perception of an entity's reliability over time.

The paper also discusses how trust is collected and aggregated across different nodes in the network. Trust information is gathered from individual nodes based on their interactions and communications with other nodes, which is then aggregated to form a comprehensive view of the network's trust landscape. This aggregation process leverages the distributed nature of IoT networks, ensuring that trust assessments are both scalable and reflective of the network's dynamic interactions.

Central to their solution is a sophisticated computational model that leverages machine learning to process and analyze trust-related data effectively. The model employs feature extraction techniques to identify relevant trust indicators from raw data, which are then utilized to train the model using two main types of machine learning algorithms: clustering algorithms and classification algorithms. Clustering Algorithms are unsupervised learning methods, such as K-means clustering [33], that are used in the initial phase to

label data points as trustworthy or untrustworthy based on their feature sets. This step is crucial for segmenting the data into categories that reflect preliminary trust assessments. Classification Algorithms are supervised learning methods, including Support Vector Machines (SVM), that are employed in the subsequent phase to predict the trustworthiness of new interactions. This predictive model, trained on the labeled data produced by the clustering process, enables it to assess new data as it flows through IoT systems, thereby forming the final trust score.

The formation of the final trust score is a critical outcome of the machine learning process. It integrates the results from both clustering and classification algorithms to provide a nuanced trust score that reflects both historical data and predictive insights. This score is used to make informed decisions about the reliability of interactions within IoT systems.

To validate the effectiveness of their proposed model, the authors conducted a series of simulations using NS-3[34] and the SIGCOMM-2009[35] dataset, which serves as a proxy for real-world IoT interactions. These simulations test the model's ability to accurately identify and classify trustworthy interactions within a controlled environment.

3.3 Thesis Approach

3.3.1 Requirements

The security solution that is going to be proposed, designed and implemented in this thesis extends the work that has been carried out within the context of the Horizon 2020 european research project IntellIoT (Intelligent, distributed, human-centered and trustworthy IoT environments - https://intelliot.eu/). As such, it borrows the requirements that were specified for a number of real-world IoT deployments and aims primarily to fulfill them. In this subsection, the operating environment that these requirements pose is going to be described and a comparative analysis with the related work that has been presented in the previous subsections is going to be performed to justify our design decisions and highlight why previously available solutions could not be effectively deployed in this case, leading to the development of our solution. While our system is modeled on the requirements of specific use cases, we believe that these use cases are generic enough in the IoT domain that our proposed solution has a broader applicability.

Operating Environment

The primary use case for the considered IoT deployment stems from the agricultural domain. In a large geographic area, a significant number of IoT nodes is spread. These nodes are mostly comprised of simple devices that act either as sensors (humidity, wind, temperature, beacons etc) or actuators (controllers of watering systems, access controllers etc). All nodes are interconnected through a centralised 5G system, however it is assumed that central network coverage may not be available for the whole area that is covered and Ad-Hoc communications may be required to be formed among nodes in order for the most remote ones to reach faraway nodes (or central systems). Nodes are all expected to be stationary or at least with very low mobility.

The IoT nodes considered are heterogeneous. While some of the nodes handling more complex sensing functions (e.g. weather sensing) may have significant computing/memory resources, the majority of nodes is expected to be very low end, typically having an embedded microcontroller that handles most of the computing functions. Also, most of the nodes are expected to be battery operated and it is generally assumed that their energy resources may be replenished sporadically. Nodes are expected to feature commercial grade hardware and therefore malfunctions and failures are expected to be non-negligible. Their key property is their low-cost.

Because of the aforementioned functionalities, the IoT nodes considered have a somewhat predetermined and predictable communication pattern. Typically, besides network operations, (e.g. connection establishments), most of the relayed traffic is periodic and well-defined. For example, a sensor node will report one or more measurements every minute. This behavior is known apparently to system operators at IoT deployment time.

Although normal-operation communication patterns may be well-defined and known a priori, the nature of the specific IoT deployment introduces a large number of uncertainties. Nodes may fail unexpectedly due to environmental issues (e.g. water drainage as a result of heavy rain), malfunctions in their hardware and energy resources depletion. Additionally, although the communications are mainly centralised, they are not considered especially reliable and failures can be often. As the sensor/actuator information can be important, reliability is handled at the higher levels of the network stack and therefore protocols that force retransmissions may be in place (UDP connections are not employed, only TCP - reliability may also be handled at the

application layer).

From the aforementioned, it becomes clear that a lot of issues may be temporal in nature and not permanent. For example, due to an unforeseen event such as a thunder, wireless communications may be disrupted for a small period of time and be reestablished shortly. Additionally, due to the low cost of the deployment and above all its large area spread, it is considered that the participating nodes cannot be physically secured (i.e. prevent an unauthorised entity from accessing the devices). While devices are only added to the IoT network in a secure manner from authorised personnel that oversees their initial deployment, it is impossible to assume that these devices will not be tampered through physical or other external malicious access.

Functional and Non-Functional Requirements

According to the description of the operating environment of the targeted applications, as well as additional requirements set by the H2020 IntellIoT Agricultural Use Case (https://intelliot.eu/wp-content/uploads/2022/09/Deliverable-D2.3-High-level-architecture-1.pdf - https://intelliot.eu/wp-content/uploads/2022/10/D2.6-High-level-architecture-final-version.pdf) the following functional requirements for the IDS system can be extracted:

1. the IDS solutions must be able to run on restricted, low-end hardware. As it is possible to have a multitude of devices to operate on, the IDS should be easy to be deployed in different underlying hardware / OS combinations (use of a high level language implementation or a containerised version is welcome)

2. the IDS operates only on the detection side issuing alarms on other systems such as attack mitigation systems, security assurance platforms and human security operators

3. the IDS has to be installed independently on every node and therefore the solution is distributed in nature and not centralised for the IoT deployment. Each IoT node with the IDS installed should be able to compute potential threats from other nodes and raise alarms independently. Other tools which may be centralised, are responsible to evaluate these alarms and take mitigation actions (or not).

4. the IDS needs to consider as threat not only cyberphysical attacks but also malfunctions or erroneous behaviors that deviate from the described operating conditions. These are going to be described as the considered threat model of the IDS.

5. the IDS needs to consider the temporal nature of misbehavior of nodes and provide mechanisms to restore a positive status of a previously recognised threat

3.3.2 Requirements Analysis and Comparison with Available IDS Systems

Firstly, IDS solutions are required to operate efficiently on restricted, low-end hardware, which is crucial for IoT deployments consisting of numerous devices with varying hardware and operating system configurations. The IDS must therefore be designed with low computational overhead and minimal memory requirements to ensure compatibility with low-end hardware. This can be achieved using lightweight algorithms and data structures to reduce resource consumption. Traditional IDS systems often necessitate substantial computational resources, rendering them unsuitable for low-end IoT devices. For instance, machine learning-based trust computational models mentioned above, leverage complex algorithms that may not be feasible on resource-constrained hardware.

Additionally, the IDS must focus solely on detection, issuing alarms to other systems such as attack mitigation systems, security assurance platforms, and human security operators. This approach ensures a clear separation between detection and mitigation, conserving resources. Traditional IDS systems that combine detection with mitigation strategies within the same system can lead to higher resource usage. By offloading mitigation to specialized systems, the process is streamlined and resource-efficient.

Another fundamental requirement for the IDS in IoT networks is that it must be implemented independently on every node, making the solution inherently distributed rather than centralized. This is crucial because communication between nodes can be unreliable, and the system may often rely on intermediate nodes in ad hoc network configurations to relay information, which are not always guaranteed to be secure. Thus, each IoT node

with the IDS installed must be capable of autonomously computing potential threats from other nodes and initiating alarms independently. This fully distributed approach ensures that each node independently analyzes traffic and computes trust scores, reducing the risk of a single point of failure and enhancing scalability across extensive networks. Unlike centralized models like the Neighbor Based Trust Dissemination (NB-TD), which aggregate data at a central point leading to increased network load and potential bottlenecks, a distributed IDS minimizes latency and improves resilience, critical for maintaining robust security under dynamic network conditions.

The IDS should also consider threats beyond just cyber-physical attacks, including malfunctions or erroneous behaviors that deviate from described operating conditions. The proposed IDS solution includes anomaly detection mechanisms that account for a wide range of potential issues, including both malicious attacks and operational anomalies, such as unexpected node behavior and hardware malfunctions. Many existing systems focus primarily on detecting known cyber-attacks and may not fully address non-malicious anomalies. For example, TRM-IoT[31] emphasizes trust based on fuzzy logic but may not adequately cover non-malicious anomalies. The system expands the scope to cover both cyber-physical threats and operational deviations.

Furthermore, the IDS must account for the temporal nature of node misbehavior and provide mechanisms to restore a positive status for previously recognized threats. Dynamic trust management can be employed, where trust scores increase over time if nodes demonstrate consistent good behavior after a detected anomaly. This helps maintain a balanced and fair trust evaluation. Traditional trust models that use static thresholds or predefined rules may not account for the temporal dynamics of node behavior. The approach ensures that nodes have the opportunity to regain trust, fostering a more adaptive and resilient network.

3.3.3 Threat Model

This model classifies a threat as any occurrence that disrupts the error-free operation of a network node, which includes both malicious attacks, as detailed in Chapter 2, and any operational anomalies that could signify a node malfunction, in the framework that the concept of the IDS is designed to address.

Attacks

In the thesis approach, TCP packets are utilized to monitor and detect potential security threats and malfunctions within the transport layer of network communications. The fundamental concept of this strategy is to identify disruptions and anomalies that are indicative of attacks. By focusing on the transport layer, where TCP operates, the characteristics of each communication session are captured and analyzed. This allows for the effective pinpointing of behaviors that deviate from established norms, potentially signaling the presence of threats or malicious activities aimed at compromising the network's integrity and functionality. Through this method, irregularities that could affect network performance and security are promptly addressed, maintaining the robustness and reliability of the system, particularly in environments where secure and dependable communication is critical. Thus, the symptoms of attacks are simulated, and abnormalities are detected that directly indicate an attack or suggest its presence. The attacks simulated include flooding attacks, scanning attacks, specific flag combinations, and DoS attacks.

System Errors

Except from anomalies that occur from attacks, system errors can also cause system damage. The following list outlines the types of system errors included in the threat model that the detection system can identify, enhancing its utility by helping identify and address inefficiencies and issues by hardware or software malfunctions.

- Node Restarts or Unplanned Reboots
  Unexpected node restarts, potentially caused by power failures or system crashes, can lead to significant time gaps between packet arrivals. This anomaly could signal a node's attempt to re-establish network sessions post-restart, characterized by an increase in SYN packets.
- Overloaded or Congested Nodes
  Nodes experiencing overload or congestion typically exhibit patterns such as consistently large packet sizes or diminished throughput.
- Resource Exhaustion
  Nodes affected by resource exhaustion might show slowed processing capabilities, leading to increased time gaps between packets and reduced throughput.

- Physical Network Issues
  Indirect indicators such as fluctuations in packet timings and data length might suggest physical network issues like cable faults or environmental impacts on network components.
- Software Bugs
  mimic unusual network behavior caused by software bugs by transmitting packets with irregular TCP flag combinations.

3.3.4 Proposal

According to the aforementioned, in this thesis, a distributed trust-based Intrusion Detection System is proposed. This system utilizes network metrics related to TCP packets, such as packet sizes, transmission frequency, and other packet information such as flags and window size, to dynamically establish thresholds for identifying anomalies. The system is structured to operate on a decentralized basis, with each node within the network autonomously evaluating the communications and interactions of its peers to assess their trustworthiness.

Trust scores for each node are derived by continuously monitoring key network metrics such as packet transfer rates and anomalies in data flow. Behaviors aligning with expected and secure network operations positively influence trust scores, while those indicative of potential security breaches result in deductions. A feedback mechanism is also included in the approach. This system continuously refines its trust assessments by incorporating feedback from previous detections and trust evaluations.

Designed for low-cost devices, this IDS approach manages to leverage distributed monitoring and trust-based metrics to dynamically respond to a broad spectrum of security challenges. The system's purpose is to operate efficiently within the limited resources available on low-cost IoT devices, addressing performance issues and resource consumption problems.

Chapter 4

Implementation

This chapter details the code's functional requirements, architecture, and implementation, providing a comprehensive overview of its design and operational specifics. It includes explanations of the core algorithms, data structures used, and the methodologies applied to ensure efficient and reliable performance.

4.1 Functionality

The primary objective of this system is to cultivate a trust score among nodes within IoT networks, enabling the identification of compromised or malfunctioning nodes through trust metrics. To achieve this, a comprehensive pipeline has been developed, consisting of the Monitoring Tool, the Safe Behavior Evaluation Tool, and the Detection and Trust Evaluation Tool, which are described later. This framework is applied to each node-to-node communication, collecting trust scores for each node, thereby making the system distributed.

This system operates through a two-phase mode approach, the "error-free mode" and the "attack mode". During the first mode, it is presumed that the nodes are operating within a safe period, free from attacks or malfunctions. Throughout this phase, communication is closely monitored to profile and establish a baseline of the nodes' standard operational behavior. In this phase, the Monitoring Tool and the Safe Behavior Evaluation Tool are utilized. In the second phase, nodes operate in a real-world unsupervised environment, facing potential threats of attack. Utilizing metrics from the safe period of operation as a baseline, the network flows are compared against these benchmarks, employing specific metrics to identify anomalies within the flow. This process enables the detection of issues stemming from either

attacks or system errors. Simultaneously, the trust score is dynamically adjusted, decreasing exponentially in response to repeated deviations from expected behavior to accurately address ongoing attacks or malfunctions, and increasing linearly during periods of acceptable operation to gradually reestablish trust. As in real life, trust is built at a slow rate and is lost far more sharply. In this phase, the Monitoring Tool and the Detection and Trust Evaluation Tool are utilized.

