Universidade Estadual de Londrina

FERNANDO HENRIQUE YOSHIAKI NAKAGAWA

ATTACK DETECTION IN SMART HOME NETWORKS USING CLUSTREAM AND PAGE-HINKLEY TEST

LONDRINA

2022

FERNANDO HENRIQUE YOSHIAKI NAKAGAWA

ATTACK DETECTION IN SMART HOME NETWORKS USING CLUSTREAM AND PAGE-HINKLEY TEST

Dissertação apresentada ao Programa de Mestrado em Ciência da Computação da Universidade Estadual de Londrina para obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Bruno Bogaz Zarpelão

LONDRINA

2022

Ficha de identificação da obra elaborada pelo autor, através do Programa de Geração Automática do Sistema de Bibliotecas da UEL

N163a Nakagawa, Fernando Henrique Yoshiaki.
Attack Detection in Smart Home Networks using CluStream and Page-Hinkley Test / Fernando Henrique Yoshiaki Nakagawa. - Londrina, 2022.
71 f. : il.

Orientador: Bruno Bogaz Zarpelão.
Dissertação (Mestrado em Ciência da Computação) - Universidade Estadual de Londrina, Centro de Ciências Exatas, Programa de Pós-Graduação em Ciência da Computação, 2022.
Inclui bibliografia.

1. Internet das Coisas - Tese. 2. Mineração de Fluxos Contínuos de Dados - Tese. 3. Aprendizado Não-supervisionado - Tese. 4. Detecção de Ataques Cibernéticos - Tese. I. Zarpelão, Bruno Bogaz. II. Universidade Estadual de Londrina. Centro de Ciências Exatas. Programa de Pós-Graduação em Ciência da Computação. III. Título.

CDU 519

FERNANDO HENRIQUE YOSHIAKI NAKAGAWA

ATTACK DETECTION IN SMART HOME NETWORKS USING CLUSTREAM AND PAGE-HINKLEY TEST

Dissertação apresentada ao Programa de Mestrado em Ciência da Computação da Universidade Estadual de Londrina para obtenção do título de Mestre em Ciência da Computação.

BANCA EXAMINADORA

Orientador: Prof. Dr. Bruno Bogaz Zarpelão
Universidade Estadual de Londrina

Prof. Dr. Sylvio Barbon Jr.
Universidade Estadual de Londrina – UEL

Prof. Dr. Rodrigo Sanches Miani
Universidade Federal de Uberlândia – UFU

Londrina, 12 de dezembro de 2022.

AGRADECIMENTOS

Não foi um período fácil no mundo inteiro, o que torna este trabalho ainda mais especial para mim. Por isso inicio agradecendo especialmente o meu orientador, professor Bruno Zarpelão, pelos ensinamentos, paciência e excepcional orientação durante o mestrado.

Agradeço aos meus pais, Claudio. e Alice, pelo amor, apoio e por terem lutado por muitos anos para me darem a melhor educação. À minha irmã Andressa pelo carinho e exemplo.

Também agradeço à toda minha família, de sangue, de amizades, ao Dan, pelo apoio, compreensão por ter me afastado um pouco e pelos momentos de descontração.

Tenho um agradecimento muito especial à minha tia Meiri Sakuma Nakagawa pelas palavras de motivação! Mais agradecimentos especiais para meus amigos com quem em algum momento, troquei alguma ideia desajeitada sobre como esta trajetória foi desafiadora! São eles: Beatriz Muniz, Felipe Salerno, Fernando Almeida, Gabriel Darcin, Íris Perfetto, Jéssica Koyama, Juliana Bambini, Juliana Trevisan, Lucinéia Pereira, Mariana Marques, Natália Vilas Boas, Nathalia Bertelli e Victor Zanin.

Sou eternamente grato aos professores do Departamento de Computação, pois sem a dedicação deles eu não estaria aonde estou hoje.

Por último, agradeço ao pessoal do Senai pela motivação e disponibilidade de me ajudar, sempre torcendo para que este trabalho fosse concluído.

"Everything you've ever wanted is sitting on the other side of fear."
"Tudo o que você sempre sonhou está do outro lado do medo."
ADDAIR, George apud DAVIS, Viola

NAKAGAWA, F. H. Y.. **Attack Detection in Smart Home Networks using CluStream and Page-Hinkley Test**. 2022. 71f. Dissertação (Mestrado em Ciência da Computação) – Universidade Estadual de Londrina, Londrina, 2022.

RESUMO

A expansão das redes IoT aumenta a procura por sistemas de segurança que detectem ataques contra estes novos alvos. Tais dispositivos têm hardware simples, com memória e poder de processamento limitados, e muitas vezes são obrigados a ter um baixo consumo de energia. Atualmente são utilizados algoritmos de aprendizagem supervisionada por lotes para resolver este problema, mas eles apresentam algumas limitações. Estes algoritmos exigem amostras benignas e maliciosas para serem treinados, o que pode ser difícil de obter em redes reais. Além disso, uma vez treinados, é difícil atualizar o modelo de aprendizagem com comportamentos recentemente encontrados. Neste trabalho, propomos um esquema online e não supervisionado para detectar ataques em redes IoT residenciais. Este esquema é baseado na combinação de dois algoritmos: CluStream e Page-Hinkley Test. O esquema proposto não requer amostras rotuladas para ser treinado e aprende de forma incremental à medida em que é utilizado. Os testes foram realizados sobre dados obtidos a partir de conjuntos de dados disponíveis publicamente, consistindo em múltiplos dispositivos domésticos inteligentes e os resultados são satisfatórios. Diferentes tipos de ataques foram detectados com uma taxa de detecção global acima de 92%, enquanto a precisão se manteve por perto de 81%, com atraso médio de 151 iterações.

Palavras-chave: Internet das Coisas, Mineração de Fluxos Contínuos de Dados, Aprendizado Não-supervisionado, Detecção de Ataques, Cibersegurança.

NAKAGAWA, F. H. Y.. **Attack Detection in Smart Home Networks using CluStream and Page-Hinkley Test**. 2022. 71p. Master's Thesis (Master in Science in Computer Science) – State University of Londrina, Londrina, 2022.

ABSTRACT

The expansion of IoT networks increases the demand for security systems that detect attacks against these new targets. These devices have simple hardware, limited memory and processing power, and often are required to have low energy consumption. Batch supervised learning algorithms have been employed to address this issue, but they present some limitations. These algorithms demand benign and malicious labeled samples to be trained, which can be hard to obtain in real networks. Also, once they are trained, it is hard to update the learning model with newly found behaviors. In this work, we propose an online and unsupervised scheme to detect attacks in smart home IoT networks. This scheme is based on the combination of two algorithms: CluStream and Page-Hinkley Test. It does not require labeled samples to be trained and learns incrementally as it is used. Tests were performed over data obtained from publicly available datasets consisting of multiple smart home devices and the results are encouraging. Different types of attacks were detected with a overall detection rate above 92%, while the precision stayed around 81%, with an average delay of 151 iterations.

Keywords: Internet of Things, Stream Mining, Unsupervised Learning, Attack Detection, Cybersecurity.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

2FA	Two-Factor Authentication
6LoWPAN	IPv6 over Low power Wireless Personal Area Networks
ACCS	Australian Center for Cyber Security
AD	Anomaly-based Detection
ADWIN	Adaptive Windowing Method
AE	Auto-Encoder
ARIMA	Autoregressive Integrated Moving Average
ARP	Address Resolution Protocol
BLE	Bluetooth Low-Energy
CDIA	Covert Data Integrity Assault
CIDN	Collaborative Intrusion Detection Network
CIDS	Collaborative Intrusion Detection System
CoAP	Constrained Application Protocol
CPU	Central Processing Unit
CUSUM	Cumulative Sum Algorithm
CWR	Congestion Window Reduced flag
DDoS	Distributed Denial of Service
DNN	Deep Neural Network
DNS	Domain Name System
DoS	Denial of Service
DSCP	Differentiated Services Code Point
DTL	Deep Transfer Learning
DVR	Digital Video Recorder
ECE	Explicit Congestion Notification Echo

ECN	Explicit Congestion Notification
EDDM	Early Drift Detection Method
GDPR	General Data Protection Regulation
GEO	Geosynchronous Equatorial Orbit
GPS	Global Positioning System
GSM	Global System for Mobile Communications
HIDS	Host-based Intrusion Detection System
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
IDS	Intrusion Detection System
IDPS	Intrusion Detection and Prevention System
IETF	Internet Engineering Taskforce
IPS	Intrusion Prevention System
IPSec	Internet Protocol Security
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
IP	Internet Protocol
IPv4	Internet Protocol version 4
LEO	Low Earth orbit
LGPD	Lei Geral de Proteção de Dados
LOF	Local Outlier Factor
LoRaWAN	Low-power, wide area networking
MDA	Marginalized De-noising Auto-encoder
MFA	Multi-Factor Authentication
MIDS	Mixed Intrusion Detecion System
MITM	Man-in-the-middle Attack

MQTT	Message Queuing Telemetry Transport
MSDA	Marginalized Stacked De-noising Auto-encoder
MKL	Multiple Kernel Learning
ML	Machine Learning
NB-IoT	Narrowband-IoT
NBA	Network Behavior Analysis
NIDS	Network-based Intrusion Detection System
NFC	Near Field Communication
NS	Nounce Sum
OSI	Open Systems Interconnection
OTA	Over-the-air
OWASP	Open Web Application Security Project
PCA	Principal Component Analysis
PIPEDA	Personal Information Protection and Electronic Documents Act
PH	Page-Hinkley
QoS	Quality of Service
RFC	Request for Comments
RFID	Radio frequency identification
SD	Signature-based detection
SDA	Stacked De-noising Auto-encoder
SLFN	Single Hidden Layer Feed-Forward Neural Network
SMOTE	Synthetic Minority Oversampling Technique
SMTP	Simple Mail Transfer Protocol
SPA	Stateful Protocol Analysis
SSL	Secure Sockets Layer
SVM	Support Vector Machine

TCP	Transmission Control Protocol
TLS	Transport Layer Security
TTL	Time to Live
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunication System
VPN	Virtual Private Network
WIDS	Wireless-based Intrusion Detection System
WPS	Wi-Fi Protected Setup

CONTENTS

1 INTRODUCTION

Internet of Things (IoT) devices in home environments ease people's everyday lives, e.g., by making daily tasks more efficient, increasing home security, and enabling health monitoring. However, while these devices bring seamless experiences of ubiquitous connectivity, they may contribute to privacy and security problems. Most people trust these home appliances, but they do not know what information these devices transmit and store. Also, as these devices have high connection availability and low security, they become targets for attackers. Malicious users may use these opportunities to create botnets, spread malware, and break into people's privacy. This current scenario creates a demand for security controls that are able to detect attacks against these new targets [9, 10, 11, 12].

IoT systems have simple hardware, including limited memory and processing power. Some security technologies are still challenging to be deployed in these devices, such as cryptography or intrusion detection systems (IDS). Several approaches proposed to detect attacks in IoT systems are based on batch machine learning algorithms. Also, they usually consist of supervised learning schemes, which require labeled samples to be trained and create a learning model [9, 10, 13].

Two main issues arise when it comes to batch supervised algorithms in this context. Firstly, batch algorithms cannot learn incrementally. These algorithms approach their problems as they were static: a finite amount of training data is gathered, a static model is trained, and then it is applied to classify future observations. When the data source behavior changes, updating the learned model may be difficult. Secondly, labeled samples are hard to obtain in real scenarios, especially malicious ones. Even if the user had these labeled samples, it would be necessary to update the model every time the network behavior changes. Most domestic users are not technology experts to manage their network security [10, 11]. Therefore, it would be troublesome for them to use an attack detection system based on a batch supervised learning algorithm.

An alternative to batch learning algorithms is to use stream learning. Stream learning algorithms consider that input stream elements arrive online, and these data streams are potentially unbound in size. For this reason, stream learning algorithms do not store the data elements in memory, only statistical information about them. Also, they can learn incrementally, updating their models as new data arrives. Like batch learning algorithms, stream learning schemes can be supervised or unsupervised. By using an unsupervised process, they do not require labeled samples to be trained [6, 14].

In this study, we propose an online and unsupervised scheme to detect attacks in

smart home IoT networks. The proposed approach starts by dividing the incoming packets into three streams: ICMP, UDP, and TCP streams. Each of these streams is inputted to a CluStream algorithm instance. This algorithm partitions the incoming packets online, creating, updating, and destroying micro-clusters as new packets arrive. Then, the scheme monitors the distance among the micro-clusters to detect new and sudden changes that may indicate attacks. The continuous collection of this indicator feeds a Page-Hinkley Test, which detects sudden changes automatically. Experiments were carried out over data collected from a publicly available dataset. The results show that the proposed approach can detect different types of attacks and has a low false-positive rate in most of the observed situations.

Previous work has proposed IDSs for IoT using different approaches mostly based on batch learning. In Anthi et al. [9] and Moustafa et al. [13], a supervised approach was used. Qaddoura et al.[15] and Vu et al. [16] used a hybrid approach, with supervised and unsupervised algorithms. Meidan et al. [17], Bezerra et al. [18] and de Melo et al. [19] employed one-class classification algorithms for attack detection. Finally, Brun et al. [20] and Ali and Li [21] used deep learning in their proposals. The main contribution of this work is to propose a new framework for detecting attacks through unsupervised learning in continuous data streams of smart home IoT networks. The model is able to adjust its sensitivity for different scenarios (different devices and protocols), combining clustering and detection of changes in network traffic, packet by packet.

This work is organized as follows: Chapter 2 presents the theoretical background, describing the fundamentals of IoT security, IDS, stream learning, stream clustering, change detection and the related work. In Chapter 3, we explain our proposed approach. Chapter 4 presents the evaluation with experimental design, metrics and results. Lastly, Chapter 5 presents the conclusion.

2 THEORETICAL BACKGROUND

This chapter provides an overview of the body of knowledge that underpins our proposal. It starts by describing a typical IoT architecture and its layers, alongside an overview on IoT attacks and vulnerabilities. Then, the chapter approaches the main concepts about intrusion detection systems, stream clustering, and change detection.

2.1 Internet of Things

In IoT, physical objects ("things") are connected to the Internet to provide and consume services like any other host. Billions of these smart devices are currently in operation on top of different communication protocols and platforms, which raises a concern about their communications' security [22, 23]. The architecture of a typical IoT system is composed of three layers of communication protocols [1, 2, 24, 4] as seen in Figure 1:

- **Perception Layer**: this layer is similar to the physical layer in the OSI (Open Systems Interconnection) reference model. In this hardware based layer, data is collected and stored by sensors and security measures should begin to take place. Another function of this layer is to transmit the collected data to the immediately higher layer. Included in the Perception Layer are technologies like RFID, barcodes, cameras, GPSs, and various kinds of sensors (temperature, pH level, humidity, gas, etc).

- **Network Layer**: it makes up the network data pathway from the sensors to the servers using mostly wireless protocols as well as other standard network protocols. In other words, it is responsible for transmitting data generated by the Perception Layer to the Application Layer. Examples of network protocols used in this layer are WiFi, ZigBee, Bluetooth, BLE, NFC, 3G, 4G, 5G, GSM, UMTS, 6LoWPAN, infrared, Z-Wave, LoRaWAN, NB-IoT, CoAP, MQTT, LEO and GEO satellites, etc.

- **Application Layer**: services can be provided to users over mobile, web-based applications or the direct interaction with IoT devices. These applications are hosted in servers or cloud services, which are responsible for processing and analyzing incoming data from the layer below. Anytime, this layer can send information and feedbacks to users using the Network Layer. It is able to control, discover and coordinate devices in a smart home environment.

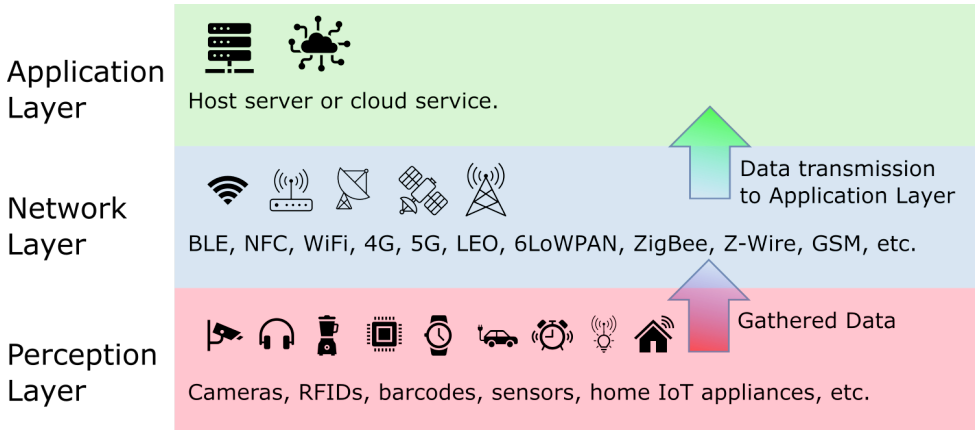

Figure 1 – IoT Architecture. Adapted from [1, 2, 3, 4]

.

Smart homes are among the most popular application segments that IoT made possible [10]. A smart home is a residence or a domestic unit with devices connected to the Internet to ease residents' lives and their everyday tasks. It is possible to automate and remotely manage locks, air conditioners, refrigerators, washing machines, surveillance cameras, etc. The demand for IoT smart home devices has been growing over the years. Due to their convenience and ease of use, they are becoming part of people's routine. The number of smart home devices expanded 11.7% from 2020 to 2021, having more than 895 million of devices sold [25]. More than 478 million homes are expected to become smart and projected to reach a $1567 billion market by 2025 [6, 14].

