FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Distributed Dendritic Cell Algorithm architecture: A Blockchain perspective

Allan Borges de Sousa

Mestrado em Engenharia Informática e Computação

Supervisor: Rui Pinto

Second Supervisor: Gil Gonçalves

August 7, 2023

© Allan Borges de Sousa, 2023

Distributed Dendritic Cell Algorithm architecture: A Blockchain perspective

Allan Borges de Sousa

Mestrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

President: José Manuel de Magalhães Cruz
Referee: Rui Pedro Ferreira Pinto
Referee: João Paulo da Conceição Soares

August 7, 2023

Resumo

O campo de Sistemas Imunológicos Artificiais (AIS) não é novo. Ao longo dos anos, vimos estudos em vários domínios tentando aproveitar essas tecnologias para melhorar seus respectivos estados da arte. Uma das aplicações populares para tais sistemas é em Sistemas de Detecção de Intrusão (IDS) e detecção de anomalias.

Nos últimos anos, um algoritmo específico tem ganhado aderência entre os IDS baseados em AIS, o Algoritmo da Célula Dendrítica (DCA). O DCA é baseado na Teoria do Perigo (DT) e usa o conceito de sinais de perigo e sinais seguros para decidir se um tipo de antígeno representa uma ameaça ou não. O DCA funciona primeiro amostrando antígenos do ambiente, representando os dados a serem classificados; depois de amostrar os antígenos, extrai sinais e calcula a saída. Dependendo da exposição a sinais específicos e antígenos, a Célula Dendrítica (DC) é classificada como madura ou semi-maduras; as DCs maduras estimulam uma resposta imune devido à probabilidade de antígenos anômalos, enquanto as DCs semi-maduras fornecem tolerância devido à probabilidade de antígenos normais.

A 4ª Revolução Industrial usa e depende de diversas novas tecnologias, como Inteligência Artificial (IA), Internet das Coisas (IoT), Sistemas Ciber-Físicos (CPS) e sistemas Blockchain. Isso, juntamente com os novos paradigmas altamente distribuídos dessas tecnologias, traz à tona o fato de que sistemas IDS tradicionais não foram pensados com foco em sistemas distribuídos.

O DCA pode ser um candidato em potencial para a integração de IDS com sistemas blockchain. Existe um mapeamento natural de um sistema que protege o corpo (sistema) por completo, em vez de proteger partes individuais. No entanto, versões tradicionais do DCA foram desenvolvidas em contextos centralizados.

Além disso, uma arquitetura distribuída para um IDS baseado em DCA foi desenvolvida. A arquitetura funciona comunicando os dados e classificações locais de cada nó para os outros nós da rede blockchain. Essa arquitetura remove a necessidade de uma entidade central para a classificação e se demonstra um primeiro passo na integração de DCA com ambientes blockchain.

Abstract

The field of Artificial Immune Systems (AIS) is not a new one. Over the years, we have seen studies across multiple domains trying to leverage these technologies to improve their respective states of the art. One of the popular applications for such systems is in Intrusion Detection Systems (IDS) and anomaly detection.

In recent years, one particular algorithm has been gaining traction among AIS-based IDS, the Dendritic Cell Algorithm (DCA). The DCA is based on the Danger Theory (DT) and uses the concept of danger signals and safe signals to decide if an antigen type represents a threat or not. The DCA works by first sampling antigens from the environment, representing data to be classified; after sampling the antigens, it extracts signals and computes the output. Depending on the exposition to specific signals and antigens, the Dendritic Cell (DC) is classified as mature or semi-mature; Mature DCs stimulate an immune response due to the probability of anomalous antigens, while semi-mature DCs provide tolerance due to the probability of normal antigens.

The 4th Industrial Revolution relies on a range of new technologies, such as Artificial Intelligence (AI), Internet of Things (IoT), Cyber-Physical Systems (CPS), and Blockchain systems. This, coupled with the new highly distributed architecture of these technologies, bring to light the fact that traditional IDS systems are not developed with distributed systems in mind.

The DCA could be a potential candidate for integrating IDS with the blockchain. We have a natural mapping of a system that protects the whole body (system) instead of focusing on protecting individual parts of the whole. However, traditional versions of the DCA were developed with a single-machine context in mind and not with a highly distributed context, such as a blockchain system.

Furthermore, a distributed DCA-based IDS architecture was developed. The architecture works by communicating the data points and local classifications of every node to the other nodes in the blockchain network. This architecture removes the reliance on a central entity for the classification process and provides a first step in integrating the DCA with a blockchain environment.

Acknowledgements

I would like to express my gratitude and appreciation to all those who have supported and contributed to the completion of this work. This work would not have been possible without their assistance, guidance, and encouragement. Specifically, I wanted to extend my thanks to the following individuals:

- **Rui Pinto** My supervisor, whose guidance was paramount to the planning and execution of this work in the best possible way.
- **Family**, Who offered me support and encouragement throughout the whole process.
- **Friends and Colleagues** Particularly my friends *Breno Accioly* and *Carolina Rosemback*, whose assistance and support were invaluable.
- **FEUP** I am grateful to FEUP for providing the necessary resources, facilities, and academic environment. The opportunities and resources made available to me by the university have contributed significantly to the successful completion of this dissertation.