4.1.1 Assumptions

To guide the implementation, the following assumptions were established:

- Focused exclusively on TCP packets: TCP is the protocol of choice for the IoT application due to its robustness and security capabilities. It reliably orchestrates the delivery and integrity of data packets, even in complex networks, which is vital for IoT systems where accuracy is non-negotiable. The protocol's connection-oriented nature ensures uninterrupted and orderly data exchange, crucial for continuous IoT operations. TCP's flow and congestion control mechanisms intelligently adjust the data flow to prevent receiver overload and maintain network equilibrium, respectively—key for the varied capacities of IoT devices. Additionally, exclusively TCP headers are utilized instead of payloads. TCP headers are processed quicker than payloads due to their smaller size and standardized format, enabling faster traffic analysis without the need for deep packet inspection. This is particularly beneficial in encrypted networks where payload data is inaccessible without decryption keys, but TCP headers remain visible and contain essential information such as source and destination ports, sequence numbers, and flags. This approach also alleviates privacy concerns since it avoids accessing potentially sensitive payload data, crucial for compliance with privacy regulations. Additionally, the consistent structure of TCP headers across different protocols facilitates uniform monitoring of various traffic types within IoT networks. Basic network anomalies and attacks can often be detected through header analysis alone, avoiding the high processing overhead associated with full payload inspection.
- Known IP addresses for nodes and established communication links between nodes: With fixed IP addresses and predefined communication links, the system is better positioned to monitor expected traffic flows and quickly pinpoint any unusual or unauthorized activities.

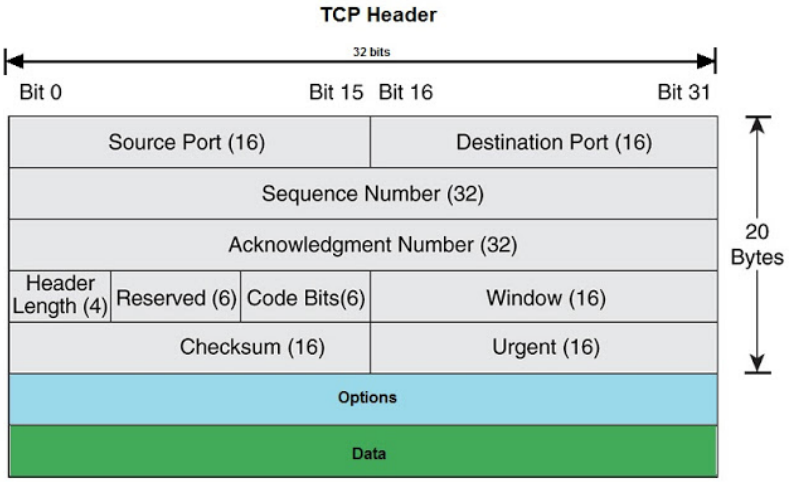

FIGURE 4.1: TCP Header

- Characterization of node behavior within defined expectations: The system effectively establishes a baseline of typical node behavior.
- Error-free operation for all nodes during the initial phase: During the initial phase, it is guaranteed that no malfunctions or attacks occur.

4.2 Architecture

The components of the system are the following:

- **Monitoring Tool**: The Monitoring Tool is designed to observe and collect network packets between the nodes.
- **Safe Behavior Evaluation Tool**: The Safe Behavior Evaluation Tool is designed to elucidate the standard operational characteristics of network traffic.
- **Detection and Trust Evaluation Tool**: The detection and trust evaluation tool analyzes traffic to identify anomalies and evaluate trust scores.

Additionally, a configuration file is set up, including parameters that can be adjusted based on the system's behavior and specific requirements. This flexibility allows the system to adapt to different conditions and operational needs, ensuring optimal performance and responsiveness to various scenarios.

As mentioned above, it is assumed that the system functions in two distinct phases: the first phase involves evaluating the safe, expected behavior of nodes (error-free/safe mode), and in the second phase nodes operate in a

period that could potentially include attacks, anomalies or system errors (attack mode). Thus, two pipeline scripts are utilized to automate the sequential executions of the tool for each phase as follows:

During the first phase, the Monitoring Tool and the Safe Behavior Evaluation Tool are employed to establish a baseline of expected behavior by profiling and extracting metrics indicative of error-free operations without any anomalies. In this stage, the Monitoring Tool operates in "safe mode", focusing specifically on capturing and analyzing typical, undisturbed node activity. The Monitoring Tool observes network traffic for each communication between the nodes and stores it into .txt files, facilitating efficient analysis of the data stream. The files are named in the format safe_data_xy.txt, where 'x' and 'y' represent the node IDs (e.g., 1, 2, 3), indicating the data flow from node x to node y. Then, the Safe Behavior Evaluation Tool is utilized to establish a baseline profile for each communication. The metrics outputted from this tool will be used later as the expected baseline behavior for the nodes. The output is a set of CSV files named as profile_metrics_xy.csv, where 'x' and 'y' represent the node IDs, similar to the naming convention used before. In this phase, the Monitoring Tool and the Safe Behavior Evaluation Tool are used sequentially: the Monitoring Tool runs first to capture network traffic, and once it completes, the Safe Behavior Evaluation Tool starts to establish the baseline profile.

During the second phase, both the Monitoring Tool and the Detection and Trust Evaluation Tool are activated and run in parallel. The Monitoring Tool switches to "attack mode", closely monitoring communications and generating files that document these interactions, with name attack_data_xy_z.txt, where 'z' represents the file sequence number. These files are produced in packets of 1000, reflecting the continuous nature of the network traffic. For example, starting from 0 packets, the first 1000 packets are in the first file, the next 1000 packets in the second file, and so on. Concurrently, the Detection and Trust Evaluation Tool remains aware, continuously scanning for new files produced by the Monitoring Tool to analyze them for potential threats or anomalies and for trust evaluation. This integrated operation ensures a dynamic response to attacks, enhancing the system's effectiveness in threat detection and trust evaluation.

The system is implemented in Python 3.10.12 and utilizing the libraries dpkt[36] library and Pcapy-NG[37]. Dpkt is a python module for fast, simple packet

creation / parsing, with definitions for the basic TCP/IP protocols. Pcapy-NG is a Python extension module that enables software written in Python to access the routines from the pcap packet capture library.

4.3 Monitoring Tool

The Monitoring Tool is designed to observe and collect network traffic between the nodes. It operates in phases one (safe mode) and two (attack mode), with a slight difference described below.

4.3.1 Packet Monitoring Configuration

The Monitoring Tool is configured to collect network interactions, specifically targeting TCP packets, using Pcapy-NG. This setup is optimized for real-time monitoring, ensuring efficient and timely analysis of live network traffic. The configuration parameters, including interface selection, buffer sizes, and capture filters, are fine-tuned to handle high-speed data flows without packet loss.

4.3.2 Packet Decoding Process

The Monitoring Tool is engineered to collect node interactions, specifically targeting TCP packets, using Pcapy-NG. This setup is fine-tuned for real-time monitoring, facilitating efficient and timely analysis of live network traffic. The primary functionality of this tool lies in its robust packet capture and decoding algorithm. The script uses the Pcapy-NG library to capture live network packets from a specified interface. The monitor() function continuously captures packets and processes each packet using the pkt_decode() function. Within this function, the Ethernet frame is decoded to extract the IP packet. If the packet is an IP packet carrying a TCP segment, the TCP segment is further processed, and relevant information (source IP, destination IP, ports, sequence number, data length, flags, window size, timestamp) is extracted and formatted into a structured line of text. The monitoring tool employs the high-performance packet capturing library Pcpay-NG to achieve efficiency and scalability, a library designed for high-performance packet capturing. The buffering and writing mechanism, with a dedicated thread for file writing, ensures that the tool can handle high-speed network traffic without bottlenecks. This design makes the tool scalable and capable of handling extensive network monitoring tasks.

4.3.3 Packet Buffering and Writing

The script employs a combination of a packet buffer and a write queue to manage the writing of captured packet data to files efficiently. In attack mode, packets are buffered until a batch of 1000 packets is collected, which is then added to the write queue for asynchronous writing to files. In safe mode, each packet is immediately added to the write queue for writing to files. This approach ensures that the monitoring process continues capturing packets without being slowed down by I/O operations. A separate thread is dedicated to writing packet data from the queue to files, continuously running the file_writer() function. This function retrieves items from the write queue, generates filenames based on the mode, source and destination IPs, and file index, and writes the packet data to the corresponding file. The tool is also highly customizable and modular. The separation of packet capture, decoding, and writing into distinct functions and threads makes the script easier to maintain and extend. Users can customize the script to capture different types of packets or to adjust the conditions for switching between safe and attack modes. This design ensures that the tool can adapt to various monitoring needs and can be modified to include additional functionalities as required.

4.3.4 Output and Utilization

In both modes, the monitoring tool writes the captured packet data to text files. Each file contains detailed records of the node interactions, including sequence numbers, IP addresses, ports, TCP sequence numbers, data lengths, flags, window sizes, and timestamps.

- Safe Mode Output: Produces files named safe_data_xy_z.txt, which are used to establish a baseline profile of error-free network behavior.
- Attack Mode Output: Produces files named attack_data_xy_z.txt, which are used to analyze and understand the characteristics of network traffic during attack scenarios.
- x: This represents the source node ID in the network traffic. It indicates the ID of the node from which the packet originated.
- y: This represents the destination node ID in the network traffic. It indicates the ID of the node to which the packet is being sent.

- z: This represents the sequence number of the file. Since network traffic is continuous and packets are captured in groups (e.g., packets of 1000), z helps in organizing and indexing these files sequentially. For example, z=1 would be the first file containing the initial set of packets, z=2 would be the next file, and so on.

4.4 Safe Behavior Evaluation Tool

The Safe Behavior Evaluation Tool is designed to elucidate the standard operational characteristics of the nodes establishes a baseline for what constitutes error-free activity within a network. The Safe Behavior Evaluation Tool operates in phase one.

4.4.1 Data Initialization

In the Data Initialization section, the files produced by the Monitoring tool during the "safe mode" are analyzed, creating a pandas DataFrame[38]. This DataFrame contains columns for sequence number, IP addresses, ports, TCP sequence number, data length, flags, window size, and timestamp. The Safe Behavior Evaluation Tool processes this data to establish a baseline of error-free network activity. By meticulously evaluating these parameters, the tool generates key metrics that define the expected behavior of the nodes, which are essential for identifying any deviations indicative of potential anomalies or attacks.

4.4.2 Profile Synthesis

Following the initialization, the methodology advances into Profile Synthesis, where the nodes' operational norms are dissected, analyzed, and consolidated. Specifically, this involves evaluating the average packet size, packet size distributions, timing intervals between packets, and TCP window sizes. For instance, the average time difference between packet transmissions is calculated to establish the regularity of traffic flow, while the standard deviation of these time intervals provides insights into the consistency of data transmission. Similarly, analyzing the average data length and its variability helps in understanding the expected data load and its fluctuations. By quantitatively assessing these parameters, a nuanced understanding of what constitutes error-free behavior within the network's ecosystem is constructed.

In the Profile Synthesis phase, the gathered metrics are combined into a unified profile that provides a clear view of what error-free network activity looks like. This unified profile captures the core aspects of the nodes' regular performance, giving a detailed and understandable overview of the network's condition when operating as expected. Bringing these metrics together into one profile is key, as it highlights the network's baseline characteristics against which any deviations can be measured, enabling effective anomaly detection.

The profile comprises several key metrics, each shedding light on different facets of the node's behavior:

- **Average Time Difference**: This metric represents the average interval between packet transmissions, providing a baseline measure of the traffic flow's regularity.
- **Standard Deviation of Time Difference**: By measuring the variability in the timing of traffic flow, this metric underscores the fluctuations in node activity, offering insights into the consistency of data transmission.
- **Average Data Length**: Reflecting the mean size of packets within the node communication, this metric offers insights into the typical payloads transmitted, shedding light on standard data communication patterns.
- **Standard Deviation of Data Length**: This metric captures the diversity in packet sizes, highlighting any deviations from established norms and potentially signaling irregular activities within the nodes' communication.
- **Total Packets**: Representing the aggregate volume of traffic analyzed, this metric illustrates the scale of data transmission within the nodes' communication, providing a measure of the activity.
- **Unique Streams**: The count of distinct communication paths.
- **TCP Flags Distribution**: Detailing the proportion of different TCP flags observed, this metric offers insights into the various states of connections, from initiation to termination.
- **Average Window Size**: Reflects the mean TCP window size, offering insights into flow control efficiency.

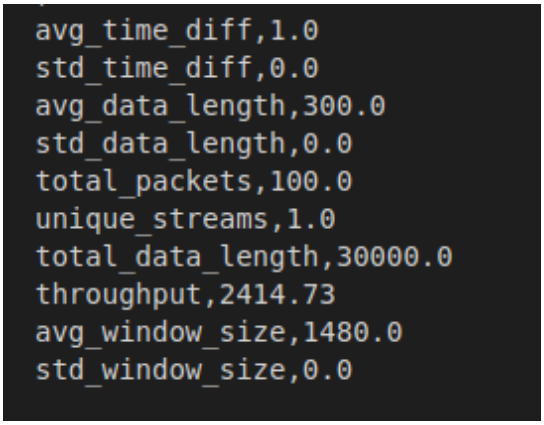

FIGURE 4.2: CSV files

- **Standard Deviation of Window Size**: Measures the variability in TCP window sizes, indicating potential inconsistencies in node's performance.
- **Throughput**: A crucial measure of the performance, throughput quantifies the rate at which data is successfully transmitted across the communication, encapsulating the node's efficiency.