2.2 IoT Attacks

Attacks against IoT systems may occur in any of the IoT architecture layers. They can be classified into physical, passive cyber attack or active cyber attack [1, 4, 26]:

- **Physical attack**: its intent is to generate physical damage to the device. An attacker can physically affect reachable devices or components (e.g., energy or data cables) to interrupt a service. Also, a physical damage does not need an attacker to occur, since it can be caused by natural disasters [4] or an unintended accident.
- **Passive cyber attack**: it is performed by collecting data from a network without the consent of its owner (e.g., network scanning). It is mostly executed to collect private information or to look for a vulnerability to perform an active cyber attack.
- **Active cyber attack**: active attacks are launched by malicious users trying to use an IoT system without authorization to achieve their own objectives [1] (e.g., DoS, DDoS, MITM, spoofing, privilege escalation, etc). Systems' configurations can be

maliciously modified to create bots, allow illegal authorizations, deny services, along with others.

Multiple kinds of attacks may affect IoT devices and networks. An attacker performs a network scanning to gather valuable information about the target. Through this attack, a malicious agent can enumerate potential vulnerabilities of the target and get prepared to exploit them [27].

There are different types of scanning, which may be used according to the attacker's objective. The first one consists of investigating a specific port on multiple IP addresses, searching for a known vulnerability. Another kind of scanning attack is to look for vulnerabilities in multiple ports on a single IP address. Also, this attack can be originated by multiple IP addresses, and/or in stealth mode, which delays the time between attacks, to avoid detection. [27, 28] .

A brute force attack involves a repeated process of interaction between an attacker and a host, which provides a service through a network protocol and destination port [29]. A malicious user can try to get a successful login authentication via multiple attempts. Some systems do not establish a limit for these attempts, so an attacker can try to break into multiple times per second. For each try, the password can vary by only 1 character or another technique can be used to find the password in less time. One of them is based on a dictionary containing most common passwords or meaningful words. Furthermore, it can be used as a cryptoanalysis method. It consists in searching for the key that returns a valid plain text [30]. If the attacker have sufficient computational power, time and no limit for attempts, it is possible to get authenticated within a host or crack a cryptosystem.

Once an attacker finds a weakness in the network, they can use privilege escalation to access the network. This process consists of exploiting bugs, vulnerabilities on a outdated operating system or flaws found in a third party dependency. Once the unauthorized access is granted, the attacker escalates the privileges to the administrator level, making it easier to control the entire system.

Firmware hijacking is especially dangerous, because it can bypass common antivirus software. In this kind of attack, attackers can send malicious updates to overwrite the current device's firmware. Once the firmware is hacked, the attacker is able to use it as part of a botnet to launch a Distributed Denial of Service (DDoS) attack or infect other nearby devices [31].

IP spoofing is a kind of attack that consists of creating fake IP packets, changing the IP header's source address to impersonate and trick a host. Systems that use IP addresses to perform authentication and/or access control are a special target for this kind of attack. Once an attacker gets access to an IoT network, they can add malicious nodes to it. These alien nodes may be used to get sensible data or to control the data

flow among the network nodes.

A man in the middle attack (also known as adversary in the middle attack) occurs when an adversary deflects a legitimate communication session between two parties and places himself between them, monitoring, filtering and changing the traffic data, acting as a proxy [9, 32]. It can be used to steal information, retrieve passwords and issue fake transactions.

Denial of service attacks are launched to take down a service by sending to the target a massive amount of requests. This generates a huge amount of data to process, causes bandwidth high usage, and drains the available memory. Once any of these resources are exhausted, the host will not be able to respond to legitimate users within an acceptable time. DoS attacks can also allow injection and execution of arbitrary commands [33, 34, 35, 26].

A distributed denial of service attack consists of an orchestrated and massive DoS offensive against a host. The attacker makes the host unavailable to legitimate activities due to an excessive allocation of resources caused by the aggression [36]. A botnet is usually used to set these enourmous attacks in motion. A botnet is a network made up of infected hosts (bots) controlled by a botmaster, which corresponds to the attacker. Once the malicious user becomes a botmaster, it is possible to command coordinated attacks against a host.

A security incident involving an IoT system is hardly the result of a single action or breach exploitation. A usual path followed by attackers begins with a network scanning to enumerate their target vulnerabilities. Once a way to get inside the IoT device is found, a malware instance may be deployed. Another way to compromise the device is to perform a DoS or MITM attack to install a malicious firmware update. A compromised device will respond to the adversary's commands. As a result, the attacker is able to get sensitive information, *brick* the device, making it unusable, or turn it into part of a *botnet*. One of the reasons to control a *botnet* is to launch DDoS attacks. Botnets also enable attackers to run distributed network scanning attacks, spread *spam* over the Internet and do cryptocurrency mining [9, 28, 34, 35].

The Mirai botnet is historically known for its large-scale DDoS attacks. About 148,000 infected IoT devices were used to carry out the attacks, including surveillance cameras, digital video recorders (DVRs) and routers. The attacks were primarily directed against Dyn, a DNS service provider. As a result of which diverse services were made unavailable to users for several hours. Soon after the attacks, Mirai's code was made publicly available and variants of the botnet were developed and deployed [17, 37, 38].

2.3 IoT Vulnerabilities

To reach their purpose, attackers need to find a way in by exploring breaches in IoT systems. The Open Web Application Security Project (OWASP) - Internet of Things [39] brings the top ten features that compromise security in IoT environments. It considers the device build phase, as well as the deployment and management of IoT systems. Most of the vulnerabilities should be mitigated by the manufacturer (insecure network services, ecosystem interfaces, update mechanism, default settings, data transfer and storage, outdated components, insufficient privacy protection and lack of device management).Few other breaches could be patched by the end user (weak or guessable passwords and lack of physical hardening). Weak and hardcoded passwords are not an exclusive problem of IoT networks. As they are often provided by the user, most users pick only lower case letters or easily guessable and publicly known words to remember their choice later. These passwords can be easily broken through brute force attacks. Also, backdoors or keyloggers can be used to get users' passwords.

Preventing IoT devices from unauthorized physical access is important because adversaries can benefit from their proximity to the device. The fact that many IoT systems are deployed in unattended environments increases the risk [4]. Attacks that explore this condition are diverse, varying from simple to sophisticated methods. Attacks based on the WPS (Wi-Fi Protected Setup) function in Wi-Fi routers are an example. Many network routers have a WPS button, which allows fast connection without a password. Devices with easy contact to the mainboard may suffer from an introduction of another physical system. Acting as a parasite, the intruder physical system can take over its target. Surfing attacks are more sophisticated. They use voice commands via ultrasonic waves to take control of a device [40, 41]. Another similar approach is to access the device's features by pointing an encoded laser beam or flashlights to the microphone which interprets the light as a digital signal [42, 43, 44].

Problematic hardware and software components included in many devices may also threaten devices' security. Obsolete and unencrypted components, frequently acquired from unverified suppliers, can leave breaches for adversaries. Software dependencies can inherit known and unknown vulnerabilities. Also, operational system customizations' may introduce exploits that could be used by malicious users.

Insecure update of devices is a threat because attackers can use it to damage or take control of the target. Attacks exploring updates become easier when devices do not allow firmware rollback, do not check counterfeit updates or update data transmission occurs on an unencrypted network line. Even when devices work with encrypted communication and methods to check firmware updates, exploits can be used to surpass these security controls and insert a custom malicious firmware [31].

Lack of attention to the system configurations is another source of risk. Users frequently unpack a device and use it without reading instructions or manipulate the default configuration settings. However, these systems may be shipped with insecure settings due to negligence or to facilitate its connection to a home network. Also, some devices do not provide functionalities to change security settings.

Insecure network services, ecosystem interfaces, data transfer and storage can contribute to compromise the device and the network's privacy, authenticity and integrity. The absence of cryptography or the use of weak cryptography during the data transmission and storage is common inside any layer of the IoT architecture. Additionally, there are devices without any authentication process to manipulate their settings. While it makes easier to the user, this can be used as an opportunity to an adversary. On the other side, sophisticated authentication methods, like 2FA (Two-Factor Authentication) or MFA (Multi-Factor Authentication), decreases the flexibility in terms of usability. This trade-off is a challenge to manufacturers that needs attention [4].

Personal information stored in IoT devices or cloud servers may be used in a way that violates users' privacy. Homes should be private and highly protected spaces, which brings up ethical challenges when smart homes and IoT are discussed. Users are potentially being watched and their data processed through data mining and big data analytics. Although attacks can be performed to obtain private data, most of the time, private data is collected out of users' consent to a service/company. This permission is often given by clicking "I Agree" without reading the service privacy police [45]. Some countries have laws to protect personal identifiable information, e.g., Lei Geral da Proteção de Dados (LGPD) in Brazil, European Union's General Data Protection Regulation (GDPR), California Consumer Privacy Act and Canada's Personal Information Protection and Electronic Documents Act (PIPEDA). However the fast technology evolution surpasses policies and laws. Shortly, it is necessary to consider ethics and people's privacy in the whole IoT smart home lifecycle development and design [46].

2.4 Intrusion Detection Systems

According to Scarfone and Bace [47], intrusion detection is defined as:

> The process of monitoring the events occurring in a computer system or network and analyzing them for signs of possible incidents, which are violations or imminent threats of violation of computer security policies, acceptable use policies, or standard security practices.

In response, to deal with these threats and violations, Intrusion Detection Systems (IDSs) are being studied and implemented to bring more security to hosts in a network.

IDSs are based on software, hardware or a mixture of both. Their main objective is to automate the intrusion detection process by capturing and analyzing network packets [48]. This includes detecting security events and warning the network administrator of them through alerts. Also, an IDS can be able to learn from past data to have more true positive alerts [26]. An Intrusion Prevention System (IPS), also known as Intrusion Detection and Prevention System (IDPS), adds one more layer of protection to the network. Alongside attack detection, these devices can actively stop intrusions.

2.4.1 Intrusion Detection Methodologies

IDSs can apply different detection methods, which are classified into three categories: *Signature-based Detection* (SD), *Anomaly-based Detection* (AD) and *Stateful Protocol Analysis* (SPA) [47, 48, 49, 50, 51, 52].

Signature-based detection, also referred to as *Knowledge-based Detection* or *Misuse Detection*, uses a pattern-matching scheme to detect attacks. A pattern is composed of already known characteristics of an attack or threat registered in a knowledge base. Then, monitored data is collected and compared to the pattern, and if they match, the system administrator receives an alert. It is a simple and effective manner to detect known attacks. However, it may be ineffective against unknown attacks and variants of known attacks, which are crafted to mislead SD-IDSs. As intruders have been improving their skills, it is necessary to keep the knowledge base always updated, which demands hard work and much time [51, 53].

In Anomaly-based Detection (AD), also known as *Behavior-based Detection*, collected data is checked for deviations from normal behavior. A normal behavior corresponds to known or expected behavior for system or network parameters like number of host's connections, amount of login attempts, amount of memory in use, packets per second rate, etc. Network packets can also be inspected for unusual values in header fields. When the analysis finds an anomaly in the collected data, the system administrator is alerted. Unlike SD, AD is effective against unknown threats. However, it can take longer to detect an anomaly and its accuracy may degrade because the normal behavior profile is always changing [53].

The Stateful Protocol Analysis (SPA) or *Specification-based Detection* relies on protocol-specific profiles made by vendors, which are based on protocol specifications from standard bodies e.g., RFCs from IETF [54]. A SPA IDS analyzes a protocol e.g. HTTP, and can recognize non-specified sequences of commands or a protocol break.

A positive aspect of this method is that widely adopted protocols are well defined and do not change frequently. Therefore, SPA IDSs do not need constant updates and can detect unknown attacks. On the other hand, this method requires a lot of resources, which becomes impracticable in smart home scenarios, restricting it to robust servers [53].

Another disadvantage is how this IDS presents alerts. It shows which part of the protocol's state machine was broken, which is difficult to interpret even for IDS operators.

While some IDSs are based on a single method, a Hybrid IDS can use multiple methods to achieve better and more accurate results. A study by Aydin, Zaim and Ceylan [55] showed that combining detection approaches contributes positively to the detection capability. This study combined two Anomaly-based Detection systems with Snort[1], an open-source Signature-based Detection system. The resulting hybrid IDS outperformed the two anomaly based IDSs and the signature-based one in terms of detection rate.

2.4.2 Types of IDSs

IDSs can be also categorized according to where they are deployed and which system they are protecting.

Host-based IDSs (HIDS) focus on protecting a particular device. An HIDS is deployed inside the protected device, collecting the local system data, such as system logs, system events, file system status, system calls, network packets, CPU, energy and memory usage. The collected data is analyzed to monitor malicious activities within a system [5, 47, 49, 51, 56].

An HIDS is best suited to detect internal threats, having high reliability and precision, due to its ability to monitor which internal processes and events are involved in a particular attack [48, 51]. Furthermore, since an HIDS is within a host, it can use unencrypted data inside a encrypted network, e.g., Virtual Private Network (VPN) [47, 48]. It may also detect attacks that can go unnoticed by a Network-based IDS (NIDS) [48].

However, HIDSs are more complicated to manage as every monitored host in a network have to be configured and maintained. HIDSs can also be attacked and be disabled because they are inside a host. Finally, additional computational cost is inflicted to monitored hosts, affecting their performance [48].

NIDSs have sensors distributed in the monitored network, connecting one or more hosts. It is possible to monitor big networks with a small number of sensors, thus generating low impact on them [48]. Sensors capture data at specific network points and analyzes information to detect attacks. The analysis of data can be made packet-by-packet or in grouped packets, for example, by a quantity of packets, packets that arrive within a time or protocol [5, 47, 49, 51].

Typically, NIDSs act in the application layer (e.g., Hypertext Transfer Protocol [HTTP], Simple Mail Transfer Protocol [SMTP], and Domain Name System[DNS]), trans-

[1] https://www.snort.org/

port layer (e.g., TCP and UDP) and network layer (e.g., IPv4). Some NIDSs also analyze the physical layer, like in Address Resolution Protocol (ARP) [47].

An advantage of NIDSs is that they can operate invisibly to attackers, granting more security against attacks that disable them [48]. However, it may not be possible to inspect the payload of encrypted network traffic [47, 48]. This leads to having less information to perform traffic security inspection, making attack detection more challenging.

2.5 Stream Learning

In traditional batch learning algorithms, data needs to be accumulated to be processed all at once, meaning that it has a start and an end point. To query from data, batch learning algorithms uses *blocking operators*, which means that the first output result is available only after processing an entire batch dataset [6, 57, 58]. This data can be stored entirely in a database and be used many times. Its processing time and demand for memory can be unlimited, so it might not meet the need for a fast/real-time answer [59]. Finally, the generated model is static, thus harder to be updated, and if the underlying distribution changes, the model may no longer be usable.

Batch data processing frameworks do not meet many requirements of current applications [59]. Big data applications, such as network monitoring, sensor networks, and smart cities demand online answers, even if they can vary within an error margin [60]. Data generated by these applications have an infinite nature, because it is not known when the data stream will end. In addition, traditional database systems were not designed to support infinite storage queries. The behavior of a stream may be unpredictable and the target concept is prone to change over time [59, 60]. This makes necessary to update the learning model frequently and automatically. Due to the nature of these applications, stream learning is an important and present subject that is being studied [6]. Figure 2 illustrates the difference between batch and stream learning schemes.

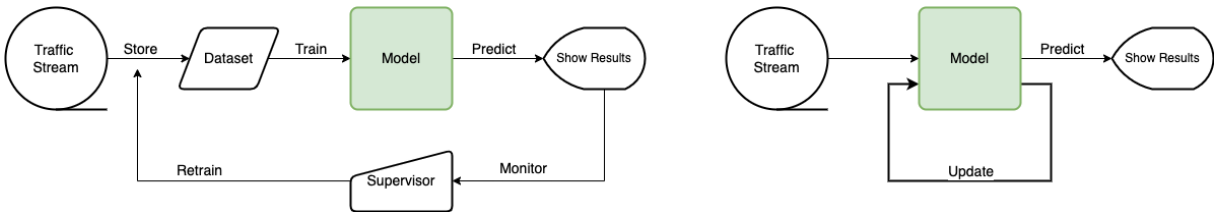

Figure 2 – Learning schemes. (a) Batch; (b) Incremental. Adapted from [5].

The type of data used in stream learning problems are referred to as data streams. Data streams are extended time series with a data flow being continuously supplied by instances $[x_1, x_2, ..., x_t]$ in order of arrival, where t is the time of the most recent data

instance, which does not need to be uniformly spaced between instances [6, 60, 61]. Next, some characteristics of data streams are presented [6, 59, 60]:

- Data streams can be infinite or have its end unknown due to the continuous data flow.
- There is a sequential order of each data instance arrival that cannot be controlled.
- The data incoming to the stream arrives online and may come at high speeds.
- After being processed, the instance is eliminated or stored in a statistical model, which can be fetched if it is still in the memory or can be approximated [58].

To support stream learning using data streams, a stream model can be categorized into three classifications [6, 59, 60]. Let's consider incoming instances as x_t, being processed by a stream model function F:

- **Insert Only Model**: the instance x_t is inserted into the model and is not able to be modified, e.g., traffic data from an IoT network (Figure 3).

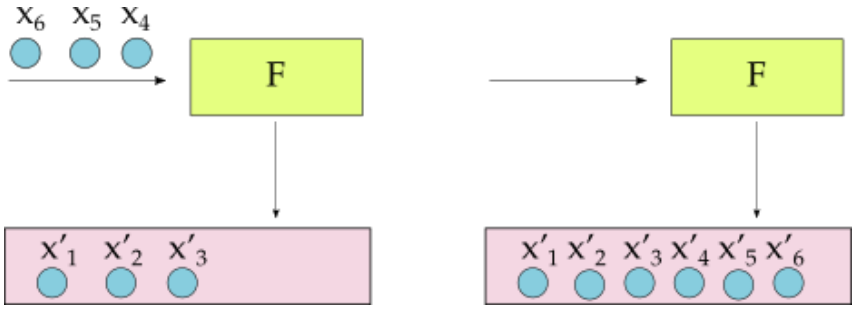

Figure 3 – Insert Only Model.