"You should be glad that bridge fell down.
I was planning to build thirteen more to that same design"

Isambard Kingdom Brunel

Contents

List of Figures

List of Tables

List of Algorithms

Listings

Abreviaturas e Símbolos

AI	Artificial Intelligence
AIS	Artificial Immune System
AM	Advanced Manufacturing
API	Application Programming Interface
BIS	Biological Immune System
CDCA	Cursory DCA
CPS	Cyber-Physical Systems
DCA	Dendritic Cell Algorithm
DC	Dendritic Cell
DFS	Distributed File System
DDoS	Distributed Denial of Service
DQ	Danger Quotient
DT	Danger Theory
DS	Danger Signal
E2E	End-to-End
FAR	False Alarm Rate
FTP	File Transfer Protocol
FPR	False Positive Rate
IIoT	Industrial Internet of Things
IDS	Intrusion Detection System
IoT	Internet of Things
MCAV	Mature Context Antigen Value
MAR	Missed Alarm Rate
MAS	Multi-Agent Systems
NDS	Neighbourhood-Danger Score
PAMP	Pathogenic Associated Molecular Pattern
pBFT	Practical Byzantine Fault Tolerance
POI	Proof of Immune
POW	Proof of Work
ROC-AUC	Receiver Operator Characteristic - Area Under Curve
SDK	Source Development Kit
SDS	Self-Danger Score
SOM	Self-Organizing Feature Map
SOC	Security Operations Center
SSH	Secure Shell
SS	Safe Signal
TC	T-Cell
TPR	True Positive Rate
UVA	Unmanned Aerial Vehicle
VANET	Vehicular Ad Hoc Network

Chapter 1

Introduction

1.1 Context

In recent years, there have been a number of industrial paradigms fundamentally different from those that appeared before them. Two examples of such industrial paradigms are Advanced Manufacturing (AM) and Industry 4.0 as a whole. For these industrial paradigms, more than automation is needed. There is a need for adaptability, customization, and autonomy. To attend to this need, these industrial paradigms leverage technologies like Artificial Intelligence (AI), Internet of Things (IoT), and Cyber-Physical Systems (CPS) [43].

This brings substantial changes to networking; whereas before, industries were mainly dominated by a few centralized servers, we now have highly distributed services spanning thousands of servers. We also have thousands of devices connected in decentralized networks, and enterprises have solutions that rely on the security of their private networks.

In addition to that, more and more advanced industries utilize blockchain technologies. Blockchain technologies offer end-to-end (E2E) secure protocols, and decentralized transactions, making it a natural fit for CPS and the Industrial Internet of Things (IIoT) [6]. One of the most invaluable advantages is the extinction of single points of failure, which can be achieved with the natural fault tolerance of these networks. Blockchain and CPS are also somewhat of a natural fit since their features are relatively compatible [52]. Consensus mechanisms, along with immutable storage, come as a possible solution to yet another problem: these new industries have trouble with confidentiality, integrity, and availability as the used systems become more distributed and decentralized [6].

These new configurations create new security risks, such as *Shadow IoT*, which is characterized by the introduction of unauthorized devices in IoT networks. These devices can then be used as an attack surface to attack the network or get access to privileged data, thus compromising the security guarantees of otherwise secure networks. Naturally, new methods and techniques have to be developed to cope with the security needs of such systems. A popular group of techniques that have seen rising in popularity is Artificial Immune Systems (AIS), which are systems that are inspired by the biological immune system. In the context of Industry 4.0 and highly connected

devices, AIS comes into play by leveraging the similarities between the Human Immune System (HIS)'s job of detecting unhealthy changes in the environment and the task of detecting anomalous behavior in a network of devices, for example. One such AIS is Dendritic Cell Algorithm (DCA), and Intrusion Detection System (IDS), which have been used in simulated industrial settings with promising results [41]. In the remainder of this Chapter, we are going to describe the motivations that led to the research of a distributed DCA.

1.2 Motivation

The highly-connected nature of these new industries is a double-edged sword: while allowing new possibilities, such as a trusted history of supply chain activities [25], it creates security challenges by increasing the attack surface, both because the number of devices, but also because the number of interfaces between different devices also increase. In contrast with old industrial paradigms, where attack surfaces were limited to a few centralized servers, this new environment also has another adversity: before, fewer servers meant that the security effort could also be focused between a few servers, but now, with thousands of connected devices, the security effort needs to be distributed. All in all, in these dynamic networks, entirely securing a system is impossible. Since offering security, privacy, and reliability is becoming part of the value industries deliver, researching novel ways to secure these systems is necessary [12].

A notable example of these vulnerabilities came to light during the infamous Stuxnet attack, which targeted Iranian nuclear facilities in 2010 [31]. The attack underscored the potential catastrophic consequences of IIoT cyberattacks, disrupting not just digital infrastructure but also physical systems and services. Despite the steps taken to improve security measures, IIoT systems remain susceptible to sophisticated attacks due to their complex, interconnected nature and the rapid pace of technological advancement [51].

Blockchain systems are also vulnerable to Sybil attacks, but that is one type of attack. Blockchain systems are also susceptible to phishing and routing attacks, for example [26]. Also, applications running on top of blockchains also get targeted. This, coupled with the increase in the usage of blockchain in advanced manufacturing industries, more research on security systems tailored for a blockchain environment needs to be conducted. In that regard, AIS-IDS might be a group of technologies to increase the security of these systems.

With AIS-IDS in mind, more research on distributed variations of the DCA needs to be conducted. Current iterations do not use a distributed architecture in order to monitor the entirety of the network. Rather, the DCA is usually utilized locally, with only the local view of the machine taken into consideration.