For the following implementation of the Detection and Trust Evaluation Tool, the required metrics are Average Time Difference, Standard Deviation of Time Difference, Average Data Length, and Standard Deviation of Data Length.

4.4.3 Output and Utilization

In the Output and Utilization phase, the synthesized profile is extracted and saved as CSV files named as profile_metrics_xy.csv for further reference in phase two. These CSV files as shown in 4.2 contain detailed metrics that serve as a baseline to compare against subsequent node behavior. By storing these profiles, it becomes possible to identify deviations or anomalies in later analyses, facilitating a more effective detection of network anomalies or potential security threats. This is the last step of phase one.

4.5 Configuration File Set Up

The configuration file for the Intrusion Detection System (IDS) is used in phase two, structured in JSON format to include several key parameters and thresholds essential for monitoring network activities, detecting anomalies, and managing trust scores. These settings are described in 5.2.

block_size	Sets the number of packets processed in each block
starting_trust_score	Establishes the initial trust level for all network nodes at the beginning of the monitoring process
maximum_trust_score	Defines the maximum trust score
trust_decrease_factor: default	Specifies the rate at which the trust score is reduced for standard anomalies
trust_decrease_factor: high	Specifies the rate at which the trust score is reduced for more severe anomalies
trust_increase_factor	Determines the rate at which the trust score is incremented during periods of safe node behavior
std_thresholds: std_packet_length	Defines acceptable variations in packet size
std_thresholds: std_time_diff	Defines acceptable variations in the timing of packet arrivals
anomaly_thresholds: size_based_anomaly: lower	Factor for the minimum acceptable packet size. The actual threshold is calculated by multiplying this factor with the expected packet size.
anomaly_thresholds: size_based_anomaly: upper	Factor for the maximum acceptable packet size. The actual threshold is calculated by multiplying this factor with the expected packet size
anomaly_thresholds: size_based_anomaly: flood	Factor to detect size-based flooding attacks. The actual threshold is calculated by multiplying this factor with the expected packet size
anomaly_thresholds: time_gap_anomaly: lower	Factor for the minimum acceptable timing gaps between packets. The actual threshold is calculated by multiplying this factor with the expected timing gap
anomaly_thresholds: time_gap_anomaly: upper	Factor for the maximum acceptable timing gaps between packets. The actual threshold is calculated by multiplying this factor with the expected timing gap
anomaly_thresholds: time_gap_anomaly: flood	Factor to detect time-based flooding attacks. The actual threshold is calculated by multiplying this factor with the expected timing gap
tcp_port_scanning: ports_num	Specifies the threshold for the number of unique ports accessed within a block that could indicate a port scanning attack

TABLE 4.1: Configuration File

This configuration file is crucial for the IDS's functionality, providing a comprehensive framework for detecting deviations from safe node behavior, dynamically adjusting trust scores, and thereby enhancing the overall security and resilience of the node. The exact usage of these values is detailed in the following section.

4.6 Detection and Trust Evaluation Tool

The Detection and Trust Evaluation Tool analyzes node communication traffic to identify deviations from expected behavior and detects anomalies. It dynamically adjusts trust scores for each node based on the presence of anomalies or consistent error-free behavior, enhancing node-to-node communication security within the IoT network.

4.6.1 Configuration and Initial Setup

The configuration parameters are loaded from the JSON file. These settings define how the tool will operate and respond to various conditions. This initial setup ensures that the tool is customizable and adaptable to different environments and threat landscapes.

4.6.2 Monitoring and Data Collection

The tool actively scans a designated directory for new data files generated by the Monitoring Tool in "attack mode". Each file is processed only once, ensuring no redundancy in the analysis. The data from each file is read and parsed into a structured format, creating a DataFrame for analysis.

4.6.3 Anomaly Detection

The core functionality of the tool lies in its anomaly detection mechanisms. The tool processes node traffic in blocks of packets, which enhances the effectiveness of monitoring. This structured approach allows for immediate detection of certain types of attacks, while others require analysis of traffic over extended intervals to confirm their presence, and are better identified at the end of each block for a more comprehensive analysis.

- Size-Based Anomaly Detection
  The concept behind size-based anomaly detection involves calculating the total packet size over a defined block of traffic. In the configuration file, factors are set to determine how much higher or lower the thresholds will be compared to the expected packet size. Specifically, three thresholds are set: upper, lower, and flooding. These thresholds are applied within each block. If the total packet size in a block exceeds the lower or upper threshold, it indicates a potential anomaly. The flooding threshold is set significantly higher to specifically identify flooding attacks. These thresholds act as a window, allowing for a certain margin of error relative to the expected packet size, thus helping to accurately detect anomalies. Any attack or anomaly related to packet size is evaluated against these thresholds, including flooding attack, overloaded or congested node, physical issues like network drop.

- Time Gap Anomaly Detection
  The time gap anomaly detection follows a similar concept. Time gap refers to the frequency at which packets arrive. Three thresholds are set: lower, upper, and flooding. These thresholds are defined by factors in the configuration file, applied on the average time arrival of every block. The lower threshold identifies unusually slow traffic, while the upper threshold detects unexpectedly high traffic rates. The flooding threshold is set to the lowest value, reflecting the high rate of packet arrivals characteristic of flooding attacks. By comparing the time intervals between packets to these thresholds, the system can detect anomalies in packet arrival frequency, ensuring effective monitoring and response to such irregularities. Any attack or anomaly related to arrival frequency is evaluated against these thresholds, including flooding attack, network bursts, node restarts, network drops.

- Standard Deviation for Packet Size and Arrival Frequency
  A preliminary check, before evaluating time and size anomalies, involves assessing the standard deviation (STD) within a block. If the standard deviations for packet size or arrival frequency exceed the values set in the configuration file, it indicates a significant anomaly, prompting a decrease in the trust score. This check serves as an additional layer of anomaly detection.

  However, it is important to note that this method has its limitations. For instance, a flooding attack with a stable frequency will not trigger

the STD threshold, as the calculated standard deviation could be zero due to the consistent frequency of the attack. Thus, while standard deviation checks are useful, they are employed as an extra measure rather than a standalone detection mechanism.

- Port Scanning Detection
  Port scanning detection is based on the number of different ports accessed, with a limit set in the configuration file. This limit acts as a threshold; exceeding it indicates a potential port scanning activity. For specific types of port scanning, such as FIN or SYN scans, the system also examines the TCP flags of the packets. If the flag pattern matches that of a known scanning technique (e.g., all packets have the FIN flag for a FIN scan), this further confirms the presence of port scanning. Thus, the system combines port count and flag analysis to accurately identify and classify port scanning activities.

- Christmas Tree Packet Detection
  The detection of a Christmas Tree Packet involves checking if the URG, PUSH, and FIN flags are set.

- Software bugs
  For software bugs, it is assumed that the URG and PSH flags are set. Therefore, packets with these two flags are detected as anomalies if both flags are present. This detection mechanism helps identify potential software bugs by flagging irregularities in TCP packet headers that deviate from normal expected behavior.

  It is notable that some attacks are detected over the course of blocks, such as those identified using packet size and time frequency anomalies. Other attacks are detected instantly, particularly those involving flag anomalies.
  Additionally, it is acknowledged that this technique might result in some false positives and negatives. Even though the thresholds are set in the best possible way, there might still be instances where some anomalies exceed these thresholds or legitimate traffic patterns are incorrectly flagged as anomalies. This is discussed in the next chapter.

4.6.4 Trust Score Management

At the end of each block, the trust score is evaluated. If no anomaly or malfunction is detected, the trust score is increased linearly according to the

value set for increasing trust in the configuration file. If an anomaly is detected, an exponential decrease is applied to the trust score. The base for this exponential decrease is the value set in the configuration file for the trust decrease factor, and the exponent is the number of consecutive blocks where this anomaly has been detected. This results in the final decrement, calculated as:

$$weight = trust\ decrease\ factor^{\ consecutive\ blocks\ with\ anomaly}$$

For the trust decrease factor, there are two values specified in the configuration file. The second value is higher than the first, providing the administrator the flexibility to categorize attacks based on their risk levels. This allows for a more nuanced response, where higher-risk attacks trigger a greater decrease in trust. By distinguishing between these two types of attacks, the system can decrease trust at different rates, ensuring a more precise and effective security mechanism. This approach helps in prioritizing and mitigating higher-risk threats more aggressively compared to lower-risk anomalies.

If no anomaly is detected in the next block, the consecutive blocks detected with anomaly parameter is reset to 0, and the trust score increases. The trust score is maintained within a range defined by a minimum value of 0 and a maximum value set in the configuration file, ensuring that it remains within these bounds during all evaluations.

Chapter 5

Evaluation

In this chapter, the IDS solution presented in the previous chapter is evaluated, and various parameter adjustments are made to observe how the system reacts. Furthermore, the cost complexity is calculated.

5.1 Environment Setup

During the setup process, VirtualBox 6.1 was configured to enable communication between two virtual machines (VMs) and the host, forming a trio that replicates a network with three nodes, as shown in 5.1. The VMs run Ubuntu 22.04 with 2048 MB base memory and 30 GB storage.
For the implementation Python 3.10.12 was used, running in a virtual environment on Visual Studio Code.

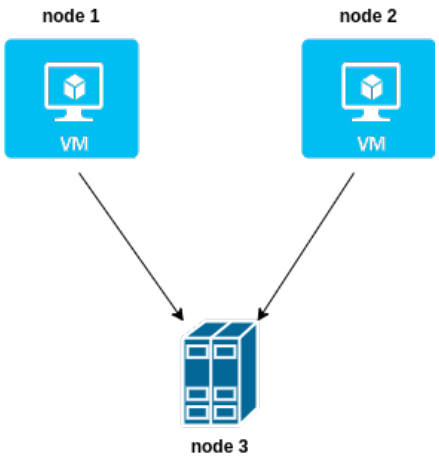

FIGURE 5.1: Setup

5.2 Tools Used

5.2.1 Nping

During the evaluation, Nping [39] was employed to generate network traffic between nodes. Nping is an open source tool for network packet generation, response analysis and response time measurement. Nping allows users to generate network packets of a wide range of protocols, letting them tune virtually any field of the protocol headers. Nping was specifically utilized to simulate TCP communications between nodes, in order to test and validate the system's effectiveness in detecting and responding to various network scenarios.

Nping options used for the simulations are listed below:

–tcp	TCP probe mode
–dest-port	Set destination port(s)
–flags <flag list>	Set TCP flags
–win <size>	Set window size.
–data-length <len>	Include len random bytes as payload
–count <n>	Stop after <n> rounds
–delay <time>	Adjust delay between probes
–rate <rate>	Send num packets per second

TABLE 5.1: Nping options

5.3 Case Study

5.3.1 Safe Node Behaviors

By referring to "safe node behavior," the communication patterns between nodes are specifically addressed. For instance, if a node connects with two others, it establishes distinct communication norms for each connection. Therefore, this node exhibits two separate error-free behaviors, corresponding to its interactions with each of the two nodes. In the thesis' methodology, TCP connections between nodes are established using the Nping tool, which sends TCP packets between specific node pairs. Specifically, the connection setup includes:

- Between Node 1 and Node 3, packets are sent every 5 seconds with a payload size of 1024 bytes, representing this flow as $Flow1 \rightarrow 3$

Average Time Difference	5.0s
STD Time Difference	0.0s
Average Data Length	1024.0 bytes
STD Data Length	0.0s
Average Window Size	1480.0
STD Window Size	0.0s
Total Packets	100
Flags Distribution	S

TABLE 5.2: Safe Behavior Profile for $Flow1 \rightarrow 3$

Average Time Difference	1.0s
STD Time Difference	0.0s
Average Data Length	300.0 bytes
STD Data Length	0.0s
Average Window Size	1480.0
STD Window Size	0.0s
Total Packets	100
Flags Distribution	S

TABLE 5.3: Safe Behavior Profile for $Flow2 \rightarrow 3$

- Between Node 2 and Node 3, packets are transmitted every 1 second, each containing 300 bytes, representing this flow as $Flow2 \rightarrow 3$

As it is mentioned in the assumptions in chapter 3, each node-to-node communication exhibits consistent periodicity and maintains uniform data length. So the metrics for safe node behaviors are the following:

5.3.2 Attack and Anomalies Simulation

As mentioned in Chapter 4, various attacks and anomalies that indicate either an attack or a system error are simulated in the following manner:

- Flooding attacks like SYN Flood, FIN Flood, ACK Flood involves high-rate transmission of SYN packets to overwhelm the target, for example 100 packets/second.
- TCP Port Scanning sequentially sends SYN packets to multiple ports to discover active services, simulating a reconnaissance scan.
- TCP FIN Scan uses transmission of FIN packets to identify open ports, simulating a reconnaissance scan.

- Christmas Tree Packet Attack involves transmission of packets with multiple TCP flags set to probe network responses, simulating a sophisticated reconnaissance scan.
- Node Restarts simulate a node quickly re-establishing connections after a restart by low-rate transmission of SYN packets.
- Node Congestion simulates an overloaded or congested node through high-rate transmission of large data packets.
- Software Bugs mimic unusual node behavior caused by software bugs by transmitting packets with irregular TCP flag combinations, such as URG and PSH.
- Resource Exhaustion effects are simulated by slow, continuous transmission of PSH packets.
- Physical Node Issues are simulated by intermittent bursts of large data packets followed by pauses, mimicking physical node connectivity issues.

5.3.3 Configuration File Setup

The configuration file includes parameters that can be adjusted based on the system's behavior and specific requirements. In the next subsection, the process of choosing these values for testing is detailed.

The **block_size** is set to 100, which defines the number of packets grouped together in a block for analysis. This setting ensures that the system processes and evaluates packets in manageable segments.