- **Insert-Delete Model**: after the first insertion into the model, a data instance x_t may be altered or eliminated, e.g., Sensors' status in a machine (Figure 4).

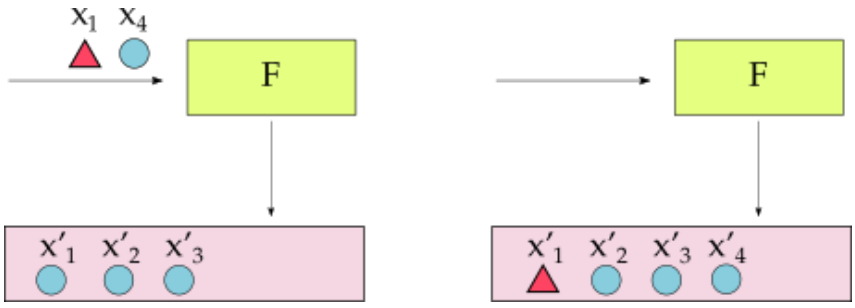

Figure 4 – Insert-Delete Model.

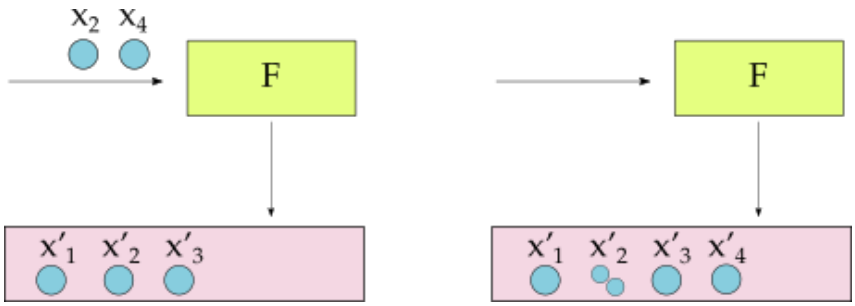

Figure 5 – Accumulative Model.

- **Accumulative Model**: x_t is used to add to $F_t = F_{t-1} + x_t$. E.g., measurement of IP addresses making requests to a host (Figure 5).

Infinite data streams are impossible to be entirely stored. To address this issue, stream learning algorithms do not store all data instances. Instead, techniques can be used to store summary or synopsis data about already processed data. Previous data, when required, can be retrieved through statistics approximations and/or using sliding windows. Although these methods have low memory requirements and allow fast calculations, there is a trade-off between how much data is stored in the summary and the precision of the retrieved data [7, 59, 60].

Without having all the data and blocking operators, it is possible to keep statistics online by computing and updating mean, standard deviation, and correlation as new data instances come in. [6]. An incremental mean calculation is given as:

$$\overline{x_i} = \frac{(i-1) \times \overline{x}_{i-1} + x_i}{i} \tag{2.1}$$

For the number of instances i, $i > 0$, at a moment, the mean $\overline{x}_i$ can be updated only storing in the memory i and the previous calculated mean $\overline{x}_{i-1}$.

To calculate the dispersion of the points, a recursive manner to get the standard deviation is:

$$\sigma_i = \sqrt{\frac{\sum x_i^2 - \frac{(\sum x_i)^2}{i}}{i-1}} \tag{2.2}$$

To keep the standard deviation, it is necessary to store i, the sum $\sum x_i$ and the sum of squares $\sum x_i^2$.

More statistical data can be calculated incrementally such as co-variance [62], median and quantiles [63], mode [64] and entropy [65].

Sliding windows are another technique that can be combined with statistical data storage to retrieve deterministic approximate answers. Many applications have greater

interest in recent past data rather than old instances, so there is a priority to store in memory the most recent data, following the windows rules. It also prevents ancient data from overly influencing analysis and statistics. There are several models of sliding windows in the literature [6, 58, 59]:

- **Sequence based**: the window size depends on the number of data points. Landmark windows and Sliding windows are sequence based models:

  - **Landmark windows**: a landmark is a point which is identified as significant in the stream, then posterior data is stored inside the window. The landmarks can be a point desired by the user or a fixed hour on a daily basis (Figure 6).

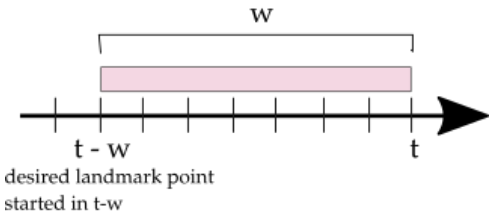

Figure 6 – Landmark window. A landmark point was set at $t - w$, then the data started to be recorded into the window.

  - **Sliding windows**: having a size w, it saves the first w instances that can be retrieved until $(t - w)$ time. When the window is full and the next instance arrives, it is necessary to free memory space excluding the $(t - w - 1)$ point. For example, for a stream with 500 points, $t = 500$, with a window size of $w = 100$, the window points x are $(x_{401}, x_{402}, ..., x_{499}, x_{500})$. When the next point $t = 501$ arrives, the data instance x_{401} is removed from the window (Figure 7).

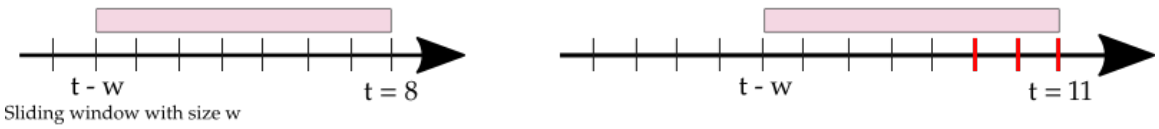

Figure 7 – Sliding window model.

- **Timestamp based**: a timestamp-based window stores all the elements within a period of time t. It is important to notice that the window size is inconstant, so

the number of instances arriving can be estimated, but not previously determined, which may cause lack of memory.

- **Natural Tilted Time Window**: recent data is stored in the window with a very fine granularity, while old data is coarsely stored. The natural time defines what is considered recent and old, depending on the application. For instance, data from one year ago can be stored with low granularity, occupying a small part of the memory. Points from one month ago are not considered so old, so they are accumulated with medium granularity. Finally, data from the same day is considered very recent and is stored with high granularity (Figure 8).

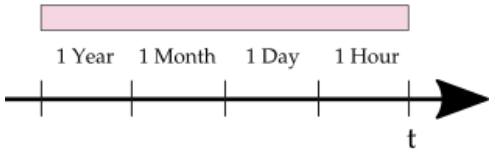

Figure 8 – Natural Tilted Time Window model.

- **Logarithmic Tilted Time Window**: while the Natural Tilted Time Window follows rules from natural time, the Logarithmic Tilted Time Window reduces logarithmically its granularity as data get older (Figure 9).

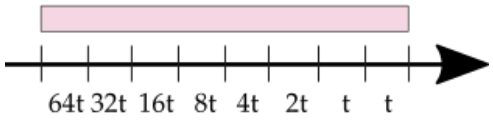

Figure 9 – Logarithmic Tilted Time Window model.

Using a stream model and techniques as described previously, a stream learning algorithm has the following characteristics:

- Its model is updated after processing every incoming data instance, so it learns incrementally.
- A prediction procedure can be executed at any time.

- It has a one-pass constraint, meaning that each instance is made to be read only once or very few times.

Discovering knowledge from streams may rely on different methods such as stream classification, novelty detection, frequent pattern mining, stream clustering and change detection. Next sections will present the methods used in this work, namely stream clustering and change detection.

2.6 Stream Clustering

Clustering is defined as a method applied to a set of data instances to group its similar objects. In other words, applying a data clustering method assigns similar instances to the same group (cluster), while divergent instances end up in different groups. The objective of a clustering method is to maximize the inter-cluster distance, which is the distance between the formed clusters, while minimizing the intra-cluster radius, which is the distance between the instances of the same cluster [14].

The distance between clusters can be measured through the Euclidean distance between their centroids. For example, given a set of data instances $d_1, d_2, ..., d_n$ and two groups g_1 and g_2, the Euclidean distance between their centroids can be defined as:

$$dist = \sqrt{c_{(g_1)}^2 + c_{(g_2)}^2} \tag{2.3}$$

where c is a centroid of a cluster, calculated by:

$$c = \frac{\sum_{i=1}^{n^{(g)}} d_i^{(g)}}{n^{(g)}} \tag{2.4}$$

In equation 2.4, g is the current group, $n^{(g)}$ is the number of instances inside the group and $d_i^{(g)}$ is a data instance in the group.

Using clustering algorithms in data streams is convenient because these algorithms does not need to accumulate data nor be labeled. However, data stream clustering is different from classic clustering due to the data stream characteristics [66], as the ones described in Table 1.

There are several proposals of clustering algorithms in the literature such as K-Means, DBSCAN, OPTICS, BIRCH, CluStream and DenStream etc. Although a considerable number of them use batch data, e.g., K-Means and DBSCAN, there are others created specifically to work with data streams, like CluStream. The latter one will be described next as the stream clustering algorithm used in this work.

	Classic clustering	Stream Clustering
Data arrival	Read the entire Batch data	On-the-fly sequential stream that can arrive at high speed
Processing mode	Offline	Online
Number of passes	As many as necessary	One pass (or very few)
Storage	Entire dataset can be stored	Statistical summary only is stored
Model update	Needs to process the whole dataset	Incremental: it is updated after every data instance processing
Predictions	Made using the whole dataset	Any time
Answer accuracy	Exact	Can be approximated

Table 1 – Comparison of classic clustering with stream clustering methods.

2.6.1 CluStream

CluStream is a continuous data stream clustering framework proposed by Aggarwal et al. [14]. The CluStream online process does not store data instances, but maintains a statistical summary in a micro-cluster structure. Let's consider a data stream consisting of a set of multi-dimensional data instances $\overline{X}_1, ..., \overline{X}_k$ arriving at time stamps $T_1, ..., T_k$. Each $\overline{X}_i$ contains d dimensions denoted by $\overline{X}_i = (x_i^1 ... x_i^d)$. The structure of a micro-cluster is defined as the tuple: $(\overline{CF2^x}, \overline{CF1^x}, CF2^t, CF1^t, n)$, where:

- $\overline{CF2^x}$: sum of the squares of the data values for each dimension;
- $\overline{CF1^x}$: sum of the data values for each dimension;
- $CF2^t$: sum of the squares of *time stamps*;
- $CF1^t$: sum of the *time stamps*;
- n: number of data points in the micro-cluster;

To start the online phase, the algorithm determines the initial clusters. This can be achieved by creating q micro-clusters with one point each, where q is the hyperparameter that sets the maximum number of micro-clusters. These points come from the first q data instances. Then, a unique identification number is assigned to every micro-cluster maintained in the memory. When a new data instance arrives, the algorithm checks whether it is within the *maximum boundary* of a micro-cluster. If so, it is added to the micro-cluster, updating the micro-cluster's tuple through the additive property. Otherwise, a new micro-cluster with only one data instance must be created. To respect the limit of q

micro-clusters, an old one must be eliminated or two must be merged when the creation of a new cluster violates this threshold. To choose between these options, firstly it is verified if any micro-cluster has its *relevance stamp* below the value set in the hyperparameter δ_c. If true, it can be deleted. The relevance stamp stands for the micro-cluster age based on time properties. Finally, if no micro-cluster's relevance stamps are below δ_c, the two nearest micro-clusters are merged.

2.7 Change Detection

Many applications have a continuous flow of data to be analyzed. Such applications can be exemplified by industrial processes, fault detection, surveillance systems, safety of machinery and complex systems, sensor networks, network monitoring and attack detection [6].

Generating processes of data flow produces sequences of observations over the time. A *concept* is a sequence that keeps similar properties within a minimum time of persistence. Observations that are in the same probability distribution at a given moment express the same concept [61].

A concept may change with time. When a concept changes and keeps a minimum durability, it is considered a *concept drift* [6, 67]. It is important to notice that there are differences between *concept drift* and *concept shift*. According to Gama [59], a concept drift is more concerned about gradual changes whereas a concept shift is more related to sudden changes. Also, *outliers* are noises that should not be confused with a concept change. Often sparse, they are not assured to be a new concept. Therefore, the difference between concept and noise is the event duration [6].

The cause of a change come from differences in the collected data properties. So, the concept learned from a given distribution starts yielding inaccurate results. For example, the change may occur in the mean, variance or correlation, as illustrated by Figure 10.

The rate of changes may come in different manners. A drift may occur in four ways [6, 7, 8] and is illustrated by Figure 11:

- *Sudden/Abrupt*: the change in concept happens immediately (e.g., change of resolution in an observed camera, replacement of a sensor, DoS attack in a network).
- *Incremental*: the change takes longer to complete, creating a step shape from one concept to another (e.g., an outdoor camera's luminosity capture during a sunset).
- *Gradual*: the concept shifts back and forth gradually until the shift is complete (e.g., the days that a user played a music from its release until it becomes old, where the

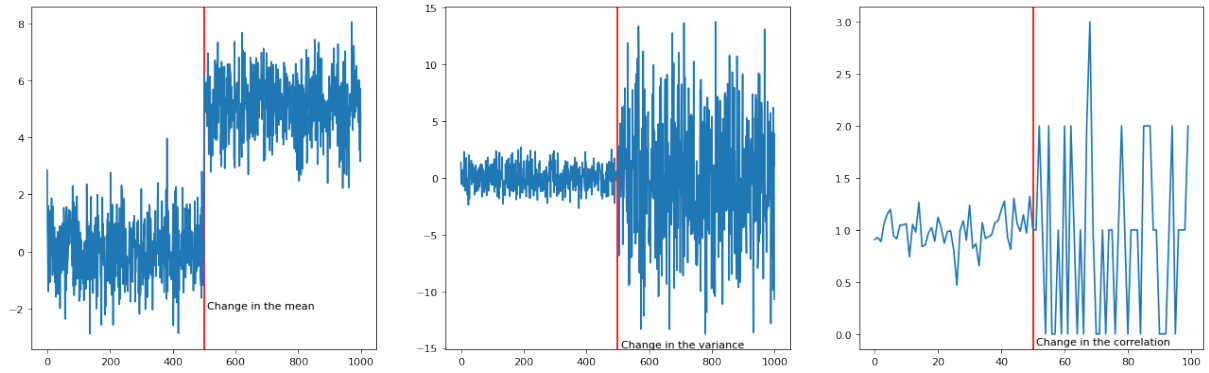

Figure 10 – Change in the (a) mean; (b) variance; (c) correlation. Adapted from [6].

user plays it everyday, but, as the time passes the music gets tiring to listen).

- *Recurring*: a change may occur and revert to its original form after some time (e.g., a sensor that broke and is fixed, a network attack that was deployed and stopped).

Sudden shifts are easier to detect, while an incremental is more challenging. A gradual and slow change can be confused with stable data. On the other hand, multiple and fast changes can prevent the model from learning and be seen as noise by an algorithm [6].

Broadly speaking, detecting novelties, outliers or anomalies, consists of detecting changes in the collected data pattern. However, subtle differences are found in the definitions of these three types of events. Novelties are a new concept that should be addressed by the model, whereas outliers are events that can negatively affect the model and results [68]. Anomalies in turn are considered outliers that are expected by the analyst [69]. It is important to notice that since outliers do not persist, adjusting the concept's minimum time of persistence is necessary to avoid confusing an outlier with an anomaly or a recurring concept.

There are many algorithms that detect changes, like the Cumulative Sum Algorithm (CUSUM) [6], Autoregressive Integrated Moving Average (ARIMA) [68], Adaptive Windowing (ADWIN) [70], Early Drift Detection Method (EDDM) [71] , and the Page-Hinkley Test (a variation of the CUSUM algorithm) [72]. The latter one is discussed in the next section as it is used in this work.