1.3 Problem specification

As pointed out in the previous sections and as it will be demonstrated with the state-of-the-art analysis, the fact that current systems could be benefited from a distributed IDS to better cope with

modern network scale and requirements, coupled with the current gaps in research, particularly investigating the potential of the DCA in a blockchain environment, we propose contributing to this area by developing an architecture for the DCA to be used as an IDS in a blockchain. Particularly, the architecture would be able to analyze data in a distributed manner while communicating with other nodes to achieve a blockchain-wide IDS. The architecture should ideally not be limited to a single DCA implementation: since there are a number of DCA implementations, each with its own benefits and limitations, binding the architecture to a single version could limit its overall potential. With this property, we also ensure that a version of the DCA more suited to a particular use case and data type scenario could be used, for example.

1.4 Objectives

The goals for this work are: to review the background concepts around our defined problem (AIS, IDS, DCA ...) and compile a literature review regarding distributed IDS, focusing on distributed AIS-based IDS. Following that, this work aims to propose a new DCA-based IDS: an IDS that is particularly suited for distributed environments, using blockchain principles (e.g., consensus algorithms) in order to achieve a distributed IDS. The algorithm should be able to receive signals from multiple sources/nodes in the network and detect intrusions while leveraging the "global" view of the network.

1.4.1 Research Questions

With this work, we aim to answer the following question:

- **How does the proposed solution perform when compared to other DCA-based IDS and a local DCA?**
- **Does the voting schema utilized in the implementation of the proposed architecture impact the performance when compared to the lowest and highest performing individual nodes?**
- **Can the DCA be used in the proposed architecture as an IDS to a blockchain environment? What are the benefits and limitations?**

1.4.2 Solution

To answer those questions, we propose a distributed architecture that uses the DCA for classification in a blockchain network. Ideally, this architecture should be able to be used as an IDS in a blockchain without locking the blockchain to depend on a central entity for the IDS while maintaining the characteristics of the classic DCA algorithm. Some desired characteristics would be:

- **Distributed** The nodes should coordinate with one another to classify threats without relying on reaching the final classifications and processing data on a single machine. This is an obligatory requirement of the architecture.
- **Minimum redundancy** Ideally, data points should not be transmitted to the whole network to avoid excessive copies of each data point across all the nodes in the network. This is an optional requirement of the architecture, and its feasibility and adoption are going to be discussed later in the development chapter.

1.5 Methodology

The methodology of this work will be as follows:

- **Define concrete use case** Firstly, a concrete use case will be defined. This use case will be used as the substrate for the DCA architecture. The use case should define technologies and architectures regarding the concrete blockchain to be used, along with what defines a transaction in the selected blockchain. The inputs to the DCA should also be defined (e.g., network packets, transaction data, etc.). Once the use case is defined, the architecture and subsequent work will be tuned toward this use case.
- **Design the architecture** With a defined use case, the next step will be to start the design of the architecture. This design should be thoroughly explained and defined. Ideally, the architecture should provide the characteristics stated in section 1.4.2.
- **Implementation of the architecture** Once the design of the architecture is finished, the implementation will begin. The implementation will be done in *NodeJS*. A blockchain framework, such as *Hyperledger Fabric*, will be used for the blockchain since our focus is on the DCA architecture.
- **Tests and Conclusion** After the architecture review, we will proceed to test the architecture by comparing it to local versions of the DCA running on each node and with related works if mapping the problem to its architectures are possible. For the comparison, relevant metrics to IDS, such as F1-score, accuracy, and False Alarm Rate (FAR), will be used to verify the suitability of the proposed solution. After the test, the result will be analyzed to determine if the research questions were answered. Finally, a conclusion of the work will be done, presenting the insights of the work, limitations, and ideas for future work.

1.6 Structure of the Thesis

This document is structured as follows:

In this Chapter, the problem, along with the research questions, devised methodology, and the overview of the proposed solution, was presented.

Next, Chapter 2 and Chapter 3, will comprise the state-of-the-art, covering the necessary background review and related research, respectively.

In Chapter 4, the proposed solution will be explained in detail, along with implementation details that might be considered relevant.

In Chapter 5, we are going to test and validate the proposed solution, performing an analysis of the obtained results in order to answer our research questions.

Finally, in Chapter 6, we are going to present the conclusions of the work completed so far, along with insights and limitations, and approaches for future work.

Figure 1.1 visually represents the structure of this document.