The **trust_decrease_factor** section includes two values: a default trust factor of 2 and a high trust factor of 3. This choice allows the system to decrease trust more rapidly in the case of attacks for which the system is known to be more vulnerable.

The **base_trust_increase_factor** is set to 2, providing a base rate for incrementing the trust score during periods without detected anomalies. This factor ensures a steady increase in trust when the system operates normally.

The **std_thresholds** section sets standard deviation thresholds for packet length and time differences. Since packet length and time differences of packet arrival are stable values, if the standard deviation exceeds the established threshold, instability in the network is assumed that might suggest error or

attack, and the trust is decreased. The standard deviation for packet length is set to 50 and for time differences is set to 2.

The **anomaly_thresholds** section defines the bounds for detecting size-based anomalies, time-gap anomalies and port scanning. For size-based and time-gap anomalies three values are set for each threshold: the first defines the upper bound, the second defines the lower bound and the third. These values are used as factors calculated with the expected average time difference and the expected total packet size (from the safe-expected behavior), in order to form the final threshold, over the block.

In the tests, the upper bound for packet size is set at 1.2. This means that for a packet size of 1024 bytes ($Flow1 \rightarrow 3$), the upper bound is 1024 bytes * 1.2 * 100 = 122880 bytes and the lower is 1024 bytes * 0.8 * 100 = 81920 bytes, and for a packet size of 300 bytes ($Flow2 \rightarrow 3$), the upper bound is 300 bytes * 1.2 * 100 = 36000 bytes and the lower is 300 bytes * 0.8 * 100 = 24000 bytes. The third parameter is for flooding attack and is set to 5. So, the threshold for $Flow1 \rightarrow 3$ is set to 1024 bytes * 5 * 100 = 512000 bytes and for $Flow2 \rightarrow 3$ is set to 300 bytes * 5 * 100 = 150000 bytes.

For time-based anomalies, three thresholds are set respectively: the lower threshold is 0.8, the upper threshold is 1.2, and the flooding threshold is 0.1. Since the analysis is conducted via packet aggregation rather than time interval, a high rate of packets would indicate a flooding attack. So, for $Flow1 \rightarrow 3$ the upper threshold is 5 seconds * 1.2 = 6 seconds and the lower threshold is 5 seconds * 0.8 = 4 seconds and for $Flow2 \rightarrow 3$ the upper threshold is 1 second * 1.2 = 1.2 seconds and the lower threshold is 1 seconds * 0.8 = 0.8 seconds. For flooding, the threshold is 1 second * 0.1 = 0.1 seconds.

For detecting port scanning, a **tcp_port_scanning** threshold is established to specify the maximum number of different ports allowed within a block without raising an alarm and decreasing the trust score. This threshold is set to 10.

The **starting_trust_score** is initially set to 5000 and the **maximum_trust_score** is set to 10000.

The above parameters can be seen in 5.2, where a snippet of the configuration file is provided.

```
"block_size": 100,
"starting_trust_score" : 5000,
"maximum_trust_score" : 10000,

"trust_decrease_factor": {
    "default": 2,
    "high": 3
},

"trust_increase_factor" : 2,

"std_thresholds": {
    "std_packet_length": 50,
    "std_time_diff": 2
},

"anomaly_thresholds": {

    "size_based_anomaly": {
        "lower": 0.8,
        "upper": 1.2,
        "flood": 4
    },
    "time_gap_anomaly": {
        "lower": 0.8,
        "upper": 1.2,
        "flood": 0.1
    },

    "tcp_port_scanning": {
        "ports_num": 10
    }
```

FIGURE 5.2: Configuration File

5.3.4 Threshold Establishment

For the packet length standard deviation threshold, the system executed scripts that sent 100 packets with a variance of 100 bytes above and below the central value. This resulted in a calculated standard deviation of 50 bytes.

Similarly, for the time deviation threshold, the scripts ran with packet arrival times varying within a 2-second range from the central value. This resulted in a calculated standard deviation of 1.5 seconds. To provide a wider tolerance window and account for additional variability, the time deviation threshold was set to 2 seconds.

For the anomaly thresholds, the values are chosen based on preference rather than experimental results. These values can be adjusted according to the specific needs of the system.

Furthermore, it is considered that the only anomaly with a high-risk factor set to three is the flooding attack.

5.3.5 Attack and Anomaly Scenarios

In the test scripts, the attacks and anomalies mentioned above are simulated using TCP packets. These simulations are designed to test the resilience and detection capabilities of the nodes under different conditions. The test simulations consist of 3000 packets for $Flow1 \rightarrow 3$ and $Flow2 \rightarrow 3$. The scripts include periods of error-free network flow, during which trust increases, as well as periods where anomalies occur, resulting in a decrease in trust.

It is important to note that:

- In some cases, a false positive or a false negative is simulated, and it is noted accordingly. In the next subsection, false positives and negatives are analyzed in detail.
- The packets are processed in blocks of 100, enabling the system to analyze batches of 100 packets each.
- In the charts, wherever trust remains stable it is because trust is calculated over blocks but represented over packets.
- Since the flow is from node x to node y, the trust score represents the perception that node y has of node x.

For $Flow1 \rightarrow 3$ the script consists of error-free traffic, SYN flood, Christmas tree packet attack, node restarts, overloaded node and two scenarios where high trasnfer rate and high std on packet size is simulated, that could indicate errors. The script is structured as follows:

- Block 1: Error-free traffic
  Expected behavior: Trust score should increase due to the absence of anomalies.
  Packet range: 0-100
- Block 2: Christmas Tree Packet attack
  Expected behavior: Trust score should decrease due to the detection of the attack.
  Packet range: 100-200
- Block 3: Node Restart
  Expected behavior: Trust score should decrease due to the detection of the anomaly.
  Packet range: 200-300
- Block 4-14: SYN Flooding Attack
  Expected behavior: Trust score should decrease due to the detection of the attack.
  Packet range: 300-1400
- Block 15-23: Error-free traffic
  Expected behavior: Trust score should increase due to the absence of anomalies.
  Packet range: 1400-2300
- Block 24: Node Congestion - *False negative*
  Expected behavior: Trust score should increase because the thresholds are not exceeded and the anomaly is not detected.
  Packet range: 2300-2400
- Block 25-27: High Transfer Rate
  Expected behavior: Trust score should decrease due to the detection of the abnormal rate.
  Packet range: 2400-2700
- Block 28-30: Traffic with High STD on packet size
  Expected behavior: Trust score should decrease due to the detection of

higher STD, above the expected threshold.
Packet range: 2700-3000

For $Flow2 \rightarrow 3$ the script consists of error-free traffic, TCP RST flood, TCP FIN scan, TCP SYN port scan and models unexpected node restarts, node congestion, physical node issues, resource exhaustion, software bugs, and burst traffic due to node recovery. The script is structured as follows:

- Blocks 1 and 2: Error-free traffic. Expected behavior: Trust score should increase due to the absence of anomalies.
  Packet range: 0-200
- Block 3: FIN scan attack
  Expected behavior: Trust score should decrease as the system detects the FIN scan.
  Packet range: 200-300
- Blocks 4 and 5: SYN port scanning - *False negative*
  Expected behavior: Trust score should increase as the SYN port scanning is not detected due to the block exchange and the threshold not being exceeded.
  Packet range 300-500
- Blocks 6 and 7: Error-free traffic
  Expected behavior: Trust score should increase due to the absence of anomalies.
  Packet range: 500-700
- Block 8: Network drop (physical issues)
  Expected behavior: Trust score should decrease due to the detection of anomalies.
  Packet range: 700-800
- Blocks 9 and 10: Error-free traffic
  Expected behavior: Trust score should increase due to the absence of anomalies.
  Packet range: 800-1000
- Block 11: Short burst of packets after sleep - *False negative*
  Expected behavior: Trust score should increase because the thresholds are not exceeded.
  Packet range: 1000-1100

- Blocks 12 and 13: Error-free traffic
  Expected behavior: Trust score should increase due to the absence of anomalies.
  Packet range: 1100-1300
- Blocks 14, 15 and 16: RST flood
  Expected behavior: Trust score should decrease as the system detects it.
  Packet range: 1300-1600
- Blocks 17, 18, 19 and 20: Error-free traffic
  Expected behavior: Trust score should increase due to the absence of anomalies.
  Packet range: 1600-2000
- Block 21: Size anomaly and syn port scanning due to security maintenance - *False positive*
  Packet range: 2000-2100 Expected behavior: Trust score should decrease as the system detects it.
- Blocks 22, 23 and 24: Error-free traffic
  Expected behavior: Trust score should increase due to the absence of anomalies.
  Packet range: 2100-2400
- Block 25: Software bugs
  Trust score should decrease as the system detects it.
  Packet range: 2400-2500
- Blocks 26, 27, 28, 29 and 30: Error-free traffic
  Expected behavior: Trust score should increase due to the absence of anomalies.
  Packet range: 2500-3000

The above blocks ranges are converted into packet ranges and the line charts of the trust score plot trust on the y-axis and packets on the x-axis. For some cases, a logarithmic scale with base 10 is utilized to better present the trust score fluctuations that were not well visible with a linear scale. For block size = 100 the trust values for every packet range is shown in 5.3.

The trust score line chart for $Flow1 \rightarrow 3$ is provided in 5.4

The trust score line chart for $Flow2 \rightarrow 3$ is provided in 5.6.

It is noted that the system works as expected in the following manner:

	Block size = 100		
Packets	Flow 1→3	Flow 1→3 (log10)	Flow 2→3
0	5000	3.69897000433602	5000
50	5000	3.69897000433602	5000
100	5002	3.69914368739448	5002
150	5002	3.69914368739448	5002
200	5000	3.69897000433602	5004
250	5000	3.69897000433602	5004
300	4996	3.6986224297021	5002
350	4996	3.6986224297021	5002
400	4969	3.69626899674553	5004
450	4969	3.69626899674553	5004
500	4888	3.6891311972345	5006
550	4888	3.6891311972345	5006
600	4645	3.66698571832966	5008
650	4645	3.66698571832966	5008
700	3916	3.5928426831311	5010
750	3916	3.5928426831311	5010
800	1729	3.23779499327392	5008
850	1729	3.23779499327392	5008
900	0	0	5010
950	0	0	5010
1000	0	0	5012
1050	0	0	5012
1100	0	0	5014
1150	0	0	5014
1200	0	0	5016
1250	0	0	5016
1300	0	0	5018
1350	0	0	5018
1400	0	0	5015
1450	0	0	5015
1500	2	0.301029995663981	5006
1550	2	0.301029995663981	5006
1600	4	0.602059991327962	4979
1650	4	0.602059991327962	4979
1700	6	0.778151250383644	4981
1750	6	0.778151250383644	4981
1800	8	0.903089986991943	4983
1850	8	0.903089986991943	4983
1900	10	1	4985
1950	10	1	4985
2000	12	1.07918124604762	4987
2050	12	1.07918124604762	4987
2100	14	1.14612803567824	4985
2150	14	1.14612803567824	4985
2200	16	1.20411998265592	4987
2250	16	1.20411998265592	4987
2300	18	1.25527250510331	4989
2350	18	1.25527250510331	4989
2400	20	1.30102999566398	4991
2450	20	1.30102999566398	4991
2500	18	1.25527250510331	4989
2550	18	1.25527250510331	4989
2600	14	1.14612803567824	4991
2650	14	1.14612803567824	4991
2700	6	0.778151250383644	4993
2750	6	0.778151250383644	4993
2800	0	0	4995
2850	0	0	4995
2900	0	0	4997
2950	0	0	4997
3000	0	0	4999