2.7.1 Page-Hinkley Test

The Page-Hinkley Test [72] detects changes based on the mean of a univariate stream. Upon a test, the algorithm considers the cumulative difference between the observed values and their mean up to the current moment [6]. At first, the algorithm needs

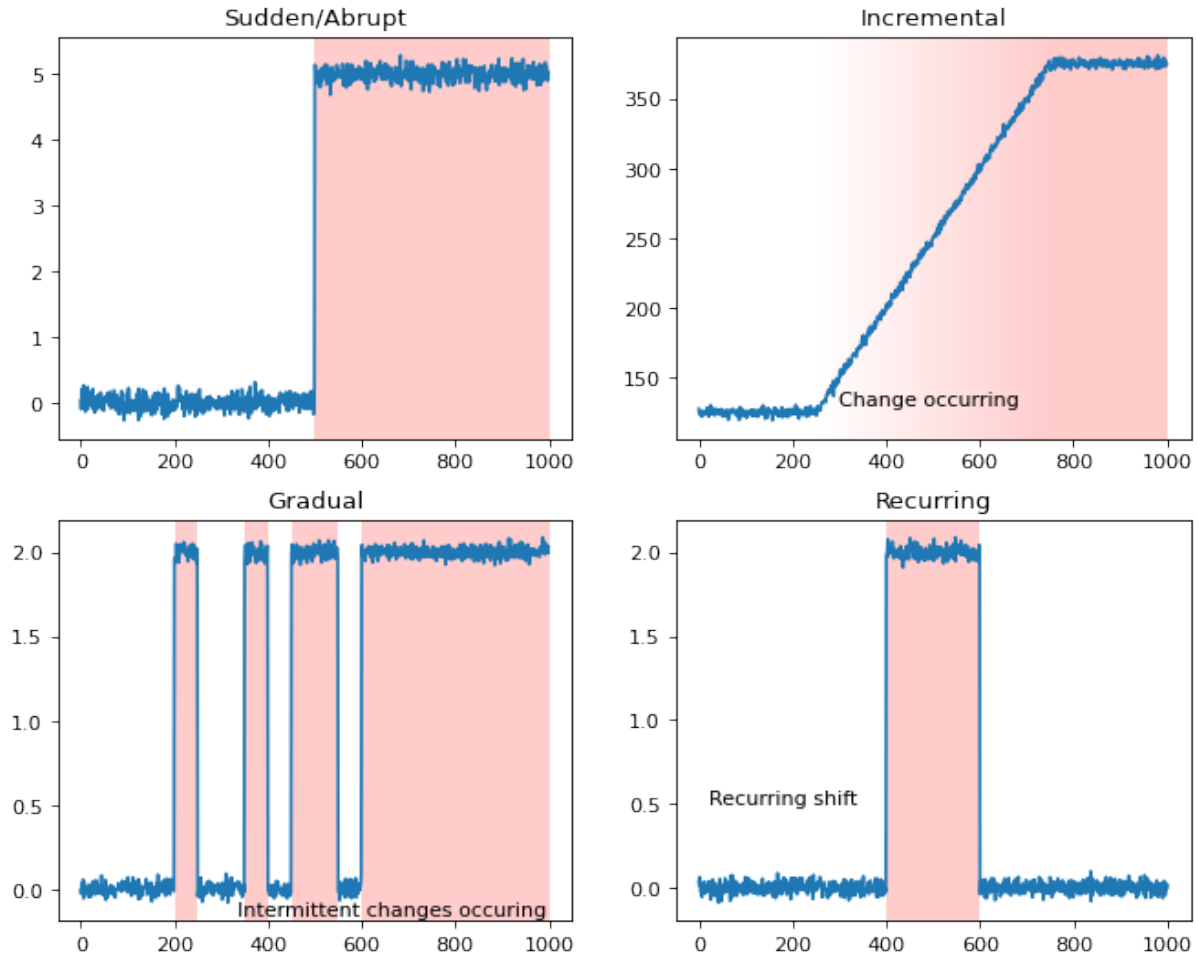

Figure 11 – Types of shifts on the mean. (a) Sudden; (b) Incremental; (c) Gradual; (d) Recurring. Adapted from [7, 8].

four hyperparameters: *min_instances* , δ_{ph}, *threshold*, and α. For each iteration i, the Page-Hinkley algorithm receives the x_i value from the stream. Then a mean M_i is calculated by:

$$M_i = M_i + \frac{(x_i - M_i)}{i} \tag{2.5}$$

Afterward, the cumulative sum of positive changes σ_i^+ is calculated:

$$\sigma_i^+ = max(0, \alpha \cdot \sigma_i^+ + (x_i - M_i - \delta_{\text{ph}})) \tag{2.6}$$

To detect changes in two-sized mode, the cumulative sum of descent values σ_i^- is given by:

$$\sigma_i^- = max(0, \alpha \cdot \sigma_i^- + (M_i - x_i - \delta_{\text{ph}})) \tag{2.7}$$

If a minimum of data instances ($i \geq min\_instances$) were analyzed and the conditions ($\sigma_i^+ > threshold$) or ($\sigma_i^- > threshold$) hold true, an alert is raised. Also, σ_i^+ or σ_i^- are reset.

2.8 Related Work

Multiple studies have been carried out in recent years to tackle attacks against IoT home networks. Supervised machine learning algorithms are frequently found as their central component. Anthi et al. [9] proposed an intrusion detection system for smart home IoT devices that makes use of supervised learning in three different phases. First, supervised algorithms classify the traffic collected in the home router according to the source/destination IoT device. Then, traffic packets belonging to each device are classified as malicious or benign by another supervised model. Lastly, a third supervised model is employed to determine the attack type when a packet is classified as malicious. The proposal was evaluated in a testbed with different types of attacks such as scanning, denial of service (DoS), and man in the middle (MITM).

Moustafa et al. [13] also employed supervised algorithms, but, in their case, the proposal relies on ensemble learning to reach better results. The proposed approach makes use of Adaboost to combine three base classifiers: decision tree, naive Bayes, and artificial neural network. The approach analyzes the collected traffic to detect botnet attacks against HTTP, DNS, and MQTT protocols. Instead of exploring ensemble learning techniques, Brun et al. [20] apply deep learning, which has been widely explored in the intrusion detection area. A model based on dense random neural networks was developed to detect DoS attacks through statistics collected from network traffic in their work.

Detecting unknown attacks can be challenging for supervised algorithms. To address this shortcoming, Wan et al. [73] proposed combining supervised and one-class classification algorithms. Instead of requiring malicious and benign observations for training, a one-class algorithm only needs samples of benign behavior. Thus, it detects attacks by spotting observations that are not similar to the benign ones used for training. In the approach by Wan et al., the network traffic analysis is divided into two steps. Firstly, a supervised algorithm is used to detect known attacks. Then, if the observation is classified as benign, it is inputted to a one-class algorithm for further analysis.

Meidan et al. [17] used only a one-class classification algorithm in their detection approach. More specifically, they applied deep auto-encoders to detect botnets in smart home network traffic. Bezerra et al. [18] also used exclusively one-class classification algorithms to detect botnets. Unlike Meidan et al., they proposed a host-based solution, which monitors the usage of resources like CPU and memory in each device. Multiple algorithms such as isolation forest, one-class SVM (Support Vector Machine), and LOF

(Local Outlier Factor) were tested. In [74], the authors proposed to use a one-class classification algorithm coupled with a reinforcement learning solution. They aimed to address a typical challenge in attack detection: devices' normal behavior can change, and the models must be updated to reflect these movements. Lastly, de Melo et al. [19] proposed an anomaly detection module based on one-class classifiers, using as input network flows. Their architecture solution adds a new layer of security using low-cost devices.

Qaddoura et al. [15] employ a hybrid approach that uses both supervised and unsupervised algorithms to detect intrusion in IoT networks. They proposed a three-stage classification method. First a clustering step based on K-means++ with data reduction is performed. Then, the SVM-SMOTE (Support Vector Machine and Synthetic Minority Oversampling Technique) oversampling method is applied. Data reduction and data oversampling are used to generate balanced training data over an unbalanced dataset. The final stage uses SLFN (Single Hidden Layer Feed-Forward Neural Network) to generate a model for testing. The IoTID20 dataset containing home IoT network traffic was used in the experiments. The attack types detected were DoS, MITM and port scanning.

Ali and Li [21] used multiple levels of deep learning auto-encoders to detect DDoS attacks. The proposed scheme is based on a stack of MDA (Marginalized De-noising Auto-encoder) instances, known as MSDA (Marginalized Stacked De-noising Auto-encoder). MDA is an improved and faster version of SDA (Stacked De-noising Auto-encoder). By combining layers with shallow and deep auto-encoders, which learn in an unsupervised manner, features are generated by encoding the input to a hidden representation. Afterward, the MKL (Multiple Kernel Learning) framework uses the features produced to combine weights to create an unified kernel. Finally, this kernel can be used as a detection model. The datasets used for evaluation were UNB ISCX intrusion detection (IDE 2012/11 and IDE 2012/16) and the UNSW-NB15 from the ACCS (Australian Center for Cyber Security).

Vu et al. [16] proposed a deep transfer learning (DTL) based on a two Auto-encoders (AEs) scheme. While the first AE learns with labeled samples, the second is trained with unlabeled data having the same network arrangement. Therefore, the used dataset can have both labeled and unlabeled samples. This approach was used to transfer information from the first AE to the second AE, because it is harder to label samples and to have a similar performance with unlabeled data. To evaluate the employed approach, they used nine IoT datasets and the results showed that transferring the label information enhances the model when classifying unlabeled data.

Yang et al. [75] employ active learning to detect intruders in wireless IoT networks. In active learning, human expert effort is demanded to label the data to make the ML model yield better results. Then the model is updated by this labeled data. Although there are methods to query labeled data from a human, this proposed approach would

not suit smart home IoT networks due to the lack of user's knowledge, as mentioned before. The datasets used were KDD 1999 and AWID, the latter one being composed of Wi-Fi traffic. The results showed that precision and recall increases with the number of labeled instances.

Instead of focusing on new detection schemes, Vaccari et al. [76] built a novel public MQTT simulated dataset to validate intrusion detection schemes. The dataset is made from data collected from a MQTT broker inside a home IoT network containing a malicious node connected directly to the broker. Their experiments used six ML algorithms containing the attacks: flooding DoS, MQTT Publish Flood, SlowITE, malformed data and brute force authentication.

Following an alternative path to ML, Li et al. [77] proposed a blockchain based IDS. More precisely, their objective is to improve the collaborative intrusion detection systems (CIDSs). In these systems, nodes in a distributed formation exchange security information among themselves, making up a collaborative intrusion detection network (CIDN). A CIDS can successfully detect various kinds of attacks, but are exposed to insider attacks. Malicious nodes can disseminate fake signatures that allow intruders to remain undetected inside the network. To detect and avoid malicious nodes, a CIDS blockchain consortium was proposed. Simulated and real CIDN environments were used to detect worm, flooding and insider exploration threats.

Overall, the reviewed studies are based on batch learning algorithms, which means they are not designed to be frequently updated. Also, they assume that training data, no matter its volume, is integrally stored to build or update the learning model. This can be a burden in smart home environments, since they may not be designed to store and process all this data. Another challenge is the demand for samples for extensive training phases, which may be hard to meet in practical scenarios. In this work, we propose a detection approach based on a stream clustering algorithm, which does not require storing observations for training and can learn incrementally, being continuously updated. Table 2 outlines the related studies showing the machine learning algorithms, detected attack types, evaluation strategy and the kind of data source.

Author	**Learning method**	**Detected attack types**	**Evaluation**	**Data source**
Anthi et al. [9]	9 supervised algorithms	Scanning, DoS/DDoS, MITM, Replay, Spoofing	Testbed	Network traffic
Moustafa et al. [13]	Supervised + ensemble learning	Botnet attacks	Simulation	Network Traffic

Brun et al. [20]	Deep learning	DoS	Testbed	Network Traffic
Wan et al. [73]	Supervised + one-class classification	Scanning, flooding, brute-force and data link layer attacks	Testbed	Network Traffic
Meidan et al. [17]	Deep auto-encoders	DDoS (Mirai and BASHLITE botnets)	Testbed	Network Traffic
Bezerra et al. [18]	One-class classification	Botnet attacks	Testbed	Temperature, CPU and memory usage
de Melo et al. [19]	One-class classification	DDoS, Replay attack, Eavesdropping	Testbed	Network flows
Heartfield et al. [74]	One-class classification + reinforcement learning	Physical, data link, network and application layers attacks	Testbed	Network traffic
Qaddoura et al. [15]	Unsupervised and supervised: K-means++, SLFN-SVM-SMOTE	DoS, MITM and Scanning	Simulation	Network traffic
Ali and Li [21]	MSDA and MKL	DDoS	Simulation	Network traffic
Vu et al. [16]	DTL + AEs	DDoS, scan, Junk, UDP and TCP flooding and combo	Simulation	Network traffic
Yang et al. [75]	Active learning	Unspecified	Simulation	Network traffic
Vaccari et al. [76]	6 supervised ML algorithms	Flooding, malformation, SlowITE and brute force	Simulated testbed	Network traffic
Li et al. [77]	Signature based	Worm, flooding and insider exploration	Simulated and real environments	Network traffic

Table 2 – Summary of related work features.

3 ATTACK DETECTION MODEL FOR SMART HOME IOT DEVICES

In this chapter, we present our approach to detect attacks in smart home IoT networks. IoT devices are known to have limited hardware resources, like low memory, storage and processing. They execute repeated and strict tasks based on the owner's daily life. To propose an approach in a smart home context, these are important features that should be considered. Also, it stimulates the creation of an online, unsupervised and with low memory requirement model. The proposed model in this work does not need to accumulate data (only a summary is stored), unlike batch solutions that may demand high memory consumption. Furthermore, we use a one-pass constraint algorithm that demands less memory space and processing power [78]. Finally, the model learns incrementally, thus it does not become obsolete and does not demand an external actor or routine to get updated.

Changes in the network traffic pattern can occur when there is an acknowledged change in a device's configuration or the occurrence of an event (e.g., change in a camera's capture frame rate, or a party at home that makes a smart door lock and a virtual assistant be used much more frequently). These changes can also happen in an unacknowledged form when an intruder installs malicious software in IoT devices or try to perform any other attack. Our approach is a detection scheme based on the hypothesis that attacks against an IoT device will generate a detectable sudden disturbance in the observed traffic.

The proposed scheme analyzes the inflow packets of each device by inspecting their headers. In a home IoT environment, packet inspection is possible due to the simplicity of the devices and their lower transmission speeds. Although some devices may require more network bandwidth (e.g., high resolution cameras, smart TVs, baby monitors, etc.), others such as door locks, smoke detectors, air conditioners and washing machines use only telemetry data. In a larger corporate network with higher traffic volume, packet inspection would be more challenging.

During the inspection, an online clustering algorithm clusters the packets. We suppose that when malicious packets arrive, they will be assigned to different and more distant clusters than the normal ones, causing a disturbance into the distance between cluster's centroids as illustrated in Figure 12. Hence, the distance between the centroids is monitored by a change detector, which is used to generate an alert when a new and distant cluster emerges.

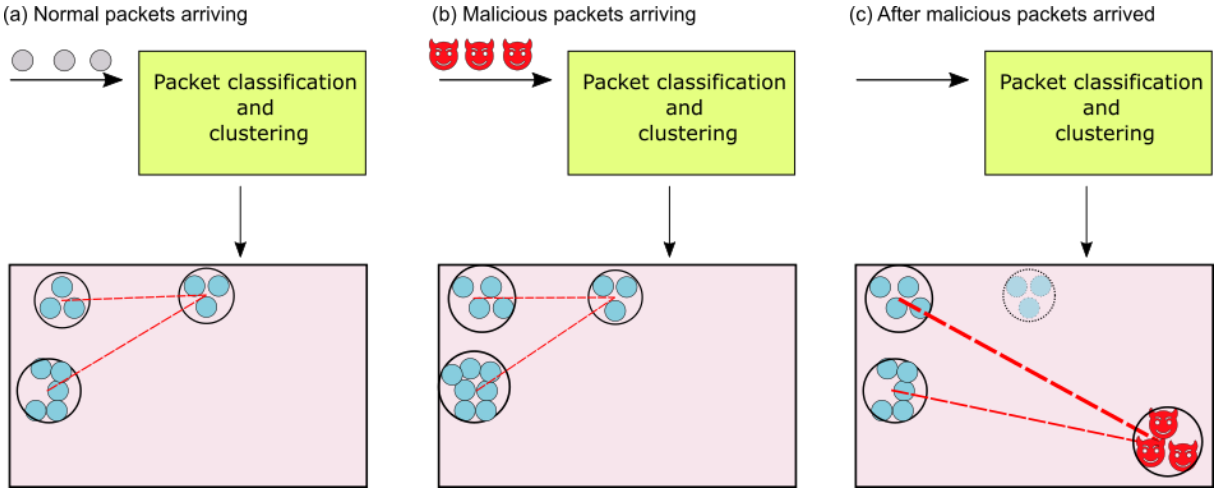

Figure 12 – Packet header classification and clustering in three moments. (a) Normal packets arrive and are clustered. The most distant cluster from each other is indicated by the red dotted line. (b) Packets with similar header will cluster together. The system is in imminence of an attack. (c) Malicious packets will be clustered together and may be distant from another ones. Also, other clusters can be eliminated in the process.

3.1 Packet Header Clustering and Change Detection

The first step consists of breaking the incoming stream of a device into three different streams, classifying each packet according to the protocol on top of the IP header in the protocol stack. For this study, we considered TCP, UDP, and ICMP protocols, and Figure 13 illustrates all the steps explained next. To separate the packets by protocol, the field Protocol in the IP header was used, where values 1, 6 and 17 correspond to ICMP, TCP and UDP protocols respectively [79].

The clustering algorithm will be applied to each packet individually, i.e., each data instance represents a packet. As TCP, UDP and ICMP headers have different fields, we decided to dedicate an exclusive clustering instance for each stream. The features extracted from the four headers (IP, ICMP, TCP, and UDP) for the clustering process are presented in Table 3 and Table 4. Broadly speaking, these fields were selected because they are frequently linked to different attacks and can potentially help divide packets into benign and malicious. More specific reasons for selecting or not each field of the IP, ICMP, TCP, and UDP headers as features are discussed next.