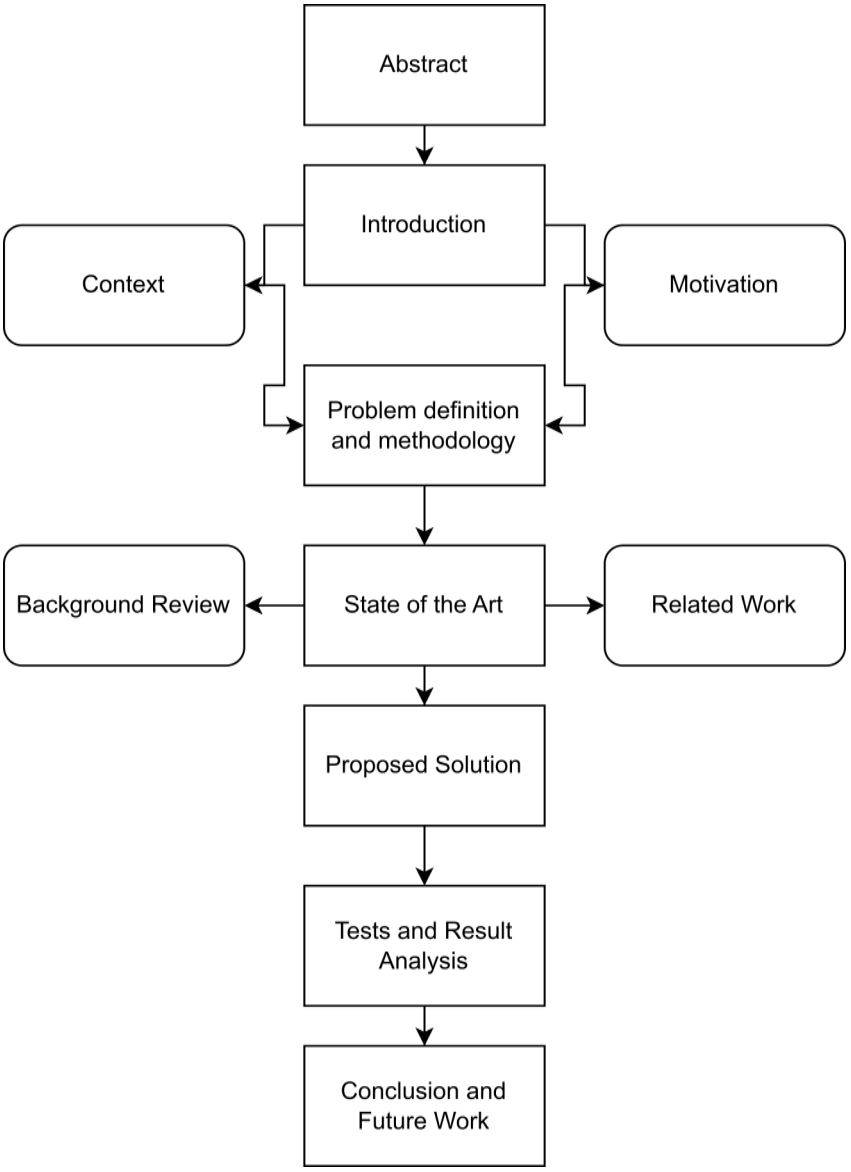

Figure 1.1: Document structure.

Chapter 2

Background

2.1 Blockchain

The term "Blockchain" originally refers to a system that uses a range of techniques to timestamp digital documents, while guaranteeing the integrity of the timestamps [20]. It generally refers to a paradigm used for maintaining information in distributed systems that have some common properties. A definition that could partially capture this generalization is Distributed Ledger Technologies (DLT), a system that records changes to the system state in a distributed ledger. The usual property of these ledgers is high tamper resistance due to the need for consensus between nodes for changes in the state. Figure 2.1 presents the architecture of a typical DLT.

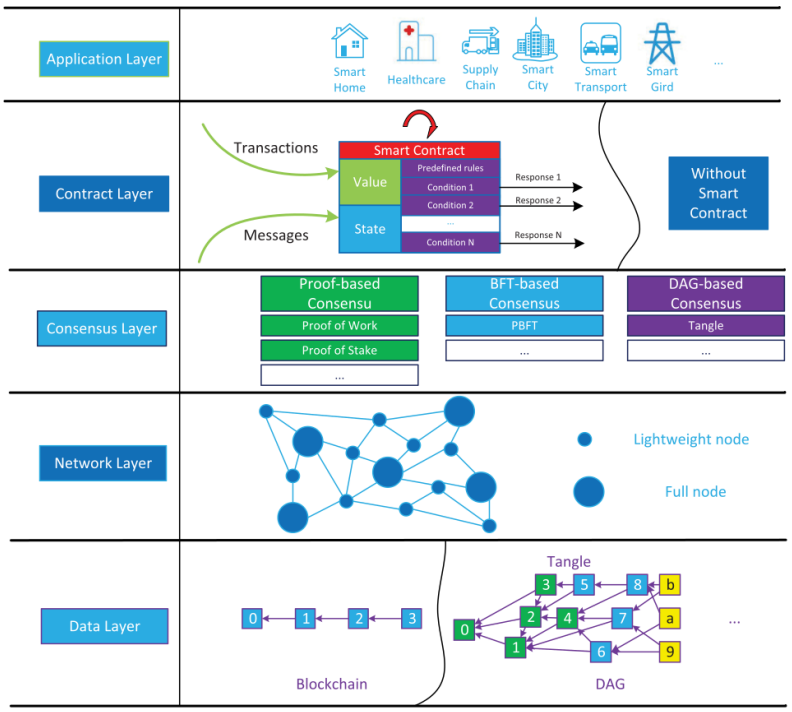

Figure 2.1: Hierarchical architecture of a typical DLT [54].

There is the Data Layer, which is responsible for the storage of the DLT. It can be based on a diversity of models, like block-chaining, or appending nodes to an acyclic graph. The next layer in the architecture is the Network Layer, which governs how nodes will connect to each other, the Consensus Layer implements the protocol responsible for maintaining the integrity of the DLT by validating transactions in a decentralized manner. After that, there are the Contract Layer specifying the business logic of the network and the Application Layer, representing the high-level use case of the DLT.

One of the most popular applications of blockchain technologies was described in [38] (Bitcoin). The white paper presents a crypto coin-based decentralized payment system that does not depend on a centralized trusted entity. It is also an example of what is called a "public blockchain." Public blockchains are fully decentralized and have no access restrictions. On the other hand, "permissioned blockchains" have access restrictions and previously agreed upon parties, heavily reducing the number of nodes but still maintaining the decentralized structure of the network. There are other key differences as well: permissioned blockchains offer more privacy since not every transaction is visible to all nodes and more efficiency since there is a smaller amount of nodes. On the other hand, public blockchains offer more security due to the higher number of nodes and more transparency.[33]

To demonstrate how a blockchain generally works, we are going to use the Bitcoin example. The network is comprised of chained blocks, in which each block is linked to the previous one by having a reference to its hash value. The first block of the network is usually called the *Genesis* block. In Bitcoin's specific implementation, it utilizes the *Merkle tree* data structure. In this representation, the transactions are represented as leaves, while the parent of a specific leaf is its hash. 2.2 visually demonstrate how the *Merkle tree* represents the network state.