FIGURE 5.3: Trust score values for block size = 100 for both flows

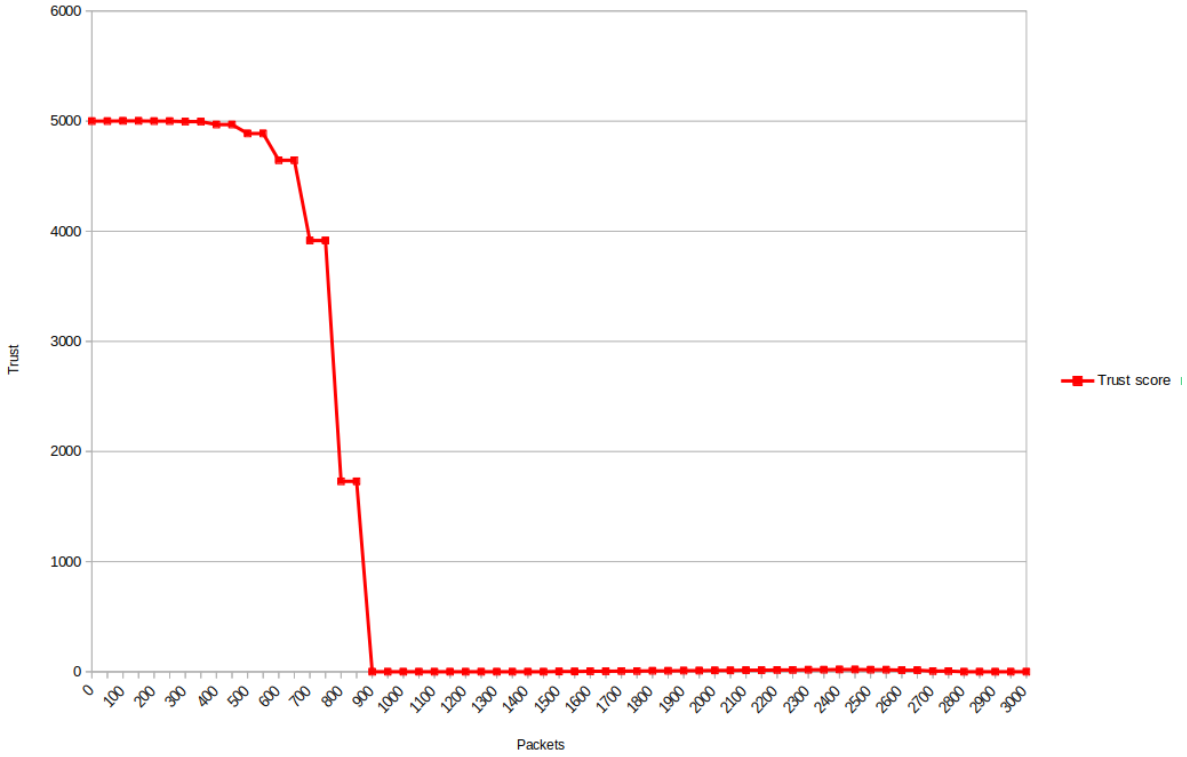

FIGURE 5.4: Trust Score - Perception of Node 1 by Node 3
block size = 100

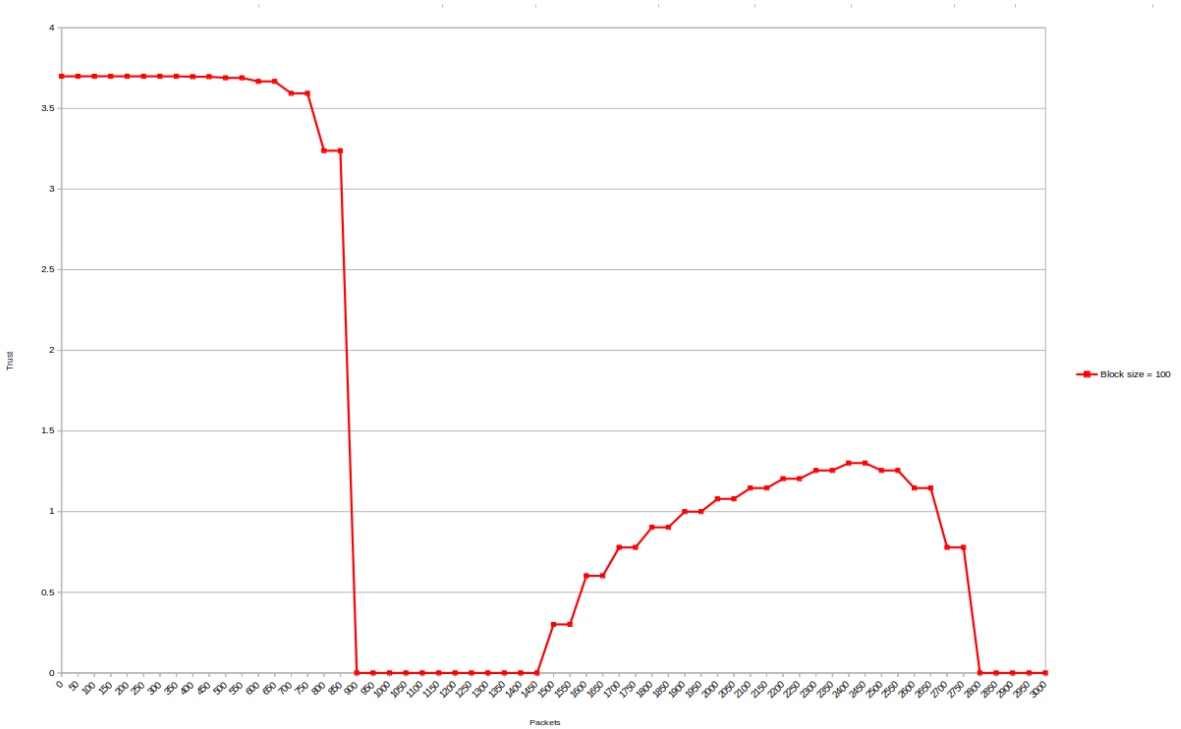

FIGURE 5.5: Trust Score - Perception of Node 1 by Node 3
block size = 100
logarithmic scale (base 10)

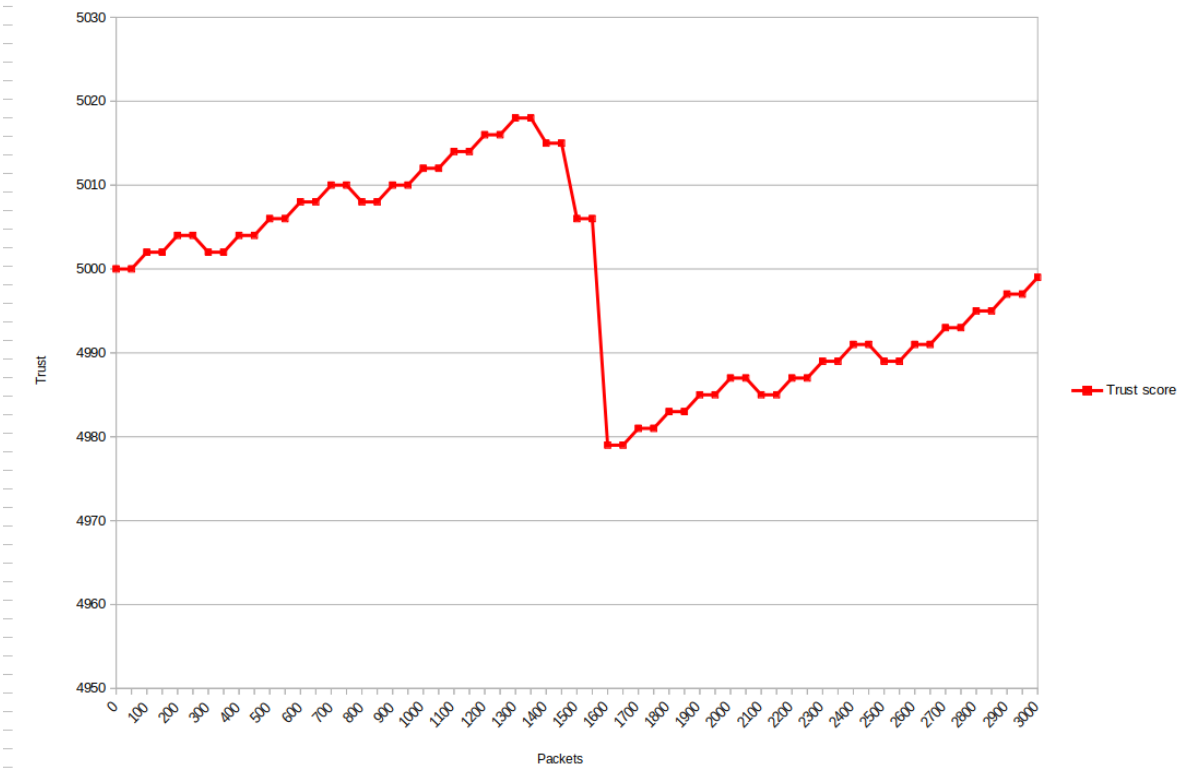

FIGURE 5.6: Trust Score - Perception of Node 2 by Node 3
block size = 100

1. The anomaly detection mechanism effectively distinguishes between different types of anomalies and attacks, adjusting the trust score accordingly.

2. The trust score adjustments align with the established thresholds. When conditions indicate a need for trust reduction, the score is appropriately diminished, while under conditions warranting an increase, the trust score is appropriately enhanced.

3. High-risk attacks, specifically flooding attacks, operate with a high-risk trust score decrease. This behavior is evident in the line charts, where the trust score decreases at a higher rate during flooding attacks.

4. It is observed that the trust score increases in a linear fashion but decreases exponentially. So, the exponential decrement is set well: during periods with consecutive blocks containing anomalies, the trust is decreased at a higher rate with each subsequent block.

5. As resulted from $Flow1 \rightarrow 3$ the trust score decreases rapidly in response to anomalies or attacks but takes significantly longer to recover and increase back to its original level. This indicates a high sensitivity

to detecting potential threats but a slower response in reinstating trust after the threat has subsided. To address this, the combination of trust decrease and increase factors should be adjusted according to the system's needs. A more balanced approach will ensure the trust score accurately reflects the network's state, penalizing anomalies swiftly while effectively rewarding sustained normal behavior.

5.3.6 False Positives - False Negatives

Since the thresholds are set, some of the values may be incorrectly identified as attacks, resulting in false positives. Conversely, actual attacks might not be recognized as such, leading to false negatives.

Concerning false negatives, there are two possibilities. One possibility is that the attack falls within the defined bounds, which means the thresholds set might not be strict enough to detect it. This emphasizes the importance of carefully considering threshold settings in any system of this nature. Another possibility is related to the block size: an attack might be segmented into parts that do not individually exceed the set thresholds, thereby escaping detection. Some examples of the false negatives that are simulated are the following:

For $Flow1 \rightarrow 3$:

- the system encounters an unexpected node congestion, where the network traffic is experiencing high traffic volume, without exceeding any of the predefined thresholds for anomalies (Block 24).

The false negative is shown in 5.7.

For $Flow2 \rightarrow 3$:

- a port scanning simulation where the attack is divided into two parts due to the block size (Blocks 4 and 5)
- short burst of packet after sleep where the time thresholds are not exceeded (Block 11)

The false negatives are shown in 5.8.

Additionally, some potential false positives in the IDS can occur due to legitimate network activities. For instance, legitimate network traffic sometimes exhibits burst behavior, which could be mistaken for a flooding attack. Additionally, network administrators often perform port scans for maintenance