Features *len, ip.flags.df, ip.flags.mf, ip.ttl, ip.frag_offset* are extracted from all the three streams. *len* represents the whole packet length and can be found in any packet, no matter the involved protocols. This feature was chosen because malicious packets usually present anomalous length. Fields *ip.flags.df*, *ip.flags.mf* and *ip.frag_offset* are linked to the IP fragmentation process and can indicate fragmentation based attacks. The *ip.ttl* field

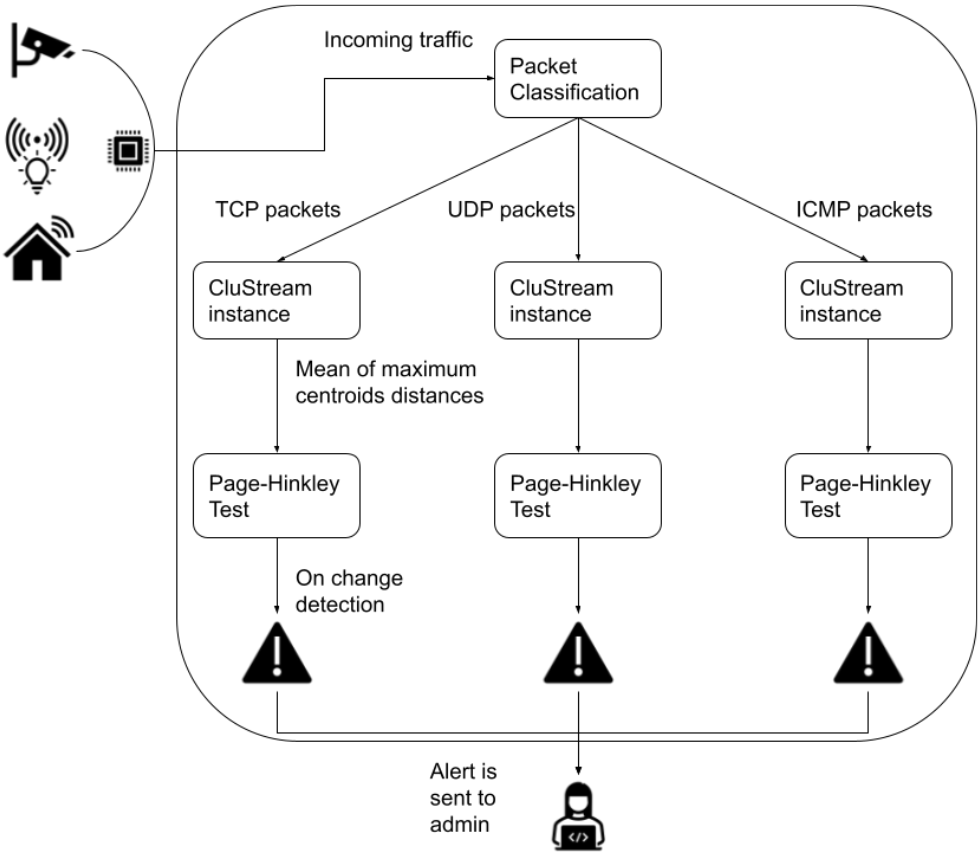

Figure 13 – Proposed Model to detect attacks in Smart Home IoT Networks.

shows the packet time to live (TTL), which is counted in hops. The operating system that builds a packet is in charge of determining its initial TTL, and every operating system has a default value for this field. Anomalous TTL values may signal packet modification or the use of some special (and malicious) software to generate packets.

The other features presented in the IP header (*ip.version, ip.hdr_len, ip.dsfield.dscp, ip.dsfield.ecn, ip.src, ip.dst, ip.len, ip.flags, ip.flags.rb, ip.proto, ip.checksum.status*) were not chosen because they are not frequently affected by attacks and would not contribute to the clustering process. More specifically, the IP version for most packets is 4 (IPv4) [80], so the *ip.version* feature is often the same for all packets. IP-specific length features (*ip.hdr_len, ip.len*) are not necessary since we already have the length of the whole packet in the *len* feature. The *ip.dsfield.dscp* and *ip.dsfield.ecn* fields are used by the Differentiated Services method, also known as *DiffServ*. DiffServ is divided into Differentiated Services Code Point (DSCP) and Explicit Congestion Notification (ECN) [81]. DSCP allows *quality of service* (QoS) in IP networks by signaling the priority of a packet (e.g., packets from a real time video conference), but it does not bring information of interest for our work. ECN is a specific field where a host can notify the congestion of a network. In our case, this field will not fit in. The features *ip.src* and *ip.dst* carry the source and

destination IP addresses, which are specific to each packet and do not give any clue of malicious pattern, unless the detection system makes use of a block list. Among IP flags, the only one not used is *ip.flags.rb*, meaning Reserved Bit, which is always 0. Finally, Checksum (*ip.checksum.status*) is a field to check errors in the header. The host or router calculates the received packet checksum and compares it to the checksum recorded in the field. If they do not match, there is an error in the header and the packet is discarded. The detected errors are not necessarily linked to malicious activity. Particularly in wireless networks, noisy channels or other transmission problems cause errors frequently. For this reason, this field was not selected.

The selected features *tcp.dstport, tcp.flags.syn, tcp.flags.ack* and *tcp.flags.push* are exclusive to TCP. The destination port (*tcp.dstport*) is an important feature. When requests are made to IoT devices, the destination port helps to identify the service that is being consumed by an external host. TCP flags like SYN, ACK and PUSH are also relevant because they are modified or explored in different attacks. Other flags such as *tcp.flags.ns, tcp.flags.cwr, tcp.flags.ecn, tcp.flags.urg, tcp.flags.reset, tcp.flags.res,* and *tcp.flags.fin* are not as much linked to attacks as the other ones mentioned above, so they were not picked.

Other TCP header fields were not picked due to the following reasons. The source port (*tcp.srcport*) from the sender does not give information about an attack because it is often random. Similarly to *ip.len*, TCP length-related fields (*tcp.len* and *tcp.hdr_len*), bring redundant information and are not needed. Some fields have values that are specific for each TCP connection and do not contribute to detect attacks when packets are analyzed individually: *tcp.window_size_value, tcp.window_size, tcp.window_size_scalefactor, tcp.seq, tcp.ack*. The Checksum status (*tcp.checksum.status*) works in the same way as the IP header checksum and was not chosen for the same reason.

In UDP packets, the only field considered to provide useful information is the destination port (*udp.dstport*). Source port (*udp.srcport*), length (*udp.length*) and checksum status (*udp.checksum.status*) were not selected. The reasons for these decisions were the same presented to the TCP header fields that have analogous functions.

A single feature was also considered in ICMP packets. The ICMP code (*icmp.code*) is used to report events, errors and to test networking conditions [82, 83]. It can also help identify DoS/DDoS attacks, particularly the ones that explore ICMP echo packets, known as ICMP Flood attacks. The *icmp.type* points to the type of control message, describing the super category of the ICMP code, thus this field is not necessary. The ICMP Checksum (*icmp.checksum.status*) is discarded for the same reason as the IP, TCP, and UDP Checksum fields. Other ICMP features like *icmp.ident, icmp.seq, icmp.seq_le, icmp.resp_in, icmp.resp_to* were not included because their values depend on the ICMP type and code provided.

Having discussed the choices about features, we can proceed to describe how they

are processed in the proposed scheme. Before going through clustering, all the features are transformed using min-max. To this purpose, we consider the protocol specification limits for each header field. For example, let's consider the feature *tcp.dstport*: its minimum value $x_{\min}$ is 0 and its maximum value $x_{\max}$ is 65535. To transform this feature into the $[0, 1]$ interval, given a feature value x , the transformed value x_{norm} is calculated as in (3.1). An overview of the first steps involving packet organization into three streams and feature transformation is given by Algorithm 1.

$$x_{\text{norm}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \tag{3.1}$$

Common features	Description
len	Datagram length in bytes
ip.flags.df	Indicates if the datagram can be fragmented.
ip.flags.mf	Indicates if the datagram is the last fragment or there are more fragments.
ip.ttl	Indicates the datagram's time to live. Once it reaches 0, it must be destroyed.
ip.frag_offset	Fragment position in the entire datagram.

Table 3 – Common features used in TCP, UDP and ICMP streams.

TCP features	Description
tcp.dstport	TCP datagram destination port.
tcp.flags.syn	Synchronization flag to start the synchronization of the sequence numbers.
tcp.flags.ack	Acknowledgement flag indicating that a connection is established.
tcp.flags.push	Push flag asking that data should be pushed to the application.
UDP features	**Description**
udp.dstport	UDP datagram destination port
ICMP features	**Description**
icmp.code	ICMP control message code.

Table 4 – Exclusive features used in TCP, UDP and ICMP streams.

After classifying the packets according to their protocols, the clustering algorithm is applied to each stream to create separated clusters for malicious and benign packets.

Algorithm 1 Packet Classification

Require:
$x[i]$: Data instance from the stream.
Ensure:
$x[i]^{\text{norm}}$: Classified and transformed data instance with extracted features.
function PacketClassification($x[i]$)
$f = [len, ip.flags.df, ip.flags.mf, ip.ttl, ip.frag\_offset]$
//specific features
$f^{\text{tcp}} = [tcp.dstport, tcp.flags.syn, tcp.flags.ack, tcp.flags.push]$
$f^{\text{udp}} = [udp.dstport]$
$f^{\text{icmp}} = [icmp.code]$
//ip.proto is the protocol on top of IP header in $x[i]$
$p = x[i][ip.proto]$
Add f^p in f
$x[i]^f$ = Extract features in f from $x[i]$
$x[i]^{\text{norm}}$ = Transform($x[i]^f$)
return ($x[i]^{\text{norm}}$)
end function

The online part of the CluStream framework [14] was chosen for the clustering step because it is focused on data streams and seeks to provide high quality clusters over time, so outdated data in the stream does not dominate the model. Also, it is one of the first stream clustering algorithms, which generated several variants and has excellent reliability in results reported for other types of streams [84, 85, 86]. It stores very little data with a growth factor of only $log_2(T)$, where T is the time elapsed since the beginning of the stream. The one-pass constraint is addressed and the formed clusters are known as *micro-clusters*, calculating their centroids and analyzing the distances between them is straightforward as described in Section 2.6.1.

To run a CluStream instance, the hyperparameters q, h, t, r, m and δ_c need to be set, where:

- q is the maximum number of micro-clusters;
- h is a time horizon considered to find micro-clusters within $(i - h, i)$, where i is the current time.
- t: cluster's maximum boundary factor, which is a factor of the root mean square (RMS) deviation of the data points in the cluster from the centroid;
- r: heuristic maximum boundary factor. For a micro-cluster with only 1 point, the maximum boundary is r times the maximum boundary of the next closest cluster;
- m: cluster's last data points. They are used to approximate the average time-stamp;
- δ_c: a threshold to eliminate a cluster. If the least relevance stamp of a cluster is below δ_c, the cluster can be erased, and a new cluster can be created.

To start using the CluStream instance, a set of initial q data instances is given to the micro-clusters, in order to create q micro-clusters with one point each. After that, as new data instances arrive, the CluStream algorithm will maintain the micro-clusters adding each data instance to a cluster within the maximum boundary factor, eliminating, creating, or merging clusters. Algorithm 2 shows the CluStream initialization process.

Algorithm 2 CluStream Instance Initialization

Require:

$x[i]^{\text{norm}}$: Classified and transformed data instance with extracted features from Packet Classification Algorithm.

q : CluStream maximum number of micro-clusters.

Ensure:

$\overline{M}$: Initial micro-clusters.

1: **function** CluStreamInstanceInitialization($x[i]^{\text{norm}}$, q, h)
2: **if** $i < q$ **then**
3: Add $x[i]^{\text{norm}}$ to $\overline{M}[i]$
4: **end if**
5: **return** ($\overline{M}$)
6: **end function**

Every time the CluStream instance clusters a new packet, the mean of the maximum distances among the cluster centroids is calculated as follows. Only the q' micro-clusters with new data instances assigned to them in the time horizon h are considered. First, the Euclidean distance for every pair of these micro-clusters centroids is calculated. The centroid of a micro-cluster m is calculated by $\overline{CF1}^{x}{}_{i}/n_i$. The calculated distances are stored in a matrix D with dimensions $q' \times q'$, where each position $d_{r,c}$ denotes the distance between the centroids of the micro-clusters r and c. For each row in D, we pick the maximum distance, denoted as $mdist[row]$. It represents the maximum distance of the centroid of the micro-cluster m to any other micro-cluster's centroid in D. Finally, we calculate the mean of all maximum distances $\overline{mdist}$ as in (3.2).

$$\overline{mdist} = \frac{\sum_{row=1}^{q'} mdist_i}{q'} \tag{3.2}$$

The calculated means of the maximum distances forms a univariate series that feeds the respective instance of a change detector. The Page-Hinkley Test was chosen as the change detector algorithm due to its capacity of detecting sudden changes in univariate series. It is an online algorithm, which meets the one-pass constraint and has been widely applied as a solid general-purpose change detector [87, 88, 89, 90]. When it observes a data instance, a stored mean is updated for the current moment. Having this mean broken a threshold, it detects an abrupt change, so the system administrator is alerted. The hyperparameters $min\_instances, \delta_{\text{ph}}, threshold$ and α for the Page Hinkley Test must be set and are crucial to determine if an alert will be raised or not. Algorithm 3 shows how CluStream and Page-Hinkley work after the CluStream instance initialization.

Algorithm 3 Stream Model Update

Require:
$x[i]^{\text{norm}}$: transformed data instance from the stream.
$\overline{M}$: Current micro-clusters from Clustream Initialization or CluStream Update.
q, h : CluStream hyperparameters.
$min\_instances, \delta_{\text{ph}}, threshold, \alpha$: Page-Hinkley Test hyperparameters.
Ensure:
A: Alert is true if change is detected, else false.
$\overline{M}$: Updated micro-clusters.
function StreamModelUpdate($x[i]^{\text{norm}}, \overline{M}, q, h, min\_instances, \delta_{\text{ph}}, threshold, \alpha$)
//Update the micro-clusters.
    $\overline{M} = CluStreamUpdate(x[i]^{\text{norm}}, i, \overline{M})$
    Keep in $\overline{M'}$ only q' micro-clusters modified between $(i - h)$ and i
//D is a matrix $q' \times q'$ containing the distances between centroids
    $D = CalculateEuclideanCentroidsDistances(\overline{M'})$
    for all *row* in D **do**
        $mdist[row] = Max(row)$
    end for
    $\overline{mdist} = Sum(mdist)/q'$
//Page-Hinkley Test applied in the mean of maximum centroids distances
    $A = PageHinkleyUpdate(\overline{mdist}, min\_instances, \delta_{\text{ph}}, threshold, \alpha)$
    return $(A, \overline{M})$
end function

Summing up, our hypothesis is that when a device is attacked, the involved packets will be clearly distinct from the other ones. When CluStream clusters a malicious packet, either a new micro-cluster is created, or this packet is assigned to an existing cluster. In the former possibility, the new micro-cluster is likely distant from the other micro-clusters since it represents a new and anomalous behavior. In the latter, the micro-cluster that received the malicious packet will have its centroid significantly changed. In both cases, the mean of the maximum distances between the micro-cluster centroids will be affected. By monitoring this indicator with the Page Hinkley Test, we can detect when it presents a sudden and significant change.

3.2 Detection System Deployment

In this section, we discuss different possibilities to deploy the proposed scheme in a smart home IoT environment. Four deployment methods could be applied:

- **Host-based deployment**: each device would host its own detector instance, and the traffic data would be captured and analyzed inside the device. This deployment method would have two benefits. The first one is to preserve user privacy as traffic would not be shared with other devices for monitoring purposes. The second benefit is related to a particularity of our proposal, which analyzes packets device by device. In the host based deployment, the detection system would not need to handle packets

belonging to different devices the same way it would if the packets were collected in a network router. Although IoT devices are resource-constrained, this deployment method might be viable since the proposed scheme does not require model training. Finally, in devices that use non-IP communication (e.g., Bluetooth), the proposed detector could not be directly applied.

- **Router-based deployment**: a smart home could have the proposed detector installed in its Wi-Fi router or any other point where there is data concentration. This kind of device is usually more robust in terms of computational capacity than other domestic IoT gadgets. Additionally, they handle TCP/IP traffic even for gadgets that use non-IP communication protocols as they may work as gateways for those gadgets. Nonetheless, an extra step would be necessary to the proposed scheme: traffic belonging to different devices would have to be classified accordingly, since the proposed scheme analyzes packets device by device. Filtering by IP or MAC addresses is the first solution that comes to mind, but it is a limited option. Spoofing attacks or dynamic address assignment could easily lead to classification errors.

- **Edge-based deployment**: the traffic handled by a router or other concentration point would be mirrored to a dedicated device. This host could be an edge device, being more powerful than a Wi-fi router. Moreover, it would not need to be deployed in the same network, although this could bring privacy concerns. Another challenge of this method is the same as the router-based deployment: an extra step to classify traffic by device is required.

- **Hybrid-based deployment**: the proposed scheme would run in IoT devices that could support it and send the results to the router, concentration point or edge device when an attack is detected. For devices that do not support the detector instance, a centralized device would take care of the process as in the router-based or edge-based deployment. The hybrid-based deployment method would combine the benefits and challenges of the other cited methods.

4 EVALUATION AND RESULTS

In this chapter, the proposed approach is evaluated. We start by presenting the experimental design, containing information about the dataset, implementation and evaluation metrics. Next, we show the results related to the proposal's efficacy in detecting attacks. Finally, three examples are discussed to provide more details about the approach's operation.

4.1 Experimental Design

To evaluate the proposed approach, we used the dataset created by [9] as a starting point. This dataset consists of packets captured in multiple smart home devices like smart plugs, cameras, and device hubs. Different attacks such as scanning, DoS, and MITM were launched against these devices, with the packets being labeled accordingly. To use the dataset in our evaluation, we first organized it by device and protocol (TCP, UDP, and ICMP). Then, we observed that the tests that originated the dataset followed a common pattern. For each device, the traffic starts with a long sequence of normal traffic, followed by a sequence of attacks. There are no situations that intersperse periods of normal traffic and attacks. As our proposed approach is based on incremental learning, it is important to observe its behavior when the traffic changes from normal to attack and vice-versa.

To address this issue, we augmented the original dataset. By following the dataset labels, the moments when attacks started and ended were identified. Also, we found the moments when the monitoring was turned off or restarted. This was possible through the analysis of the packet's timestamps. In some points, there are gaps between packets' timestamps that clearly indicate a pause in the monitoring, which is expected in experimental environments. Using these moments as references, we extracted self-contained periods of normal and attack traffic, making a collection of them. From this point on, these self-contained periods are referred to as clippings. Then, from this collection of clippings, we made scenarios that combine normal and attack traffic at different moments. The scenarios used in this study are illustrated by an example in Figure 14 and their details, like clipping order, length and attack type, are presented in Appendix A. Figure 14 illustrates the scenario for a device called Hive Hub in the TCP stream. The clippings #1, #2, #3, #5, #7 and #9 are made of benign traffic. Clipping #4 contains malicious DoS packets, while #6 is made of MITM packets and #8 is made of scanning attacks. In total, we made 11 scenarios, where each scenario is a stream characterized by device + protocol, as Table 5 presents. This table also provides details about the amount of packets (length) and the normal:attack ratio for each scenario. The dataset consisting of these

scenarios is available on the Web[1].