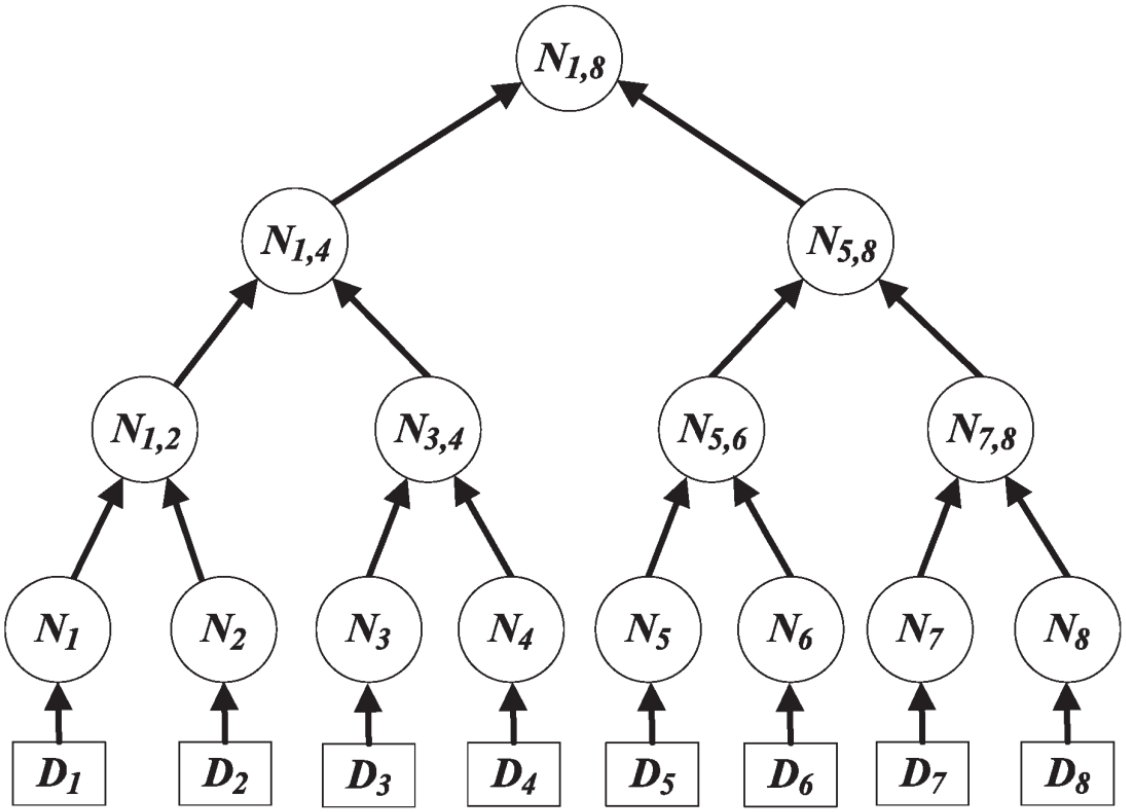

Figure 2.2: Diagram of a Merkle Tree, taken from [32].

In the above diagram, the structure of the Merkle Tree is presented: the leaves (data blocks) have their respective hashes as parents, and the parents of two adjacent nodes have their combined hash (hash of the concatenation of the two hashes) as parents. From this diagram, it is easy to see that changing a data block would eventually change the *Top Hash* (root) as well, providing the ability to guarantee that the ledger hasn't been mutated in an unexpected way.

When modifying the network state (executing a transaction), all the nodes in the network must agree on which node is going to add the transaction to the network. The process in which the nodes end up agreeing on a specific node is called consensus. In the blockchain space, there is a myriad of consensus algorithms, each one of those with its advantages and disadvantages. In the case of Bitcoin, the consensus algorithm used is called Proof of Work (POW). POW is a group of techniques that provide a mechanism for an entity to prove that it has spent a required amount of computational effort [28]. In Bitcoin's case, for example, the computation required amounts to discovering a sequence of bytes that, when hashed, return a digest with a predetermined amount of consecutive 0s at the beginning of the digest.

However, the proof of work is only one consensus protocol. The challenge involved requires significant computational power, POWs also increase latency and energy consumption. These factors make it a poor choice for IoT and IIoT environments. For such scenarios, other consensus protocols may be used, such as the Raft protocol, a leader election-based protocol used by the IBM Blockchain Platform, for example.

Another change in the blockchain world that facilitated its adoption by IIoT environments is the introduction of *smart contracts*. Smart contracts are programs that run on the blockchain. It encapsulates the required business logic and allows blockchains to be deployed in various environments, including IIoT. The transaction and state modifications executed by smart contracts are trusted by the network due to their immutability and because transactions generated by smart contracts are validated by consensus protocols as well. [47]

Another property of such systems is *decentralization*. However, even the level of decentralization can vastly vary between "blockchain systems" [46]. The consensus protocol can also impact the scalability of the network. Overall, there is always a trade-off between scalability, security, and decentralization in blockchain networks: a harder, challenge-based POW might increase the security of the network at the expense of scalability, while permissioned blockchains might be able to scale better and retain the security aspects, but impacting the level of decentralization of the system.

Blockchain technologies have seen usage in industrial configurations, such as usage in supply chain systems in the IIoT [48]. These systems leverage blockchain properties such as tamper-proof history and self-management properties to mitigate classical security problems, such as reliance on centralized cloud infrastructures and single points of failure. In the context of AIS applied to blockchain networks, current research is limited and leaves room for further exploration. One of the reasons could be that traditional AIS systems need some sort of central authority or data processing component, and that would break the decentralized nature of the blockchain.