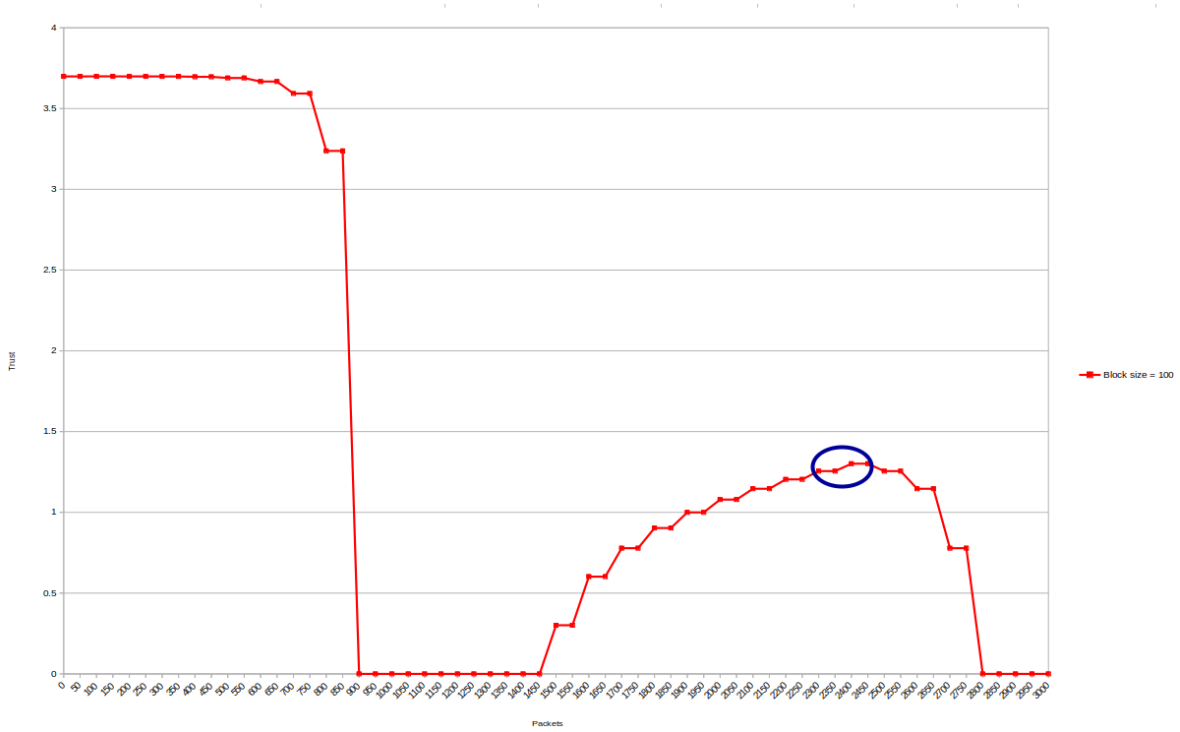

FIGURE 5.7: False Negatives - Perception of Node 1 by Node 3

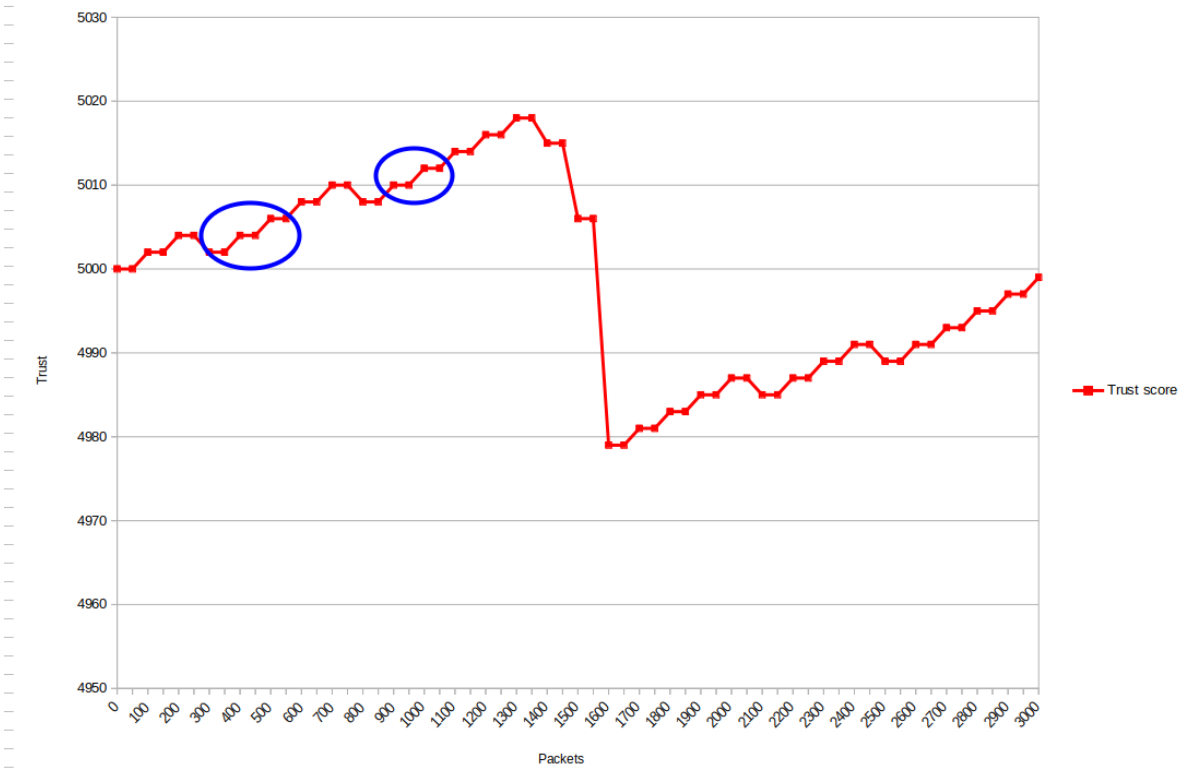

FIGURE 5.8: False Negatives - Perception of Node 2 by Node 3

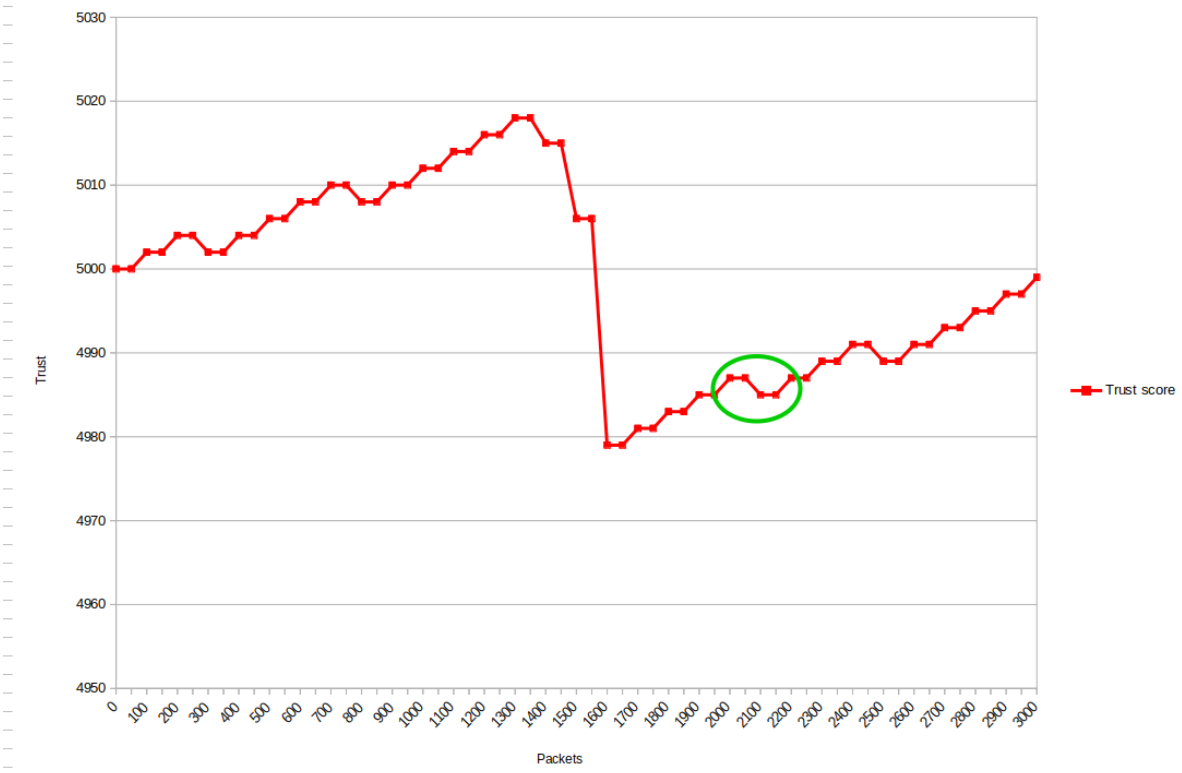

FIGURE 5.9: False Positives - Perception of Node 2 by Node 3

and security checks, which could be falsely flagged as malicious port scanning. The last scenario is implemented in:
$Flow2 \rightarrow 3$:

- port scanning for security purpose (Block 21)

The false positive is shown in 5.9.

5.3.7 Response Time

Referring to response time, it specifically indicates the number of packets that have passed until the alarm is triggered and the trust level decreases. A significant factor influencing the response time is the block size. If the block size is very high, then the response time increases because it takes longer to accumulate enough packets to analyze. Conversely, if the block size is lower, the block finishes more quickly, leading to faster detection and response times. This is analyzed in the next section.

5.4 Modifications in Configuration File

According to the system's specifications and needs, all parameters can be adjusted to achieve better responsiveness. For example:

5.4.1 Block Size

For modifications, the block size is set to 50, 100, 200, and 300 as shown in 5.10, 5.12, 5.11 and 5.13. The analysis was conducted for both flows. A logarithmic scale was utilized in some charts to better visualize the differences between the block sizes.

It is noteworthy that if the block size was set to 1, then the system would analyze and detect the flow on a per-packet basis rather than in batches. This approach would enable instant detection of anomalies such as a Christmas Tree attack based on single-packet criteria. However, it would be inefficient for detecting attacks that require analysis over a period of time, such as those involving subtle patterns or cumulative effects.

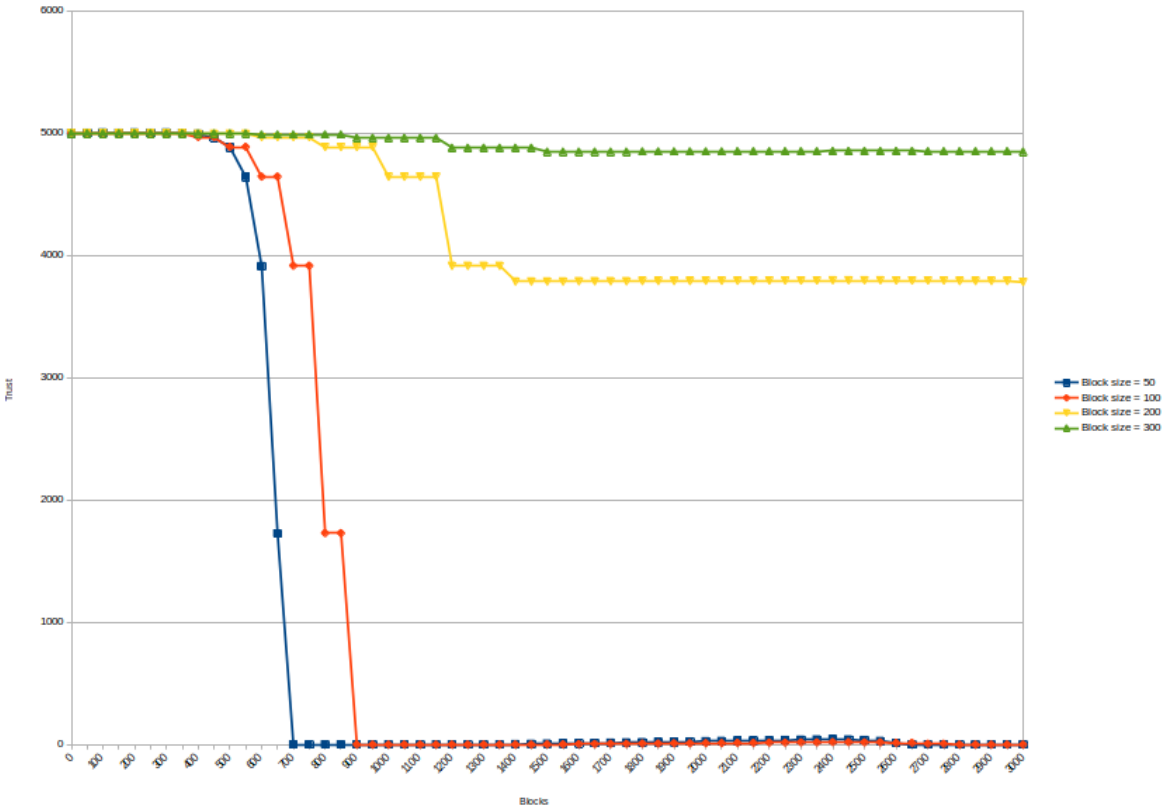

FIGURE 5.10: Trust Score - Perception of Node 1 by Node 3
Block size comparison

Comparing the block sizes, conclusions are:

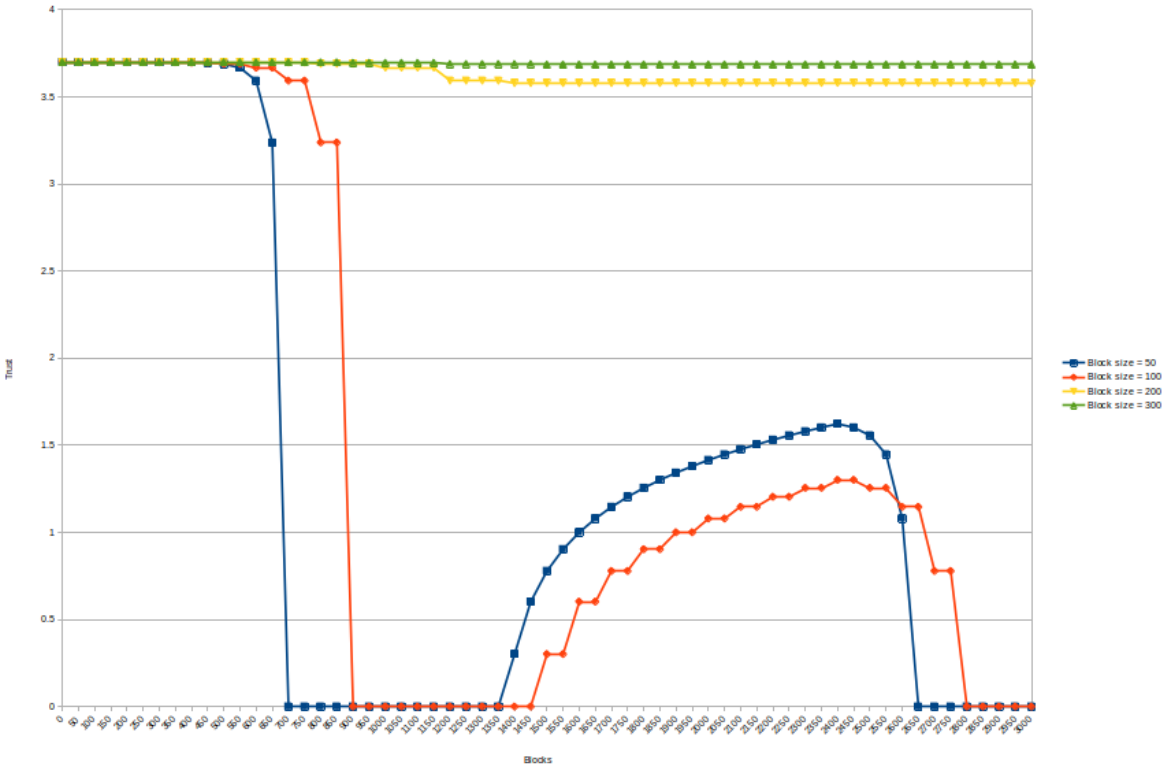

FIGURE 5.11: Trust Score - Perception of Node 1 by Node 3
Block size comparison
logarithmic scale