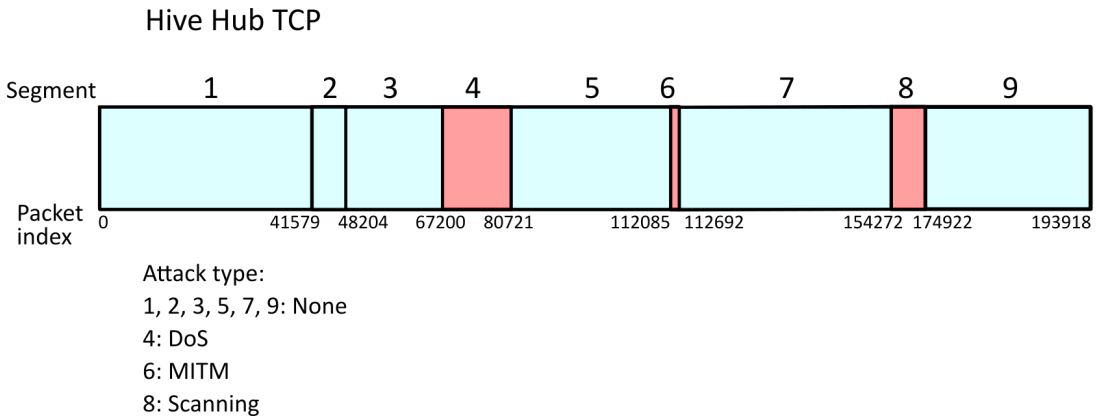

Figure 14 – Dataset clippings that make up the Hive Hub TCP scenario.

Device name	Protocol	Attack Type	Length	Unbalance Ratio
Hive Hub	TCP	DoS, MITM and Scanning	193919	4.6:1
	UDP	MITM and Scanning	4464	10.2:1
	ICMP	DoS and MITM	36248	51:1
Samsung Smart Things	TCP	DoS, MITM and Scanning	122833	3.5:1
	UDP	MITM and Scanning	10612	30.2:1
Lifx Smart Lamp	TCP	DoS and MITM	64199	5.4:1
	ICMP	MITM	89073	164:1
TP-Link NC200 Camera	TCP	DoS, MITM and Scanning	39848	2:1
	UDP	DoS, MITM and Scanning	8549	3.4:1
TP-Link SmartPlug	TCP	DoS	8896	12.6:1
	UDP	DoS and IoT-Toolkit	12477	0.34:1

Table 5 – Description of the experimental scenarios.

After the data preparation, we implemented the proposed approach using the Python programming language. To implement the CluStream module, the code from River framework [64] repository[2] was used as a starting point. Some changes were added to this code, including the calculus of the mean of maximum centroid distances. As for

[1] https://github.com/fernando-nakagawa/iot_datasets
[2] https://github.com/online-ml/river/blob/main/river/cluster/clustream.py

the Page-Hinkley Test, we made use of the River's Page-Hinkley implementation[3], with modifications to detect changes in two-sided mode.

Finally, the metrics used to evaluate the proposed approach are:

- **Detection Rate**: proportion of how many attacks were detected. If an attack goes unnoticed by the system, this value gets lower. This metric ranges from 0 (worst result) to 1 (best result);
- **Precision**: $TP/(TP + FP)$: the percentage of alerts that were true. TP stands for true positives and FP stands for false positives. The sensitivity of the system should be well-balanced to avoid the generation of false alarms or no alarm when an attack begins. It is important to note that this metric measures the accuracy of detecting attack events rather than the accuracy of detecting malicious packets.
- **Average Delay**: the average delay of all detected attacks. Delay is calculated by counting the iterations between the beginning of an attack and its detection. Each iteration corresponds to a new packet joining the stream.

4.2 Results

CluStream has its hyperparameters q, h, t, r, m and δ_c as described in section 3.1. The CluStream implementation used was programmed to left the CluStream hyperparameters t, m and δ_c with their default values. h, q and r can be respectively set as:

- *time_window*: the algorithm will consider only data that arrived *time_window* iterations before the current processing time.
- *max_micro_clusters*: determines the maximum number of micro-clusters to be used by the CluStream algorithm.
- *micro_cluster_r_factor*: it is the multiplier for the micro-cluster radius, which determines the cluster's *maximum boundary*. A new data instance needs to be located within a cluster's maximum boundary to be added into it.

Values for the hyperparameters were standardized for all scenarios. To choose the values, we executed the CluStream algorithm with different value combinations. The used values for *time_window* were: 100, 500 and 1000. For *max_micro_clusters*, we applied: 10, 50, 100 and 150. The *micro_cluster_r_factor* was fixed with its default value: 2. Then, we analyzed the purity of the micro-clusters produced by CluStream for each set

[3] https://github.com/online-ml/river/blob/main/river/drift/page_hinkley.py

of hyperparameter values. The purity of a cluster is determined by the homogeneity of its data instances in terms of their classification. A pure cluster is one that has only benign or malicious packet data. The hyperparameter value set that produced the highest number of pure clusters were adopted. More specifically, the best combination found was $time\_window = 100$ and $max\_micro\_clusters = 50$.

As for the Page-Hinkley Test hyperparameters, firstly, we used the default values: $min\_instances = 30, \delta_{\text{ph}} = 0.005, threshold = 50, \alpha = 0.9999$. Then, different values were analyzed to find the best setting in terms of precision and detection rate. The tested values for *min_instances* were: 10,30, 300, 500, 750, 1000, 2000, 3000, 5000 and 10000. For δ_{ph}, we tested: 0.0005, 0.005, 0.05, 0.04, 0.03, 0.02, 0.01 and 0.5. The thresholds analyzed were: 0.01, 0.1, 0.5, 1, 10, 100, and 200. In α values, we tested: 0, 0.3, 0.6, 0.9, 0.99, 0.999 and 1. To improve the results in some scenarios, we found better values in between the determined values described.

Table 6 contains the Page-Hinkley's hyperparameters values used, and, in Table 7, we present the results for each scenario, detection rate, precision and average delay in iterations.

Device name	Protocol	*min_ instances*	δ_{ph} value	*threshold*	α value
Hive Hub	TCP	10	0.005	200	0.999
	UDP	30	0.0005	5	1
	ICMP	300	0.0005	5	1
Smart Things	TCP	30	0.05	1.2	0.9
	UDP	2000	0.005	0.8	0.9
Lifx	TCP	300	0.005	1	0.5
	ICMP	10000	0.01	0.01	0.2
TP-Link Camera	TCP	3000	0.05	5	0.9
	UDP	500	0.021	0.1	0.999
TP-Link Plug	TCP	30	0.5	100	0.99
	UDP	750	0.005	0.5	0.99

Table 6 – Hyperparameters used in Page-Hinkley Test.

After analyzing a total of 591,118 packets in 11 scenarios, the overall detection rate is above 92%, precision is approximately 81% and the average delay is near 151 iterations. Average delays did not follow a pattern. In some streams, attacks were detected very quickly, while others took longer. In most cases, the proposed approach yield better results to TCP streams than to UDP or ICMP ones.

In the Hive Hub UDP, Samsung Smart Things UDP and Lifx Smart Lamp ICMP scenarios, streams presented a very irregular behavior even in normal traffic periods, causing frequent changes in the mean of the maximum distances. The Lifx Smart Lamp

Device name	Protocol	Detection rate	Precision	Average Delay
Hive Hub	TCP	66.6%	100%	377.5
	UDP	100%	50%	98
	ICMP	100%	100%	36.5
Smart Things	TCP	100%	100%	2.3
	UDP	50%	25%	152
Lifx	TCP	100%	100%	1.5
	ICMP	100%	12.5%	66
TP-Link Camera	TCP	100%	100%	332
	UDP	100%	100%	56.7
TP-Link Plug	TCP	100%	100%	288
	UDP	100%	100%	252

Table 7 – Results for each device and protocol.

ICMP scenario also had one MITM attack, which we considered the hardest one to detect because it spoofs legitimate messages. The MITM attack was the only one not detected in Hive Hub TCP and Smart Things UDP scenarios.

Conversely, seven out of eleven scenarios reached 100% of detection rate and precision with TCP streams accounting for the vast majority. Considering the attack types DoS, scanning and IoT-Toolkit, the detection rates were 100% for all the scenarios. It is also noteworthy that our approach was able to detect all the different attack types: DoS, scanning, MITM, and IoT-Toolkit. The results show that CluStream and the mean of maximum distances among centroids provide a good indicator of attacks. Moreover, Page-Hinkley Test could detect these changes precisely.

4.3 Illustration Examples

To better illustrate how the proposed approach works, we present more details on how the attacks were detected in three scenarios. More specifically, the objective is to show through these examples how the mean of maximum distances reacts to different attacks and how the proposed approach uses this indicator to detect these attacks. Also, we show how the CluStream algorithm creates, deletes and merges micro-clusters in normal and malicious behaviors.

The scenario with TCP stream in the Samsung Smart Things Hub is presented in Figure 15. Until packet #42816, the traffic is normal, and the mean of the maximum distances present small and regular variations. After #15418 and #21201, there is a change of dataset clippings containing normal behavior and a very subtle change occurs. A DoS attack starts with packet #42817. Suddenly, the mean of maximum distances raises and then drops, signaling that there is a predominant behavior in this stream now, which makes

the micro-clusters stay very close to each other as they are updated or created with DoS packets. As a consequence of the change, an alert is quickly raised. After packet #48768, the traffic goes back to normal, so the centroids move back to the previous pattern, thus generating two alerts that are not considered false positives since they are related to the attack. Next, a MITM attack occurs between packets #64188 and #64848, being properly detected by the approach. Finally, from packet #86464 to #107413, a scanning attack occurs, firing various alerts. It is noticeable that the varying nature of scanning attacks generated an irregular pattern in the mean of maximum centroid distances.

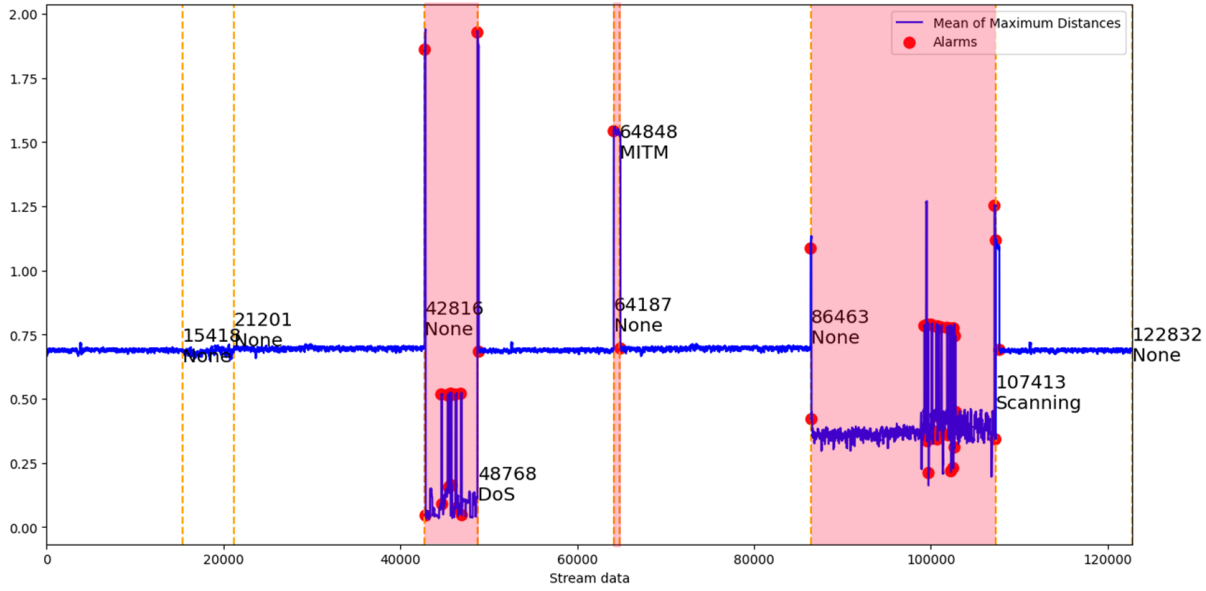

Figure 15 – TCP stream in Samsung Smart Things Hub: mean of maximum centroid distances and generated alerts.

Figure 16 shows cumulative cluster creations, deletions and merges performed by CluStream for the Samsung SmartThings TCP stream. It is possible to notice that between normal data transitions, no significant changes occur. During the DoS attack, there is a slight increase in the amount of creations and merges, while the number of deletions decreases a little. The behavior remains constant during the MITM attack. At the beginning of the scanning attack, the number of clusters created and merged increases significantly and after the attack, the slope goes back to its previous pattern. Since DoS and scanning attacks have distinct behaviors from normal traffic, these results suggest that the algorithm creates micro-clusters to assign the malicious packets, while older normal traffic micro-clusters are deleted and merged. Also, these results suggest that when the number of creations or merges shows a significant increase, irregular patterns are generated in the mean of maximum centroid distances.

Figure 17 depicts the scenario with ICMP stream in the Hive Hub. From the beginning to packet #18368, the traffic is normal, and we can observe that the mean of

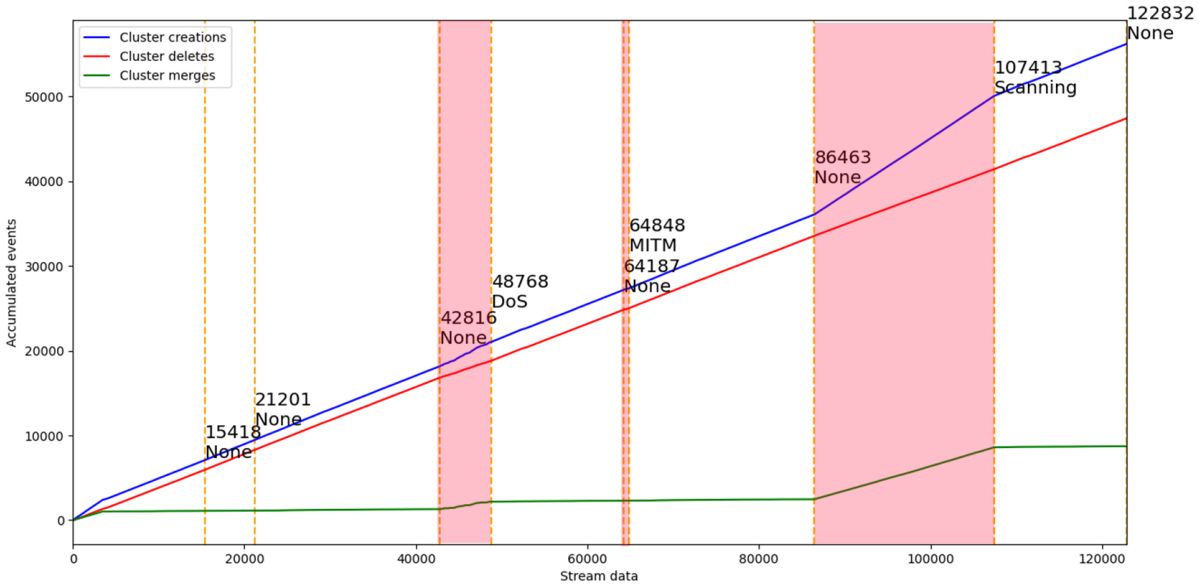

Figure 16 – TCP stream in Samsung Smart Things Hub: cluster creations, deletions and merges.

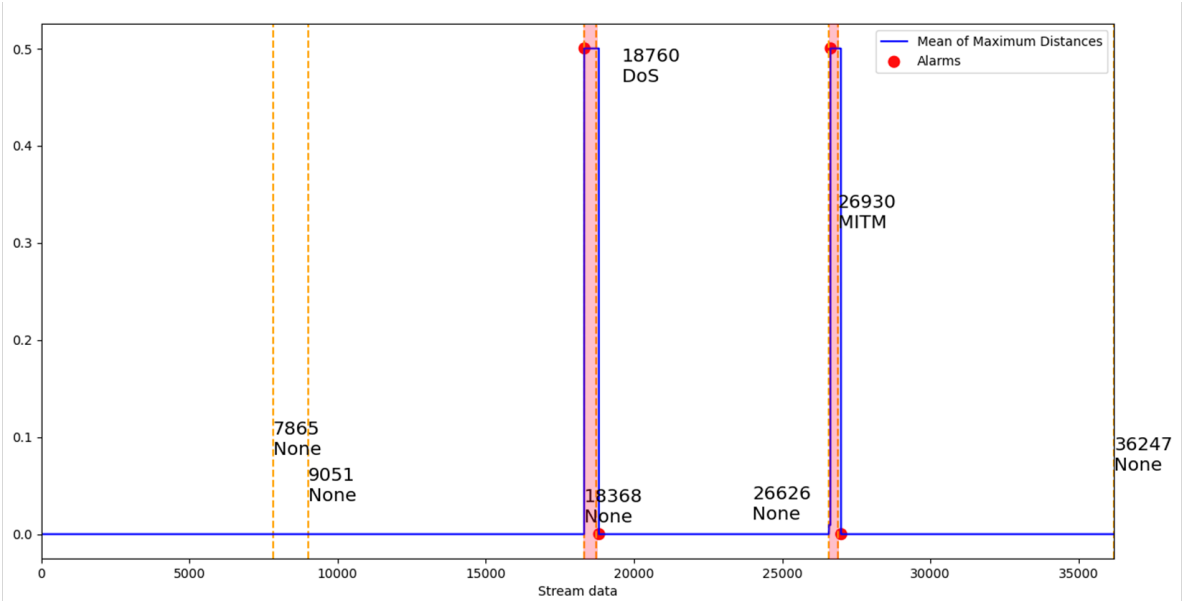

Figure 17 – ICMP stream in Hive Hub: mean of maximum centroid distances and generated alerts.

maximum distances is stable. When the DoS attack starts, there is a very high peak in the mean of maximum distances, signaling that either a new micro-cluster was added far from the other ones or an existing one was updated and moved due to the new behavior brought by the attack. As time passes, the past micro-clusters regarding normal traffic become less relevant and are erased by CluStream. When the DoS attack stops (packet #18760), there is a sudden behavior change again, which reflects in the mean of maximum distances. Later in this scenario (packet #26627), a MITM attack starts, showing again that the proposed approach could detect it quickly, generating alerts.