2.1.1 Artificial Immune Systems

AIS is a popular branch of AI. It takes heavy inspiration from the Biological Immune System (BIS). The BIS is a remarkable system that is capable of adaptation and precise discrimination between *self/non-self*. Even though the notion of *self vs. non-self* is challenged by the Danger Theory (DT), Artificial Immune Systems gained traction as a successful branch of AI [43].

These characteristics make AIS find its use in problems where the task at hand involves discerning between normal and abnormal behavior in a dynamic environment (new emerging threats, for example) as opposed to finding implicit patterns in data, which is usually the task in AI systems.

Traditionally, there are four main strategies used within AIS: *negative selection*, *clonal selection*, *immune networks*, and *danger theory*. Negative selection is based on exposing randomly-created detectors to self/normal/positive antigens, and detectors that match those are eliminated, leaving mature detectors that are going to keep monitoring the environment and alert a human to verify if the matched antigen is indeed a threat [3]. This approach has its limitations. As shown by [30], this approach is severely affected by scaling problems when dealing with real-world network data, particularly because of the time it takes to generate a sufficient number of detectors. Clonal selection, on the other hand, is an approach that relies on the cloning of members of a population of detectors, which have a high affinity to specific input patterns (antigens) [53]. It possesses some

characteristics that are not present in Genetic Algorithms, such as creating a memory-cell population that is capable of recognizing similar patterns in an efficient manner. Immune networks are based on the principles of clonal selection and affinity between antigens and antibodies in the human immune system. It works by generating a population of clones. Clones with a higher affinity to the problem are selected and improved upon, while clones with low affinity are discarded. These processes allow INs to be able to solve non-linear problems such as classification, optimization, and anomaly detection. [13]

The last strategy, and the one that is more relevant to this work, is the danger theory. Traditionally, AIS systems use the idea of *self* / *non-self* to discriminate between harmless and harmful entities. This is an idea that came from classical immunology [3]. [1] reviews this system and points out some problems with this classification. The main idea of Danger Theory (DT) is that rather than discriminating between *self*/*non-self* [4], the immune system relies on danger signals (these can come from "self" as much as from "foreign entities"). The idea of discrimination remains, but how it is achieved is different: In the biological context, the immune system would use surrounding signals (cell death, for example) to match potential threats and react. The difficulty of selecting what is normal(self/safe) and threat(non-self/danger) still exists, but with DT, the signal does not depend on an arbitrary representation of *non-self* [3].

2.1.1.1 Danger Theory: Biological perspective

Biologically, danger signals are generated by necrosis and apoptosis. Necrosis is unregulated cell death that follows cellular stress and generates inflammation through debris. Apoptosis is a regulated cell death with very defined triggers. Even though both processes end with cell death, the path that the immune systems follow in both of these processes is different. The immune system then uses the signals generated from the cell deaths (input) to trigger a pro-inflammatory response (immune response/output) or not. [3]

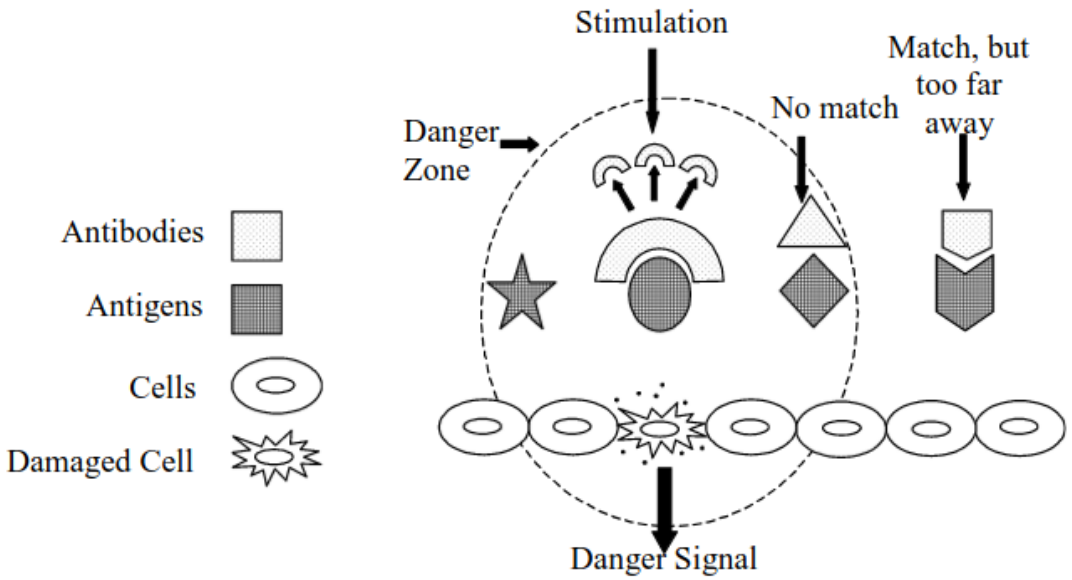

Figure 2.3: Danger Theory Model [4].

2.1.1.2 Danger Theory: Application in AIS

For mapping ideas from the Danger Theory in a biological context to an AIS context, firstly, the signals must be decided on. This is a very specific task that depends heavily on the problem, and it is a task that must be thought through. As an example, for applications in data mining, there is no inherent "danger" meaning to danger signals, as shown in [4]. Besides the challenge of mapping the signals and danger contexts of the problem at hand to fit the DT model, there is also the need to map the system itself as well. What is going to be the antigen-presenting cells? Do we need an abstraction for T-cells? One popular answer to these questions is the Dendritic Cell Algorithm (DCA), which is particularly popular for Intrusion Detection Systems.