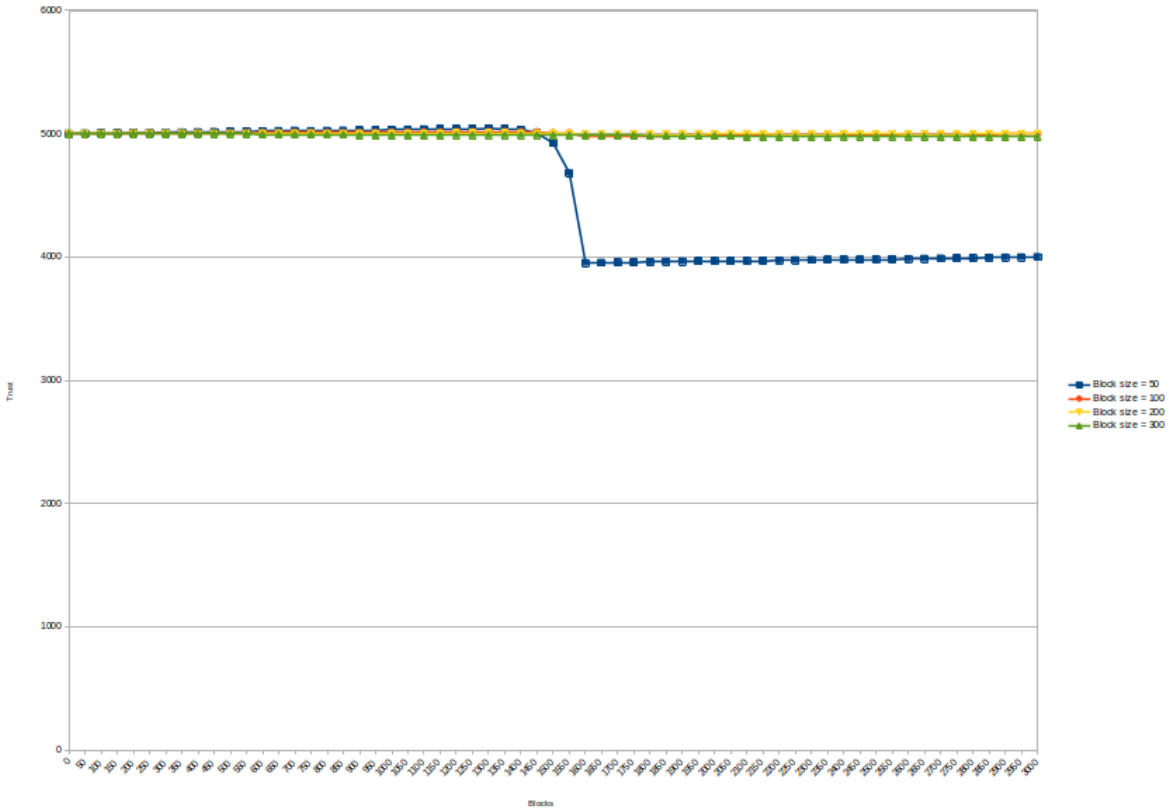

FIGURE 5.12: Trust Score - Perception of Node 2 by Node 3
Block size comparison

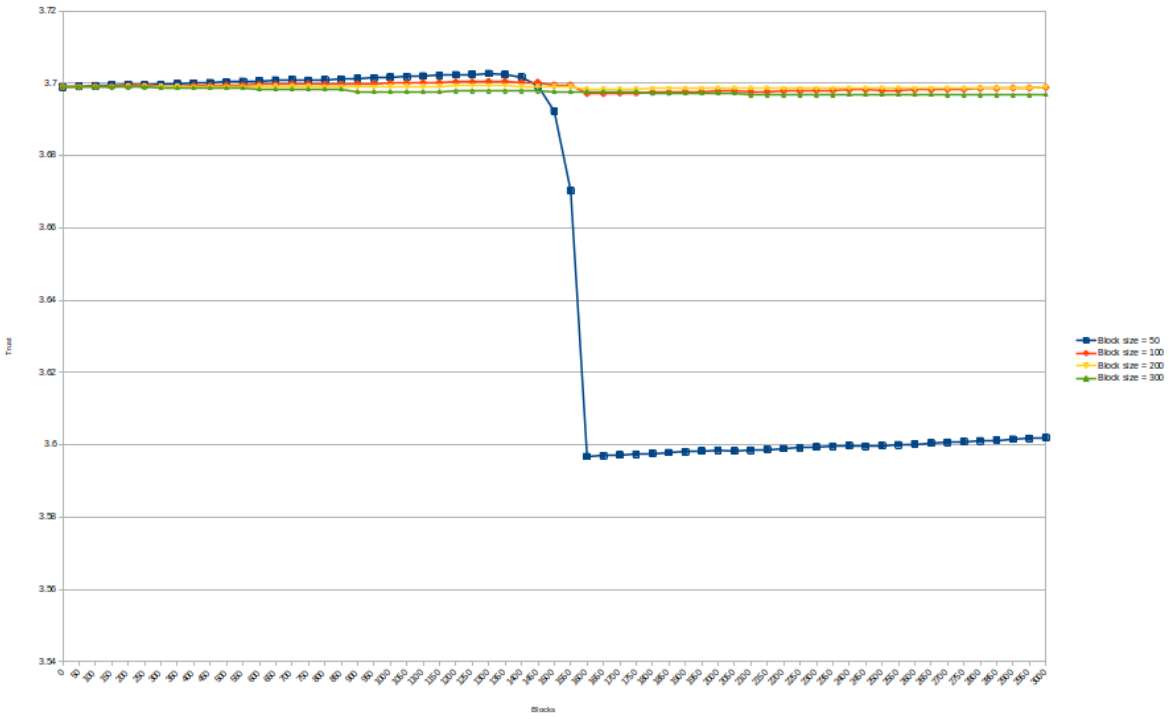

FIGURE 5.13: Trust Score - Perception of Node 2 by Node 3
Block size comparison
logarithmic scale

1. Stability and Sensitivity
    - Block Size 50: Exhibits high sensitivity to anomalies with sharp drops and lower overall stability, indicating it quickly reacts to changes but might be prone to false positives. This can be seen clearly in the charts where the trust score fluctuates significantly.
    - Block Size 100: Balances sensitivity and stability, showing periods of increase, sharp decline, and recovery, indicating effective detection and management of anomalies. This balance is visible in both flows.
    - Block Size 200: Demonstrates moderate sensitivity with more stability compared to smaller blocks. It effectively detects anomalies and recovers trust more gradually, offering a balanced approach. The charts for both flows reflect this moderate approach.
    - Block Size 300: Exhibits slower reactions and larger intervals of stability, indicating it might miss shorter anomalies but provides a more stable trust assessment over longer periods. This slower reaction is apparent in both flows where trust remains stable for longer periods before changing.
2. Anomaly Detection and Trust Recovery
    - Block Size 50: Quickly detects and reacts to anomalies, but the trust recovery is slower, leading to longer periods of low trust. This is shown in the trust score dropping sharply and taking longer to recover in the charts.
    - Block Size 100: Shows effective anomaly detection and recovery, with the ability to bounce back to higher trust levels after addressing issues. Both flows demonstrate this ability to recover trust levels effectively.
    - Block Size 200: Provides a balanced detection and recovery mechanism, suitable for networks requiring consistent monitoring without frequent fluctuations. The gradual recovery in trust scores can be observed in the charts.
    - Block Size 300: Slower in detecting and reacting to anomalies, with prolonged periods of low trust, indicating it might be better for environments where anomalies are less frequent but more severe.

This is evident from the longer stable periods in the trust score before reacting to anomalies.

3. Use Case Suitability

   - Block Size 50: Suitable for environments requiring high sensitivity and quick reactions to any traffic changes, such as highly dynamic networks. This suitability is supported by the quick reaction times in both flows.
   - Block Size 100: Ideal for balanced environments where both anomaly detection and trust recovery are crucial, such as typical corporate networks. Both flows show a good balance of sensitivity and stability.
   - Block Size 200: Suitable for networks needing consistent and balanced monitoring, offering a compromise between sensitivity and stability. The charts reflect this balanced approach.
   - Block Size 300: Best for environments where stability is preferred over sensitivity, such as long-term monitoring of relatively stable networks. This preference for stability is shown in the charts by the prolonged periods of stable trust scores.

4. Final Trust Scores

   - Block Size 50: The final trust score shows a significant impact from detected anomalies and slower recovery, as shown in both charts.
   - Block Size 100: The final trust score suggests effective anomaly detection and good recovery to maintain high trust. This is evident in both flows.
   - Block Size 200: The final trust score shows balanced detection and recovery, with a moderate impact from anomalies. Both flows reflect this moderate impact.
   - Block Size 300: The final trust score indicates a moderate impact from anomalies with slower detection and recovery. The charts for both flows show this slower reaction and prolonged stability.

The comparative analysis of different block sizes for both flows demonstrates that smaller block sizes, such as 50, provide high sensitivity and quick reactions to anomalies, making them suitable for dynamic environments that require immediate responses. However, this high sensitivity comes at the

cost of lower overall stability and a higher likelihood of false positives. On the other hand, larger block sizes, such as 300, offer greater stability and are better suited for environments where long-term monitoring is essential, despite being slower in detecting and reacting to anomalies. Block sizes of 100 and 200 provide a balanced approach, effectively managing both detection and recovery, making them ideal for general use in diverse network environments. This analysis highlights the importance of selecting an appropriate block size based on specific network requirements and the nature of the traffic, balancing the trade-offs between sensitivity, stability, and resource efficiency.

5.4.2 Trust Factors

Adjusting trust factors changes the rate of exponential decrease or increase in the trust score. Higher trust decrease factors will cause the trust score to drop more rapidly during anomalies, while lower factors will result in a slower decrease. Conversely, increasing the trust increase factor will allow the system to recover trust faster after an anomaly is resolved, whereas a lower increase factor will slow down the recovery rate.

5.4.3 Anomaly Thresholds

Modifying these thresholds can make the system more or less sensitive to detect anomalies. Lower thresholds can increase sensitivity, detecting more anomalies but potentially increasing false positives, while higher thresholds can reduce sensitivity, focusing on more severe anomalies.

5.5 Cost Complexity

In this section, the cost complexity is analyzed.

5.5.1 Monitoring Tool

Main components affecting Complexity:

1. **Packet Capture and Decoding**
    - Each packet is processed individually as it is captured. The operations performed per packet include parsing the packet, checking conditions, and potentially appending data to a buffer. Each of

these operations is $\mathcal{O}(1)$, so processing n packets results in $\mathcal{O}(n)$ complexity.

2. **Buffer Management (Attack Mode)**

   - In attack mode, packets are buffered until a threshold is reached, after which they are written to a file. This conditional buffering and periodic dumping contribute negligibly to the overall complexity but can affect performance due to batch processing.

3. **File Writing**

   - Direct Writing in Safe Mode: In the error-free operational mode, packets are written to files immediately as they are captured. While the complexity of writing a single packet is $\mathcal{O}(1)$, the cumulative effect of writing n packets one by one can lead to $\mathcal{O}(n)$ complexity, primarily due to the overhead associated with continuous file access and disk I/O.

   - Batch Writing in Attack Mode: When in attack mode, packets are written in batches, which consolidates multiple write operations into fewer transactions. Although the total number of packets written remains $\mathcal{O}(n)$, batch processing can significantly reduce the overhead caused by frequent disk access, effectively improving the script's performance.

4. **File Writer Thread**

   - Operates concurrently with packet capture, writing out buffered packets. The separation of capture and writing may improve performance but does not change the overall complexity of writing n packets, $\mathcal{O}(n)$.

5. **Conditional Logic for Packet Processing**

   - Condition Checks per Packet: Each packet undergoes several condition checks, such as determining if it meets the criteria to be written immediately or not based on the mode. Additionally, checks include TCP flags and packet type validations, all of which are $\mathcal{O}(1)$ operations. These operations, while individually constant in time, contribute to the overall processing logic and flow control within the script.

6. **Thread Management and Looping**

- A separate thread is used for writing files and the main thread for capturing and processing packets. Thread creation and joining are $\mathcal{O}(1)$ operations, while the loop that captures packets operates indefinitely until interrupted.

Overall Complexity for Monitoring Tool
The dominant factor is the packet processing and writing, both of which are $\mathcal{O}(n)$, where n is the number of packets captured and processed. The use of queues and threading introduces overhead for context switching and inter-thread communication, but this overhead does not change the linear relationship between the number of packets and the time complexity of the script. he complexity of operations that depend on external factors, like writing to disk, may vary based on the system and conditions but are generally considered $\mathcal{O}(n)$ for n packets. Thus, the script's cost complexity for processing and writing network packets is $\mathcal{O}(n)$ where n is the number of packets. This linear complexity indicates that the execution time scales directly with the number of packets, under the assumption that the complexity of per-packet operations remains constant. The actual runtime performance can be affected by factors such as disk IO efficiency, system load, and how the operating system schedules threads and handles IO operations.

5.5.2 Safe Behavior Evaluation Tool

1. **File Reading and DataFrame Initialization**
    - Reading File Line by Line: This operation has a time complexity of $\mathcal{O}(n)$
    - DataFrame Creation: Constructing a DataFrame from a list of lines involves $\mathcal{O}(n)$ operations, as each line is converted into a row in the DataFrame.
2. **Throughput Calculation**
    - Min and Max Timestamps: Finding the minimum and maximum timestamps in a DataFrame column is $\mathcal{O}(n)$ which involves scanning the timestamp column.
    - Summing Data Length: The sum of the data length column is $\mathcal{O}(n)$.
3. **Packet Profile Calculation**

- Time Difference Calculation: Using .diff() to calculate differences between consecutive timestamp entries is $\mathcal{O}(n)$
- Grouping and Counting Unique Streams: The group-by operation and calculating the number of unique group combinations involve hashing, which is $\mathcal{O}(n)$.
- Statistical Measures: Computing average, standard deviation, and sums for various packet attributes involves $\mathcal{O}(n)$ operations.

4. **File Processing**
   - Globbing Files: Using glob.glob to find files matches the pattern in the directory is $\mathcal{O}(m)$ where m is the number of files.
   - Processing Each File: Each file is processed independently, where the operations for each file described above sum to $\mathcal{O}(n_i)$ for n_i lines per file. The total time complexity when processing all files is thus $\sum O(n_i) = O(N)$, where N is the total number of lines across all files.

Overall Complexity for Safe Behavior Evaluation Tool
The overall complexity for the safe behavior evaluation tool remains $\mathcal{O}(N)$, where N is the total number of lines across all files.

5.5.3 Detection and Trust Evaluation Tool

1. **Finding and Sorting New Files**
   - Globbing Files: Uses glob.glob to list files that match a specific pattern, with a complexity of $\mathcal{O}(m)$, where m is the total number of files in the directory.
   - Sorting Files: After extracting numeric values from filenames, the files are sorted, which is $\mathcal{O}(klogk)$ where k is the number of files that match the pattern. Sorting is based on numeric values extracted from filenames, typically an efficient operation but still adheres to log-linear complexity due to the sorting.
2. **Data Initialization from Files**
   - Reading and Parsing Lines: Each line in the file is read and parsed, leading to an $\mathcal{O}(n)$ complexity, where n is the number of lines in the file.