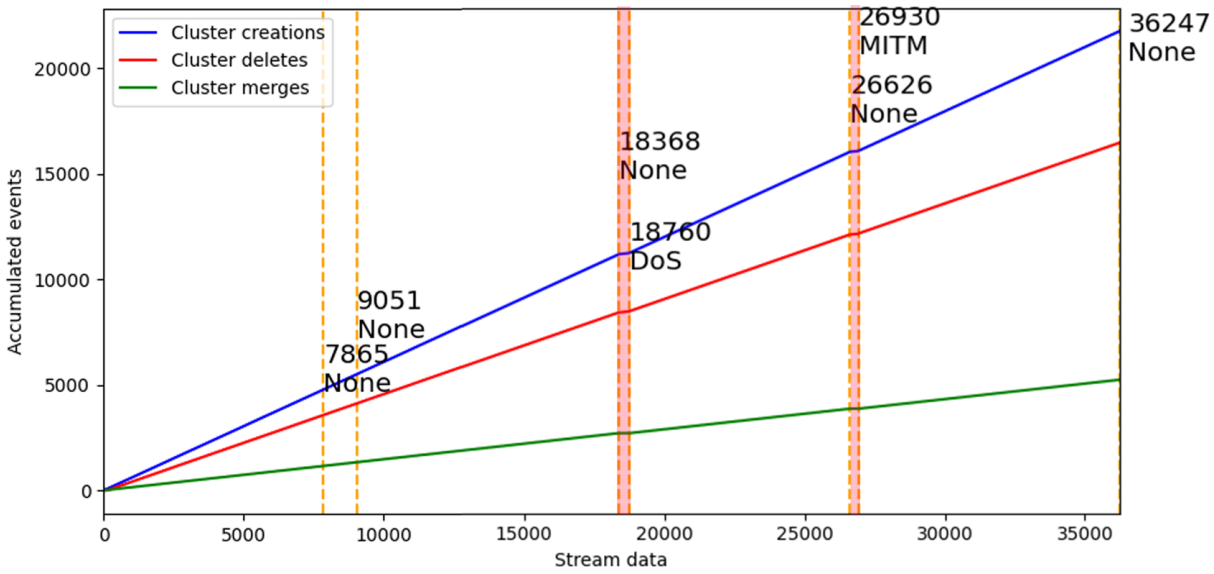

Figure 18 – ICMP stream in Hive Hub: cluster creations, deletes and merges.

Creation, deletion, and merge events for the Hive Hub ICMP scenario are represented in Figure 18. In opposition to the previous scenario, the number of creations, deletions and merges decreases in both attacks. This behavior demonstrates that the ICMP packets arriving at the time of the attack are very similar, thus are being absorbed by the micro-clusters, decreasing the number of events.

Finally, we present the Samsung Smart Things UDP scenario, which, unlike the previous ones, showed low detection rate and precision. Figure 19 shows that the mean of maximum centroid distances does not show any association with the occurrence of attacks. The normal behavior of the mean of maximum centroid distances in this scenario is irregular and presents many peaks. Meanwhile, during attacks, the distances present smaller irregularities. Therefore, the proposed method cannot distinguish between benign and malicious behavior. As a result, alerts are generated across the entire scenario, without good accuracy.

Figure 20 shows Samsung SmartThings UDP scenario containing its cumulative

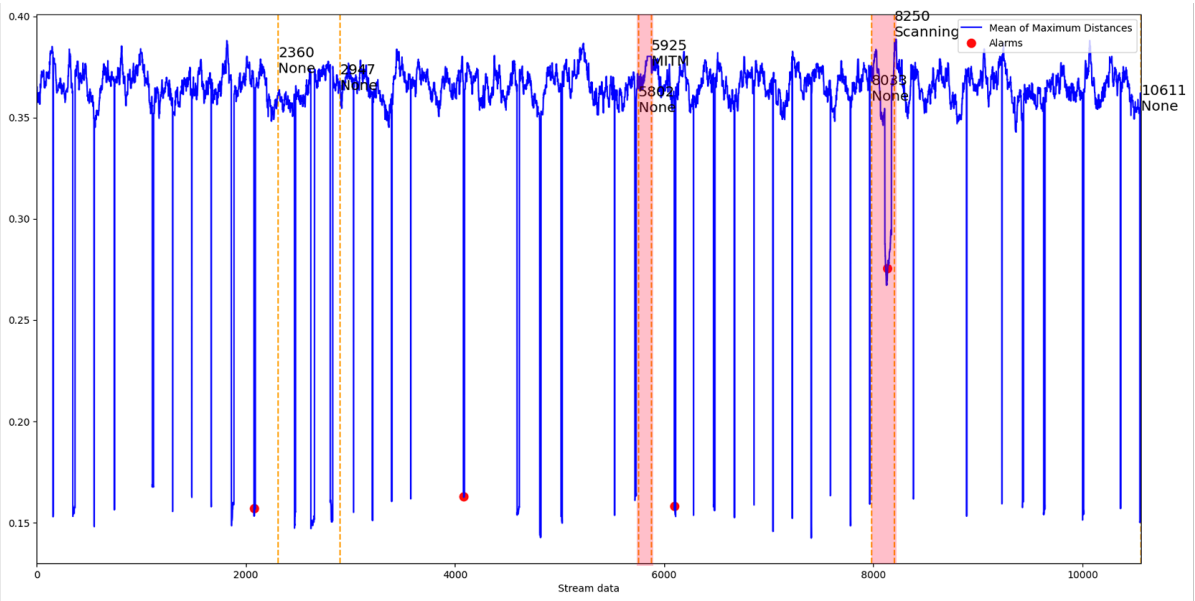

Figure 19 – UDP stream in Samsung Smart Things: mean of maximum centroid distances and generated alerts.

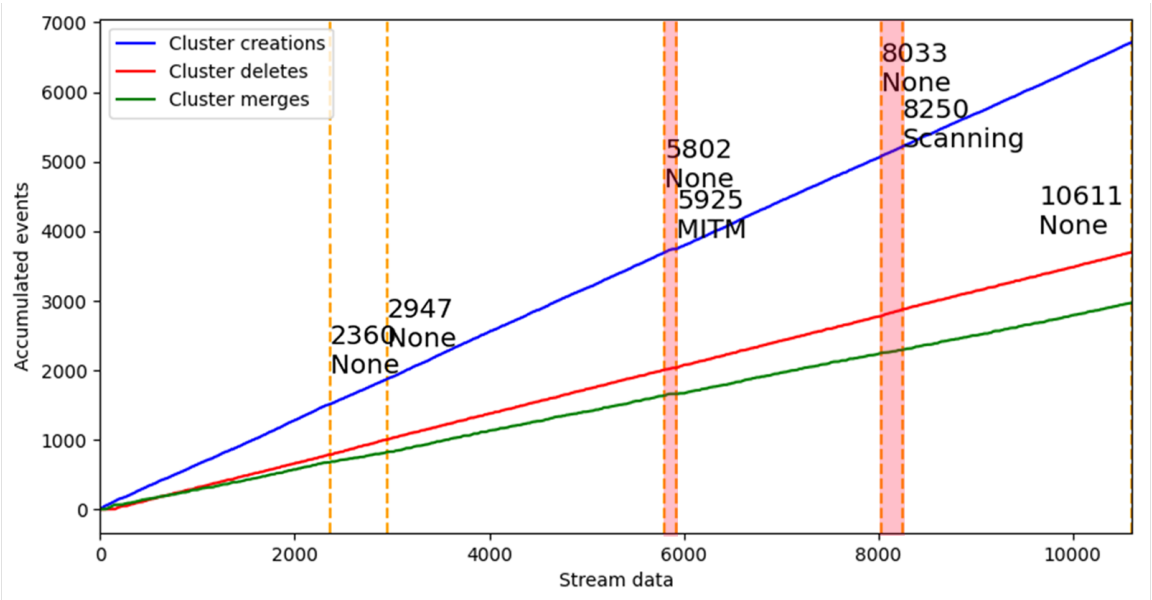

Figure 20 – UDP stream in Samsung Smart Things: cluster creations, deletes and merges.

cluster creations, deletions and merges. There is a slight decrease in merges after packet #2360 and a modest reduction in creation and merge rates during the MITM attack. This change during the attack reflects in the smaller irregularities in the mean of maximum distances, indicating that micro-clusters were absorbing the malicious packets. Also, in contrast with the previous two scenarios, the rate of events almost do not change during the entire scenario.

4.4 Discussion

Overall, the proposed approach achieved high detection rate and precision for about two-thirds of the scenarios. These results suggest some findings, particularly when the main features of our proposal are considered. Firstly, the results confirmed that it was possible to distinguish malicious and benign traffic by analyzing it packet by packet with a combination of stream clustering and change detection algorithms. Anthi et al. [9] had already reached good results by inspecting packet headers in IoT traffic, but they made use of supervised classifiers. Other studies like [17], which employed one-class classification, analyzed the network traffic by grouping packets and extracting their statistical features. In [91] it was proposed an IP flow-based IDS, which aggregates packets into flows and send them for storage and analysis.

The CluStream behavior was another highlight. The way the algorithm managed the micro-clusters' dynamics as new data points arrived clearly reflected the packets' behavior. Minor (and possibly natural) changes in traffic behavior were usually represented by new clusters close to the previous ones. Conversely, attacks drove the algorithm to create new clusters far from the previous ones, which could be more easily detected. As a result, the distance between micro-cluster centroids showed to be a good indicator of malicious behavior. The Page-Hinkley Test was a reliable choice to detect changes in the centroid's distances when an attack was occurring. Once the Page-Hinkley Test allows a fine tuning of its hyperparameters, it was possible to adjust its sensitivity for each scenario. Hence, subtle and sudden changes could be properly distinguished, avoiding false positives.

The findings reported above do not apply to all scenarios. In some devices, UDP and ICMP streams showed very irregular behavior, even under normal conditions. For this reason, the distance between centroids did not present distinct movements for malicious and normal traffic in these scenarios. TCP streams followed a more systematic behavior under normal conditions. Consequently, the proposed approach achieved the best possible results for most of the analyzed TCP streams. The only exception was a MITM attack that was not detected in the TCP stream collected from the Hive Hub.

5 CONCLUSION

The increasing adoption of IoT home devices draws attention to their security. Attackers see new opportunities to break into these devices since they are more vulnerable and become more numerous. There are many types of attacks that exploit multiple IoT vulnerabilities that should be addressed. This work proposed a new approach for change detection in the network behavior through an unsupervised stream learning algorithm.

Currently, data stream is an important type of data that is highly present in the use of cybernetic networks. Challenges related to continuous data flows are arising and new solutions using stream learning need to be created. The proposed model uses a packet classification scheme, in combination with stream clustering and change detection algorithms. Respectively, the algorithms used in experiments were the CluStream and the Page-Hinkley Test.

After organizing the packets into three streams, CluStream instances process them, maintaining the packets partitioned in micro-clusters at low memory cost. Then, the Page-Hinkley Test is applied over the mean of maximum distances among micro-cluster centroids to detect changes in the device's incoming traffic. The proposed approach does not require labeled samples to be trained or build a learning model. This is closer to the needs of IoT smart homes.

To create experiments with real data packets, scenarios were made by using data clippings. Every scenario was made with normal-to-normal, normal-to-attack and attack-to-normal behavior transitions. It is expected that no significant changes occurs during normal-to-normal transitions and alerts are raised when there is an attack transition.

Experiments performed with data from five devices in eleven scenarios showed that the best results were mostly found for TCP and ICMP streams. A total of seven out of eleven scenarios reached 100% of detection rate and precision, which is a great result. Also, despite the different changes that distinct attacks caused in the micro-clusters' behavior, the proposed approach could detect all experimented types of attacks.

In future work, we intend to evaluate the proposed approach with different unsupervised and change detection algorithms. We also want to use other datasets, as well as perform experiments deploying the proposed approach into IoT devices.

BIBLIOGRAPHY

[1] TAHSIEN, S. M.; KARIMIPOUR, H.; SPACHOS, P. Machine learning based solutions for security of Internet of Things (IoT): A survey. *Journal of Network and Computer Applications*, v. 161, p. 102630, 2020. ISSN 1084-8045. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1084804520301041>.

[2] BORGIA, E. The Internet of Things vision: Key features, applications and open issues. *Computer Communications*, v. 54, p. 1–31, 2014. ISSN 0140-3664. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0140366414003168>.

[3] CONTI, M.; KALIYAR, P.; LAL, C. Censor: Cloud-enabled secure IoT architecture over SDN paradigm. *Concurrency and Computation: Practice and Experience*, v. 31, n. 8, p. e4978, 2019. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.4978>.

[4] AL-GARADI, M. A. et al. A survey of machine and deep learning methods for Internet of Things (IoT) security. *IEEE Communications Surveys & Tutorials*, v. 22, n. 3, p. 1646–1685, 2020.

[5] MOLINA-CORONADO, B. et al. Survey of network intrusion detection methods from the perspective of the knowledge discovery in databases process. *IEEE Transactions on Network and Service Management*, v. 17, n. 4, p. 2451–2479, 2020.

[6] GAMA, J. *Knowledge discovery from data streams*. [S.l.]: CRC Press, 2010.

[7] CERAVOLO, P. et al. Evaluation goals for online process mining: A concept drift perspective. *IEEE Transactions on Services Computing*, v. 15, n. 4, p. 2473–2489, 2022.

[8] GAMA, J. a. et al. A survey on concept drift adaptation. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 46, n. 4, mar 2014. ISSN 0360-0300. Disponível em: <https://doi-org.ez78.periodicos.capes.gov.br/10.1145/2523813>.

[9] ANTHI, E. et al. A supervised intrusion detection system for smart home IoT devices. *IEEE Internet of Things Journal*, v. 6, p. 9042–9053, 2019.

[10] PISHVA, D. Internet of Things: Security and privacy issues and possible solution. In: *2017 19th International Conference on Advanced Communication Technology (ICACT)*. [S.l.: s.n.], 2017. p. 797–808.

[11] ZHENG, S. et al. User perceptions of smart home IoT privacy. *Proc. ACM Hum.-Comput. Interact.*, Association for Computing Machinery, New York, NY, USA, v. 2, n. CSCW, nov. 2018.

[12] CHOW, R. The last mile for IoT privacy. *IEEE Security & Privacy*, v. 15, n. 6, p. 73–76, 2017.

[13] MOUSTAFA, N.; TURNBULL, B.; CHOO, K.-K. R. An ensemble intrusion detection technique based on proposed statistical flow features for protecting network traffic of Internet of Things. *IEEE Internet of Things Journal*, v. 6, n. 3, p. 4815–4830, 2019.

[14] AGGARWAL, C. C. et al. A framework for clustering evolving data streams. In: ELSEVIER. *Proceedings 2003 VLDB conference*. [S.l.], 2003. p. 81–92.

[15] QADDOURA, R. et al. A multi-stage classification approach for IoT intrusion detection based on clustering with oversampling. *Applied Sciences*, v. 11, n. 7, 2021. ISSN 2076-3417.

[16] VU, L. et al. Deep transfer learning for IoT attack detection. *IEEE Access*, v. 8, p. 107335–107344, 2020.

[17] MEIDAN, Y. et al. N-BaIoT—network-based detection of IoT botnet attacks using deep autoencoders. *IEEE Pervasive Computing*, v. 17, n. 3, p. 12–22, 2018.

[18] BEZERRA, V. H. et al. IoTDS: A one-class classification approach to detect botnets in Internet of Things devices. *Sensors*, v. 19, n. 14, 2019. ISSN 1424-8220.

[19] MELO, P. H. A. D. de; MIANI, R. S.; ROSA, P. F. Familyguard: A security architecture for anomaly detection in home networks. *Sensors*, v. 22, n. 8, 2022. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/22/8/2895>.

[20] BRUN, O. et al. Deep learning with dense random neural networks for detecting attacks against IoT-connected home environments. In: GELENBE, E. et al. (Ed.). *Security in Computer and Information Sciences*. Cham: Springer International Publishing, 2018. p. 79–89. ISBN 978-3-319-95189-8.

[21] ALI, S.; LI, Y. Learning multilevel auto-encoders for DDoS attack detection in smart grid network. *IEEE Access*, v. 7, p. 108647–108659, 2019.

[22] SINHA, S. *State of IoT 2021: Number of connected IoT devices growing 9% to 12.3 billion globally, cellular IoT now surpassing 2 billion*. 2021. Disponível em: <https://iot-analytics.com/number-connected-iot-devices/>.

[23] SEE, A. v. *Number of Internet of Things (IoT) connected devices worldwide from 2019 to 2030*. 2021. Disponível em: <https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/>.

[24] ELAZHARY, H. Internet of Things (IoT), mobile cloud, cloudlet, mobile IoT, IoT cloud, fog, mobile edge, and edge emerging computing paradigms: Disambiguation and research directions. *Journal of Network and Computer Applications*, v. 128, p. 105–140, 2019. ISSN 1084-8045. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1084804518303497>.