The DT is by no means the academic consensus regarding how the immune system works, but its interpretation of the BIS has some proven characteristics useful for AIS ([21], and [9]).

AIS-based approaches have seen some success in recent research, applied in fields such as networking and aviation [43].

2.2 Intrusion Detection Systems

IDS are one of a myriad of techniques to protect computer systems. One of its primary goals is to detect misuse and abuse, both from internal and external entities [3]. [14] proposes a taxonomy for intrusion detection systems. Figure 2.4 presents his characteristics for defining IDS.

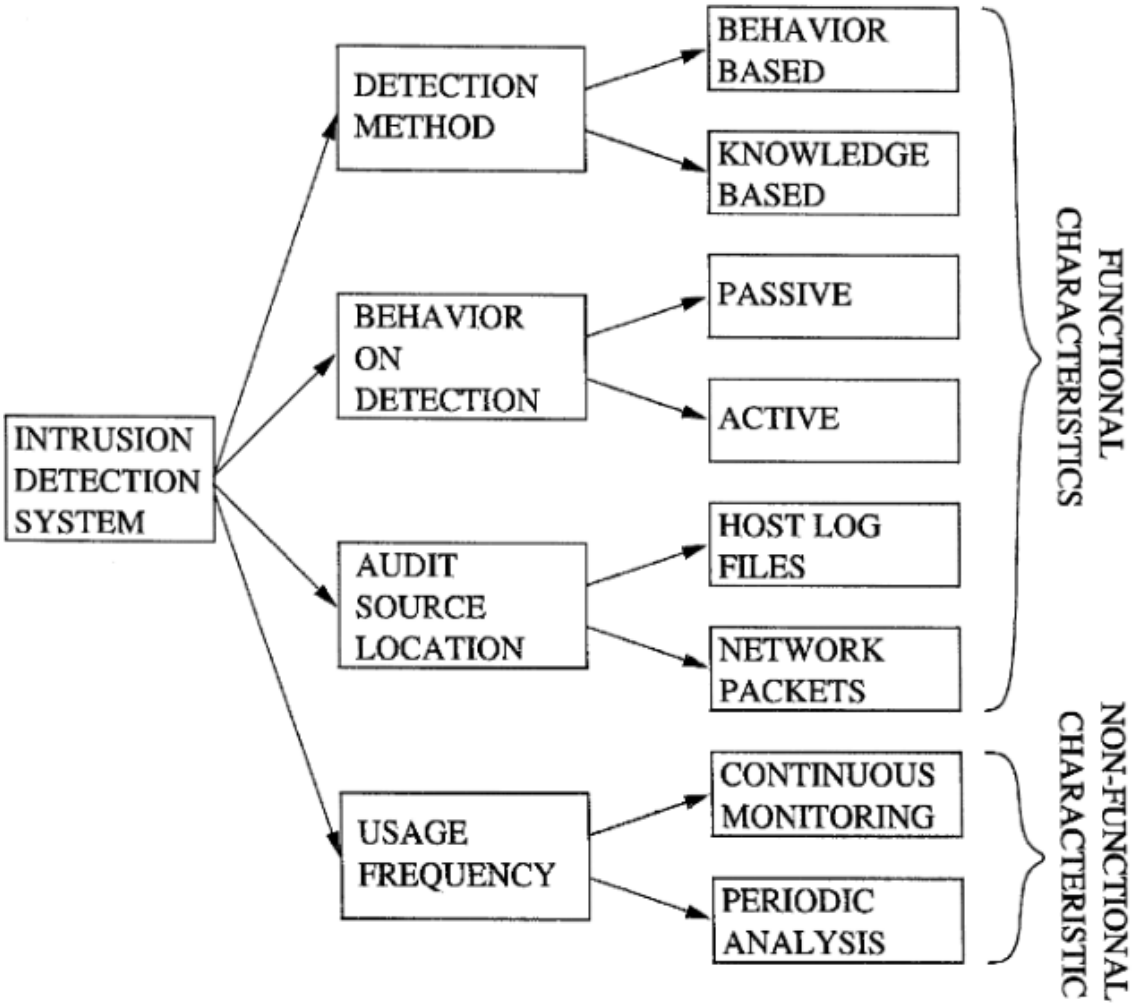

Figure 2.4: Taxonomy of IDS [14].

The *usage frequency* refers to the "periodicity" of the IDS. There are systems that run in real-time, while there are others that need to run periodically due to the requirement of having an existing "database" to be scanned and classified. The *audit source location* refers to the source of information to be analyzed (this will depend on a number of factors besides the actual IDS, such as the environment it is running on and the particular use case). *behavior on detection* characterizes the type of action that is taken once an intrusion is detected. It is self-explanatory: any sort of action besides alarming is characteristic of an active IDS. Finally, we have the *detection method*, which is divided into *knowledge-based* and *behavior-based*. The former relies on signatures of known attacks, while the latter relies on a model of normal behavior, and behavior that falls out of the model is deemed malicious.

A signature-based IDS utilizes signatures of known attacks and threats, for example, the memory trace or network packet pattern of a known virus, to detect malicious behavior. The main disadvantage of this approach is its lack of robustness: it cannot detect previously unknown threats,

so it is not appropriate for a scenario in which new threats are deployed at a high frequency [36].