- DataFrame Creation: Constructing a pandas DataFrame from this list of dictionaries and converting data types (e.g., converting timestamps to datetime objects and some strings to numeric types) also runs in $\mathcal{O}(m)$.

3. **Loading and Caching Metrics**

   - Reading safe metrics file: Loading file into a DataFrame has a complexity of $\mathcal{O}(p)$, where p is the number of rows in the file
   - Caching Metrics: Storing these metrics in a dictionary for quick lookup is $\mathcal{O}(1)$ per operation but improves efficiency for subsequent accesses.

4. **Anomaly Detection and Trust Management**

   - Processing Blocks: The script divides packets into blocks of a fixed size. Processing all packets this way is $\mathcal{O}(n)$, since each packet is processed exactly once.
   - Anomaly Checks: Each anomaly check is $\mathcal{O}(1)$ per packet.
   - Trust Score Adjustments: Adjustments are made based on the anomalies detected within each block. While each adjustment is $\mathcal{O}(1)$, the frequency of these adjustments depends on the number of anomalies detected. If many anomalies are detected across packets, this can approach $\mathcal{O}(n)$.

Overall Complexity for Detection and Trust Evaluation Tool

The overall cost complexity of the Detection and Trust Evaluation Tools is $\mathcal{O}(nlogn)$ where n is the dominant factor representing the number of lines in the files and the number of packets processed.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

This thesis aims to implement an Intrusion Detection System for IoT networks, leveraging trust metrics to create a flexible and adaptive system. This IDS is designed to effectively respond to network anomalies and potential security threats by employing dynamic trust assessments.

The proposed IDS is highly adaptable due to its flexible configuration file, which contains several parameters related to node communication. These parameters can be adjusted at any time to meet the system's needs, ensuring that the IDS can adapt to varying network conditions and requirements.

The system responds accurately to anomalies by adjusting trust levels as expected, either increasing or decreasing trust based on the behavior observed. This dynamic adjustment helps maintain the integrity and reliability of the IoT network.

A topic to be discussed is the block size. It is well understood that a shorter block size may detect anomalies more quickly and adjust trust scores promptly compared to larger block size. This is particularly advantageous for attacks detected instantly, as fewer packets will pass after the attack if it occurs at the start of the block. However, this increased sensitivity comes with a trade-off in terms of CPU usage.

When considering the computational complexity of the Detection and Trust Evaluation Tool, which has an overall cost complexity of O(nlogn), where n represents the number of lines in the files and the number of packets processed, it becomes clear that the block size directly impacts CPU load. Smaller block size result in more frequent evaluations, increasing the CPU load due to more frequent data processing and anomaly checks, thus increasing the

value of log(n). Conversely, larger block size reduce the frequency of these evaluations, lowering the CPU demand but potentially delaying the detection of anomalies. With larger blocks, the total number of packets processed in each evaluation increases, which also impacts the overall complexity.

Thus, the choice of block size should balance the need for timely anomaly detection with the available CPU resources. In environments where CPU resources are constrained, larger block sizes may be necessary to maintain overall system performance, while in more resource-rich settings, shorter block sizes can be employed to enhance the responsiveness of the IDS.

6.2 Future Work

The proposed Intrusion Detection System (IDS) for IoT networks, while robust in its current form, offers substantial opportunities for enhancement to better meet the security needs of diverse network environments. One major feature for future purpose to be added is extending the tool to control what happens when the trust of a node falls below a permitted threshold. This means implementing mechanisms to ban or limit the actions of such nodes. The current implementation focuses primarily on collecting trust data rather than managing or controlling nodes with low trust scores.

Additional improvements could involve refining how trust is collected. For example, if the same attack occurs multiple times, it should automatically be considered high risk, reducing the need for constant administrator intervention. This self-learning capability would allow the system to adapt to recurring threats more effectively. Another enhancement in the implementation involves creating a configuration file for each node communication instead of a single configuration for all nodes. This modification will improve efficiency and adaptability, tailoring the settings to the specific requirements of each communication flow.

Moreover, a pivotal improvement could be enhancing the detection mechanism to address the problem where some attacks are fragmented across two blocks due to the block size, potentially leading to missed detections. Implementing a sliding window approach could mitigate this issue. By analyzing overlapping segments of traffic rather than discrete blocks, the system can maintain a continuous view of the network activity, thereby detecting anomalies that span multiple blocks. This approach would improve the accuracy of anomaly detection and reduce the likelihood of splitting attacks

into undetected parts, ensuring a more comprehensive and effective security monitoring.

Another enhancement would be to extend the tool's compatibility to work with other protocols beyond TCP. Although the focus was on TCP packets to demonstrate the mechanism, incorporating other protocols would broaden the system's applicability and enhance its security coverage. Moreover, the list of detectable attacks can be expanded by implementing more sophisticated functions to identify various types of threats.

Currently, the system is designed with the assumption that node-to-node communication exhibits consistent periodicity and uniform packet sizes. This limits its applicability in scenarios where traffic patterns are more variable. Developing methodologies to detect anomalies in such non-uniform networks could involve designing the system to accommodate varying traffic patterns, thus ensuring its robustness and applicability across different network scenarios. These advancements aim to create a more flexible and adaptive IDS capable of effectively responding to both network errors and potential security threats through sophisticated monitoring and analysis techniques.

These enhancements not only aim to improve the preventive capabilities and responsiveness of the IDS but also ensure it remains highly effective in the evolving landscape of IoT security challenges.

References

[2] Martin Bauer et al. “IoT Reference Model”. In: *Enabling Things to Talk: Designing IoT solutions with the IoT Architectural Reference Model*. Ed. by Alessandro Bassi et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 113–162. ISBN: 978-3-642-40403-0. DOI: 10.1007/978-3-642-40403-0_7. URL: https://doi.org/10.1007/978-3-642-40403-0_7.

[3] Musa G. Samaila et al. “Security Challenges of the Internet of Things”. In: *Beyond the Internet of Things: Everything Interconnected*. Ed. by Jordi Mongay Batalla et al. Cham: Springer International Publishing, 2017, pp. 53–82. ISBN: 978-3-319-50758-3. DOI: 10.1007/978-3-319-50758-3_3. URL: https://doi.org/10.1007/978-3-319-50758-3_3.

[4] Shikhar Verma et al. “A Survey on Network Methodologies for Real-Time Analytics of Massive IoT Data and Open Research Issues”. In: *IEEE Communications Surveys & Tutorials* 19.3 (2017), pp. 1457–1477. DOI: 10.1109/COMST.2017.2694469.

[5] Arsalan Mosenia and Niraj K. Jha. “A Comprehensive Study of Security of Internet-of-Things”. In: *IEEE Transactions on Emerging Topics in Computing* 5.4 (2017), pp. 586–602. DOI: 10.1109/TETC.2016.2606384.

[6] Yinghui Huang and Guanyu Li. “Descriptive models for Internet of Things”. In: *2010 International Conference on Intelligent Control and Information Processing*. 2010, pp. 483–486. DOI: 10.1109/ICICIP.2010.5564232.

[7] Ala Al-Fuqaha et al. “Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications”. In: *IEEE Communications Surveys & Tutorials* 17.4 (2015), pp. 2347–2376. DOI: 10.1109/COMST.2015.2444095.

[8] Tasneem Yousuf et al. “Internet of Things (IoT) Security: Current Status, Challenges and Countermeasures”. In: *International Journal for Information Security Research* 5 (Dec. 2015), pp. 608–616. DOI: 10.20533/ijisr.2042.4639.2015.0070.

[9] Ansam Khraisat et al. “Survey of intrusion detection systems: techniques, datasets and challenges”. In: *Cybersecurity* 2.1 (July 2019), p. 20.

ISSN: 2523-3246. DOI: 10.1186/s42400-019-0038-7. URL: https://doi.org/10.1186/s42400-019-0038-7.

[10] Jia Guo, Ing-Ray Chen, and Jeffrey J.P. Tsai. "A survey of trust computation models for service management in internet of things systems". In: *Computer Communications* 97 (2017), pp. 1–14. ISSN: 0140-3664. DOI: https://doi.org/10.1016/j.comcom.2016.10.012. URL: https://www.sciencedirect.com/science/article/pii/S0140366416304959.

[11] Avani Sharma et al. "Towards trustworthy Internet of Things: A survey on Trust Management applications and schemes". In: *Computer Communications* 160 (2020), pp. 475–493. ISSN: 0140-3664. DOI: https://doi.org/10.1016/j.comcom.2020.06.030. URL: https://www.sciencedirect.com/science/article/pii/S0140366419319073.

[12] Ing-Ray Chen, Jia Guo, and Fenye Bao. "Trust Management for SOA-Based IoT and Its Application to Service Composition". In: *IEEE Transactions on Services Computing* 9 (Oct. 2014), pp. 1–1. DOI: 10.1109/TSC.2014.2365797.

[13] Yosra Ben Saied et al. "Trust management system design for the Internet of Things: A context-aware and multi-service approach". In: *Computers & Security* 39 (2013), pp. 351–365. ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2013.09.001. URL: https://www.sciencedirect.com/science/article/pii/S0167404813001302.

[16] Sebastian Ries et al. "CertainLogic: A Logic for Modeling Trust and Uncertainty". In: *Trust and Trustworthy Computing*. Ed. by Jonathan M. McCune et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 254–261. ISBN: 978-3-642-21599-5.

[17] Y. Wang and Julita Vassileva. "Bayesian Network-Based Trust Model". In: Nov. 2003, pp. 372–378. ISBN: 0-7695-1932-6. DOI: 10.1109/WI.2003.1241218.

[20] Mukrimah Nawir et al. "Internet of Things (IoT): Taxonomy of security attacks". In: *2016 3rd International Conference on Electronic Design (ICED)*. 2016, pp. 321–326. DOI: 10.1109/ICED.2016.7804660.

[21] Rajesh Shrivastava et al. "Securing Internet of Things devices against code tampering attacks using Return Oriented Programming". In: *Computer Communications* 193 (Sept. 2022), pp. 38–46. DOI: 10.1016/j.comcom.2022.06.033.

[22] Ashraf Al Sharah, Hamza Abu Owida, and Talal A. Edwan. "Aggressive Jamming Attack in IoT Networks". In: *2022 4th IEEE Middle East*

and North Africa COMMunications Conference (MENACOMM). 2022, pp. 235–239. DOI: 10.1109/MENACOMM57252.2022.9998231.

[23] Shahriar Mohammadi and Hossein Jadidoleslamy. "A Comparison of Link Layer Attacks on Wireless Sensor Networks". In: *International Journal on Applications of Graph Theory in Wireless ad hoc Networks and Sensor Networks* 02 (Mar. 2011). DOI: 10.5121/jgraphhoc.2011.3103.

[25] Qiong Zhang and Wenzheng Zhang. "Accurate detection of selective forwarding attack in wireless sensor networks". In: *International Journal of Distributed Sensor Networks* 15.1 (2019), p. 1550147718824008. DOI: 10.1177/1550147718824008. eprint: https://doi.org/10.1177/1550147718824008. URL: https://doi.org/10.1177/1550147718824008.

[27] Sakshi Sachdeva and Parneet Kaur. "Detection and analysis of Jellyfish attack in MANETs". In: *2016 International Conference on Inventive Computation Technologies (ICICT)*. Vol. 2. 2016, pp. 1–5. DOI: 10.1109/INVENTIVE.2016.7824793.

[29] Okan Yaman et al. "A Novel Countermeasure for Selective Forwarding Attacks in IoT Networks". In: *2022 3rd International Informatics and Software Engineering Conference (IISEC)*. 2022, pp. 1–6. DOI: 10.1109/IISEC56263.2022.9998204.

[30] Zeeshan Ali Khan and Peter Herrmann. "A Trust Based Distributed Intrusion Detection Mechanism for Internet of Things". In: *2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*. 2017, pp. 1169–1176. DOI: 10.1109/AINA.2017.161.

[31] Dong Chen et al. "TRM-IoT: A Trust Management Model Based on Fuzzy Reputation for Internet of Things". In: *Comput. Sci. Inf. Syst.* 8 (Oct. 2011), pp. 1207–1228. DOI: 10.2298/CSIS110303056C.

[32] Upul Jayasinghe et al. "Machine Learning Based Trust Computational Model for IoT Services". In: *IEEE Transactions on Sustainable Computing* 4.1 (2019), pp. 39–52. DOI: 10.1109/TSUSC.2018.2839623.

External Links

[1] "intelliot". In: (). URL: https://intelliot.eu/.

[14] "belief theory". In: (). URL: https://en.wikipedia.org/wiki/Dempster%E2%80%93Shafer_theory.

[15] "subjectivelogic". In: (). URL: https://en.wikipedia.org/wiki/Subjective_logic.

[18] "fuzzylogic". In: (). URL: https://en.wikipedia.org/wiki/Fuzzy_logic.

[19] "regression". In: (). URL: https://en.wikipedia.org/wiki/Regression_analysis.

[24] "osi". In: (). URL: https://en.wikipedia.org/wiki/OSI_model.

[26] "dos". In: (). URL: https://en.wikipedia.org/wiki/Denial-of-service_attack.

[28] "replay attack". In: (). URL: https://en.wikipedia.org/wiki/Replay_attack.

[33] "kmeans". In: (). URL: https://en.wikipedia.org/wiki/K-means_clustering.

[34] "ns3". In: (). URL: https://www.nsnam.org/.

[35] "sigcomm". In: (). URL: https://conferences.sigcomm.org/sigcomm/2009/.

[36] "dpkt". In: (). URL: https://dpkt.readthedocs.io/en/latest/.

[37] "Pcapy". In: (). URL: https://pypi.org/project/pcapy-ng/.

[38] "pandas". In: (). URL: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html.

[39] "Nping". In: (). URL: https://nmap.org/nping/.