[25] IDC. *Worldwide smart home devices market grew 11.7% in 2021 with double-digit growth forecast through 2026, according to IDC*. 2022. Disponível em: <https://www.idc.com/getdoc.jsp?containerId=prUS49051622>.

[26] VACCA, J. R. *Computer and information security handbook*. [S.l.]: Newnes, 2012.

[27] KAUSHIK, A. K.; PILLI, E. S.; JOSHI, R. Network forensic system for port scanning attack. In: *2010 IEEE 2nd International Advance Computing Conference (IACC)*. [S.l.: s.n.], 2010. p. 310–315.

[28] BARNETT, R. J.; IRWIN, B. Towards a taxonomy of network scanning techniques. In: *Proceedings of the 2008 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries: Riding the Wave of Technology*. New York, NY, USA: Association for Computing Machinery, 2008. (SAICSIT '08), p. 1–7. ISBN 9781605582863. Disponível em: <https://doi.org/10.1145/1456659.1456660>.

[29] VYKOPAL, J. A flow-level taxonomy and prevalence of brute force attacks. In: ABRAHAM, A. et al. (Ed.). *Advances in Computing and Communications*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. p. 666–675. ISBN 978-3-642-22714-1.

[30] JO, H.-J.; YOON, J. W. A new countermeasure against brute-force attacks that use high performance computers for big data analysis. *International Journal of Distributed Sensor Networks*, v. 11, n. 6, p. 406915, 2015. Disponível em: <https://doi.org/10.1155/2015/406915>.

[31] RONEN, E. et al. IoT goes nuclear: Creating a zigbee chain reaction. In: *2017 IEEE Symposium on Security and Privacy (SP)*. [S.l.: s.n.], 2017. p. 195–212.

[32] HWANG, H. et al. A study on mitm (man in the middle) vulnerability in wireless network using 802.1x and eap. In: *2008 International Conference on Information Science and Security (ICISS 2008)*. [S.l.: s.n.], 2008. p. 164–170.

[33] EASTTOM, C. *Computer Security Fundamentals, Third Edition*. Pearson Certification, 2016. ISBN 9780134470627. Disponível em: <https://books.google.com.br/books?id=y9XdnQAACAAJ>.

[34] IBM. *Anatomy of an IoT malware attack*. 2019. Disponível em: <https://developer.ibm.com/articles/iot-anatomy-iot-malware-attack/>.

[35] OWASP. *Denial of service*. 2015. Disponível em: <https://wiki.owasp.org/index.php/Denial_of_Service>.

[36] JIA, Y. et al. Flowguard: An intelligent edge defense mechanism against IoT DDoS attacks. *IEEE Internet of Things Journal*, v. 7, n. 10, p. 9552–9562, 2020.

[37] VIGNAU, B. et al. The evolution of IoT malwares, from 2008 to 2019: Survey, taxonomy, process simulator and perspectives. *Journal of Systems Architecture*, Elsevier, v. 116, p. 102143, 2021.

[38] SALIM, M. M.; RATHORE, S.; PARK, J. H. Distributed denial of service attacks and its defenses in iot: a survey. *The Journal of Supercomputing*, Springer, v. 76, n. 7, p. 5320–5363, 2020.

[39] OWASP Internet of Things top 10 2018. 2018. Disponível em: <https://owasp.org/www-pdf-archive/OWASP-IoT-Top-10-2018-final.pdf>.

[40] BUSHWICK, S. *Ultrasonic attack device hacks phones through solid objects*. Scientific American, 2020. Disponível em: <https://www.scientificamerican.com/article/ultrasonic-attack-device-hacks-phones-through-solid-objects/>. Acesso em: 09/06/2022.

[41] PAGANINI, P. *SurfingAttack - hacking phones via ultrasonic waves*. 2020. Disponível em: <https://www.cyberdefensemagazine.com/surfingattack-hacking-phones-via-ultrasonic-waves/>. Acesso em: 09/06/2022.

[42] SUGAWARA, T. et al. *Light Commands: Laser-Based Audio Injection Attacks on Voice-Controllable Systems*. arXiv, 2020. Disponível em: <https://arxiv.org/abs/2006.11946>.

[43] GREENBERG, A. *Hackers can use lasers to 'speak' to your Amazon Echo*. Conde Nast, 2019. Disponível em: <https://www.wired.com/story/lasers-hack-amazon-echo-google-home/>. Acesso em: 09/06/2022.

[44] PERLROTH, N. *With a laser, researchers say they can hack Alexa, google home or Siri*. The New York Times, 2019. Disponível em: <https://www.nytimes.com/2019/11/04/technology/digital-assistant-laser-hack.html>. Acesso em: 09/06/2022.

[45] BASHIR, M. et al. Online privacy and informed consent: The dilemma of information asymmetry. *Proceedings of the Association for Information Science and Technology*, v. 52, n. 1, p. 1–10, 2015. ISSN 23739231.

[46] BUGEJA, J.; JACOBSSON, A.; DAVIDSSON, P. The ethical smart home: Perspectives and guidelines. *IEEE Security & Privacy*, v. 20, n. 1, p. 72–80, 2022.

[47] SCARFONE, K.; MELL, P. Intrusion detection and prevention systems. In: _____. *Handbook of Information and Communication Security*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. p. 177–192. ISBN 978-3-642-04117-4. Disponível em: <https://doi.org/10.1007/978-3-642-04117-4_9>.

[48] BACE, R.; MELL, P. Intrusion detection systems, National Institute of Standards and Technology (NIST). *Technical Report 800-31*, 2001.

[49] ZARPELÃO, B. B. et al. A survey of intrusion detection in Internet of Things. *Journal of Network and Computer Applications*, v. 84, p. 25–37, 2017. ISSN 1084-8045. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1084804517300802>.

[50] LIAO, H.-J. et al. Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, v. 36, n. 1, p. 16–24, 2013. ISSN 1084-8045. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1084804512001944>.

[51] JOSE, S. et al. A survey on anomaly based host intrusion detection system. *Journal of Physics: Conference Series*, IOP Publishing, v. 1000, p. 012049, apr 2018. Disponível em: <https://doi.org/10.1088/1742-6596/1000/1/012049>.

[52] JABEZ, J.; MUTHUKUMAR, B. Intrusion detection system (IDS): Anomaly detection using outlier detection approach. *Procedia Computer Science*, v. 48, p. 338–346, 2015. ISSN 1877-0509. International Conference on Computer,

Communication and Convergence (ICCC 2015). Disponível em: <https://www.sciencedirect.com/science/article/pii/S1877050915007000>.

[53] ZAMAN, S. et al. Security threats and artificial intelligence based countermeasures for Internet of Things networks: A comprehensive survey. *IEEE Access*, v. 9, p. 94668–94690, 2021.

[54] DAS, K. *Protocol Anomaly Detection for Network-based Intrusion Detection*. Sans, 2021. Disponível em: <https://www.sans.org/white-papers/349/>.

[55] AYDIN, M. A.; ZAIM, A. H.; CEYLAN, K. G. A hybrid intrusion detection system design for computer network security. *Computers & Electrical Engineering*, v. 35, n. 3, p. 517–526, 2009. ISSN 0045-7906. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0045790609000020>.

[56] STAVROULAKIS, P.; STAMP, M. *Handbook of information and communication security*. [S.l.]: Springer Science & Business Media, 2010.

[57] BARBARÁ, D. Requirements for clustering data streams. *SIGKDD Explor. Newsl.*, Association for Computing Machinery, New York, NY, USA, v. 3, n. 2, p. 23–27, jan 2002. ISSN 1931-0145. Disponível em: <https://doi.org/10.1145/507515.507519>.

[58] BABCOCK, B. et al. Models and issues in data stream systems. In: *Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. New York, NY, USA: Association for Computing Machinery, 2002. (PODS '02), p. 1–16. ISBN 1581135076. Disponível em: <https://doi.org/10.1145/543613.543615>.

[59] GAMA, J.; RODRIGUES, P. P. *Data Stream Processing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. 25–39 p. ISBN 978-3-540-73679-0.

[60] MUTHUKRISHNAN, S. *Data streams: Algorithms and applications*. [S.l.]: Now Publishers Inc, 2005.

[61] READ, J. et al. Data streams are time series: Challenging assumptions. In: SPRINGER. *Brazilian Conference on Intelligent Systems*. [S.l.], 2020. p. 529–543.

[62] SCHUBERT, E.; GERTZ, M. Numerically stable parallel computation of (co-)variance. In: *Proceedings of the 30th International Conference on Scientific and Statistical Database Management*. New York, NY, USA: Association for Computing Machinery, 2018. (SSDBM '18). ISBN 9781450365055. Disponível em: <https://doi.org/10.1145/3221269.3223036>.

[63] JAIN, R.; CHLAMTAC, I. The p^2 algorithm for dynamic calculation of quantiles and histograms without storing observations. *Commun. ACM*, Association for Computing Machinery, New York, NY, USA, v. 28, n. 10, p. 1076–1085, oct 1985. ISSN 0001-0782. Disponível em: <https://doi.org/10.1145/4372.4378>.

[64] MONTIEL, J. et al. *River: machine learning for streaming data in Python*. 2020.

[65] SOVDAT, B. Updating formulas and algorithms for computing entropy and gini index on time-changing data streams. *CoRR*, abs/1403.6348, 2014. Disponível em: <http://arxiv.org/abs/1403.6348>.

[66] ZUBAROĞLU, A.; ATALAY, V. Data stream clustering: a review. *Artificial Intelligence Review*, v. 54, n. 2, p. 1201–1236, Feb 2021. ISSN 1573-7462. Disponível em: <https://doi.org/10.1007/s10462-020-09874-x>.

[67] SILVA, J. A. et al. Data stream clustering: A survey. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 46, n. 1, jul 2013. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/2522968.2522981>.

[68] CHANDOLA, V.; BANERJEE, A.; KUMAR, V. Anomaly detection: A survey. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 41, n. 3, jul 2009. ISSN 0360-0300. Disponível em: <https://doi-org.ez78.periodicos.capes.gov.br/10.1145/1541880.1541882>.

[69] AGGARWAL, C. C. *An Introduction to Outlier Analysis*. Cham: Springer International Publishing, 2017. 1–34 p. ISBN 978-3-319-47578-3. Disponível em: <https://doi.org/10.1007/978-3-319-47578-3_1>.

[70] BIFET, A.; GAVALDA, R. Adaptive learning from evolving data streams. In: SPRINGER. *International Symposium on Intelligent Data Analysis*. [S.l.], 2009. p. 249–260.

[71] BAENA-GARCIA, M. et al. Early drift detection method. In: *Fourth international workshop on knowledge discovery from data streams*. [S.l.: s.n.], 2006. v. 6, p. 77–86.

[72] PAGE, E. S. Continuous inspection schemes. *Biometrika*, [Oxford University Press, Biometrika Trust], v. 41, n. 1/2, p. 100–115, 1954. ISSN 00063444.

[73] WAN, Y. et al. IoTArgos: A multi-layer security monitoring system for internet-of-things in smart homes. In: *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*. [S.l.: s.n.], 2020. p. 874–883.

[74] HEARTFIELD, R. et al. Self-configurable cyber-physical intrusion detection for smart homes using reinforcement learning. *IEEE Transactions on Information Forensics and Security*, v. 16, p. 1720–1735, 2021.

[75] YANG, K. et al. Active learning for wireless IoT intrusion detection. *IEEE Wireless Communications*, v. 25, n. 6, p. 19–25, 2018.

[76] VACCARI, I. et al. Mqttset, a new dataset for machine learning techniques on mqtt. *Sensors*, v. 20, n. 22, 2020. ISSN 1424-8220.

[77] LI, W. et al. Designing collaborative blockchained signature-based intrusion detection in IoT environments. *Future Generation Computer Systems*, v. 96, p. 481–489, 2019. ISSN 0167-739X.

[78] KHALID, A. et al. Scalable and practical one-pass clustering algorithm for recommender system. *Intelligent Data Analysis*, v. 21, n. 2, p. 279 – 310, 2017. ISSN 1088467X. Disponível em: <https://www.researchgate.net/publication/314297600_Scalable_and_practical_One-Pass_clustering_algorithm_for_recommender_system>.

[79] REYNOLDS, J.; POSTEL, J. *RFC 1700: Assigned numbers*. 1994. Disponível em: <ftp://ftp.internic.net/rfc/rfc1700.txt>.

[80] POSTEL, J. *RFC 791: Internet Protocol*. 1981. Disponível em: <ftp://ftp.internic.net/rfc/rfc791.txt>.

[81] NICHOLS, K. et al. *RFC 2474: Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*. 1998. Disponível em: <ftp://ftp.internic.net/rfc/rfc2474.txt>.

[82] TANENBAUM, A. *Redes de computadores*. [S.l.]: Elsevier, 2003. ISBN 9788535211856.

[83] POSTEL, J. *RFC 792: Internet Control Message Protocol*. 1981. Disponível em: <ftp://ftp.internic.net/rfc/rfc792.txt>.

[84] AGGARWAL, C. C. et al. A framework for projected clustering of high dimensional data streams. In: *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*. [S.l.: s.n.], 2004. p. 852–863.

[85] SANGAM, R. S.; OM, H. Equi-clustream: a framework for clustering time evolving mixed data. *Advances in Data Analysis and Classification*, Springer, v. 12, n. 4, p. 973–995, 2018.

[86] GRUA, E. M. et al. Clustream-gt: Online clustering for personalization in the health domain. In: *IEEE/WIC/ACM International Conference on Web Intelligence*. [S.l.: s.n.], 2019. p. 270–275.

[87] BARDDAL, J. P. et al. A survey on feature drift adaptation: Definition, benchmark, challenges and future directions. *Journal of Systems and Software*, v. 127, p. 278–294, 2017. ISSN 0164-1212. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0164121216301030>.

[88] GAMA, J.; SEBASTIAO, R.; RODRIGUES, P. P. Issues in evaluation of stream learning algorithms. In: *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. [S.l.: s.n.], 2009. p. 329–338.

[89] GAMA, J.; SEBASTIAO, R.; RODRIGUES, P. P. On evaluating stream learning algorithms. *Machine learning*, Springer, v. 90, n. 3, p. 317–346, 2013.

[90] SEBASTIÃO, R.; FERNANDES, J. M. Supporting the page-hinkley test with empirical mode decomposition for change detection. In: SPRINGER. *International Symposium on Methodologies for Intelligent Systems*. [S.l.], 2017. p. 492–498.

[91] SANTOS, L. et al. A flow-based intrusion detection framework for Internet of Things networks. *Cluster Computing*, 2021. ISSN 1573-7543. Disponível em: <https://doi.org/10.1007/s10586-021-03238-y>.

Appendix

APPENDIX A – SCENARIO'S DETAILS

This appendix presents the details of the 11 scenarios used in this work.

Device name	Protocol	Clipping #	Attack Type	Length
Hive Hub	TCP	1	None	41580
		2	None	6625
		3	None	18996
		4	DoS	13521
		5	None	31364
		6	MITM	607
		7	None	41580
		8	Scanning	20650
		9	None	18996
	UDP	1	None	1004
		2	None	50
		3	None	1001
		4	MITM	100
		5	None	1010
		6	Scanning	298
		7	None	1001
	ICMP	1	None	7866
		2	None	1186
		3	None	9317
		4	DoS	392
		5	None	7866
		6	MITM	304
		7	None	9317

Table 8 – Hive Hub scenarios.

Device name	Protocol	Clipping #	Attack Type	Length
Samsung Smart Things	TCP	1	None	15419
		2	None	5783
		3	None	21615
		4	DoS	5952
		5	None	15419
		6	MITM	661
		7	None	21615
		8	Scanning	20950
		9	None	15419
	UDP	1	None	2361
		2	None	587
		3	None	2855
		4	MITM	123
		5	None	2108
		6	Scanning	217
		7	None	2361

Table 9 – Samsung Smart Things scenarios.

Device name	Protocol	Clipping #	Attack Type	Length
Lifx Smart Lamp	TCP	1	None	12819
		2	None	2942
		3	None	12819
		4	DoS	9456
		5	None	12819
		6	MITM	525
		7	None	12819
	ICMP	1	None	24420
		2	None	6499
		3	None	33194
		4	MITM	540
		5	None	24420

Table 10 – Lifx scenarios.

Device name	Protocol	Clipping #	Attack Type	Length
TP-Link NC200 Camera	TCP	1	None	4166
		2	None	3222
		3	None	5600
		4	DoS	4181
		5	None	4166
		6	MITM	216
		7	None	5600
		8	Scanning	8531
		9	None	4166
	UDP	1	None	1092
		2	None	648
		3	None	1388
		4	DoS	1082
		5	None	1020
		6	MITM	219
		7	None	1092
		8	Scanning	620
		9	None	1388

Table 11 – TP-Link NC200 Camera scenarios.

Device name	Protocol	Clipping #	Attack Type	Length
TP-Link SmartPlug	TCP	1	None	1042
		2	None	4700
		3	None	1410
		4	DoS	702
		5	None	1042
	UDP	1	None	486
		2	None	894
		3	None	486
		4	DoS	1206
		5	None	486
		6	IoT-Toolkit	8433
		7	None	486

Table 12 – TP-Link SmartPlug scenarios.

PUBLISHED WORKS

Trabalhos publicados pelo autor durante o programa.

1. Fernando Henrique Yoshiaki Nakagawa, Sylvio Barbon Jr., Bruno Bogaz Zarpelão **Attack Detection in Smart Home IoT Networks using CluStream and Page-Hinkley Test**, Latincom (IEEE Latin-American Conference on Communications), 2021, IEEE, (Qualis CC 2021, B1).