On the other hand, an anomaly-based IDS relies on a known pattern of normal/healthy behavior, usually called a *baseline*. Having a baseline profile, the IDS can differentiate anomalous behavior from the normal baseline profile. The main advantage of this approach is that it can detect known threats and previously unknown threats since the system relies on a normal baseline profile. On the other hand, the main disadvantage is that normal behavior is classified as anomalous [36]. There are also hybrid systems that try to leverage both manners of detection ([3], [14], [36]).

In the context of these IDS categories, signature-based IDS might not be the most appropriate type for our context due to its lack of robustness and adaptability. On the other hand, anomaly-based IDS seems more suited to our context since its capable of detecting the traces of previously unseen forms of attack. There are also a category of AIS-based IDS. These IDS rely on bio-inspired computation, more specifically artificial immune techniques, to build a more robust and adaptable IDS.

AIS-based IDS uses a number of different techniques, including artificial antibodies, artificial antigens, and artificial lymphocytes, to detect and respond to potential threats. These systems have several advantages over traditional IDS, including the ability to adapt and learn from past experiences and the ability to detect previously unknown threats. It also offers a way of thinking about these systems that do not rely entirely on the distinction between *self/non-self*, which has its fair share of scope problems ([3], [4]).

There have been numerous studies on AIS-based IDS in recent years, and the results have been generally positive ([41], [44]). Generally, common metrics to verify the performance of such systems are detection rate, false positive rate, precision, and recall.

Overall, AIS-based IDS shows great promise as a method for detecting and defending against cyber threats. However, further research is needed to fully understand their capabilities and limitations and to determine how they can be effectively integrated into distributed environments, such as blockchain systems.

2.2.1 Dendritic Cell Algorithm

The DCA is the focus of this work. The DCA is based on the DT and the behavior of Dendritic Cells (DC). According to [19], "DCs have the power to suppress or activate the immune system through the correlation of signals from an environment, combined with location markers in the form of antigen." Firstly, it is important to note that the DCA does not attempt dynamic learning and, as it is going to be explained, is different from traditional negative selection.

To briefly present the DCA *in vivo*, basic DC biology and terminology will be presented. For a more detailed explanation and overview of the DC from a biological perspective, refer to [37].

The DC's job in the innate immune system is to detect threats and determine the T-cell's response to antigens. This response could either be a tolerable or an immune response. The response to be generated by the DC will depend on the concentration of a handful of signals: Pathogenic

associated molecular patterns (PAMPs), apoptotic signals, and inflammatory cytokines. The concentration of these signals will determine the state of the DC: immature, semi-mature, or mature. [19]

- **PAMPs** — Known threat-signatures.
- **Danger signals** — Released by damage to cells.
- **Safe signals** — Released by expected cell death.
- **Inflammatory cytokines** — Released by general cell distress.

The DCA was first introduced in the *Danger Project* [3]. Even though its focus was on IDS (which is one of the focuses of this work), the DCA has been shown to work successfully in other scenarios, such as machine learning [18]. As demonstrated in [18], the DCA is sensible for changes in context, one of the characteristics that might make it suitable for IDS problems.

PAMPs have a very important role in the classification process of the DCA. This means that PAMPs are very problem-specific. In other words, the DCA heavily benefits from expert knowledge. It is a double-edged sword: While it can benefit greatly from expert knowledge, this means that problem modeling can be challenging, and getting at least an adequate representation of what should the danger signals be is paramount for the success of the algorithm.

Besides the problem modeling, its first implementation had a couple of problems. For instance, it relied on randomly generated numbers, and as a consequence of that, the deterministic DCA (dDCA) was introduced in [17]. Over the years, other adaptations of the algorithm, like the cursory DCA [41] that focuses on modularizing the DCA and reducing the reliance on entire dataset pre-processing, were introduced in the AIS literature as well, with the objective of solving its limitations.

In the following Algorithm (1), we present a basic pseudo-code example of how the DCA works at a high level, extracted from [18].

Algorithm 1 High-level DCA.

```
1: Create DC pool of N cells
2: for each data item do
3:     Pick M DCs from pool
4:     for each DC do
5:         Add antigen(DataLabel) to antigenCollected list
6:         Update input signal concentrations
7:         Calculate concentrations for output cytokines
8:         Update running total of each output cytokines
9:         if total cms > fuzzy threshold then
10:            Remove DC from the pool and migrate
11:            Create new DC
12: for each DC that migrates do
13:    if concentration of semi > mature then
14:        antigenContext ← semi
15:    else
16:        antigenContext ← mature
17: for each antigen that entered the system do
18:    Calculate the number of times presented as mature or semi
19:    if semi > mature then
20:        antigen ← benign
21:    else
22:        antigen ← malignant
```

Initially, a dendritic cell (DC) pool is created. For every data item in the dataset, M DCs are randomly selected from the pool. Each of these DCs then performs a series of actions. The selected DCs gather an antigen corresponding to the data label, after which they adjust their input signal concentrations accordingly. Subsequently, the DCs compute the output cytokine concentrations and update the cumulative counts of each cytokine type. If the total count of cms (safe signal added to danger signal) exceeds a specified fuzzy threshold, the DC is removed from the pool, simulating migration. To maintain the size of the DC pool, a new DC is created. The algorithm then evaluates each migrating DC to determine whether the concentration of semi-mature cells surpasses that of mature cells. Depending on the comparison result, the antigen context is assigned as either 'semi' or 'mature'. Lastly, the algorithm assesses each antigen introduced into the system. The counts of times an antigen is presented as either 'mature' or 'semi' are calculated, and based on these counts, the antigen is classified as either 'benign' or 'malignant'.

Figure 2.5 also shows an overview of the possible states of a DC.

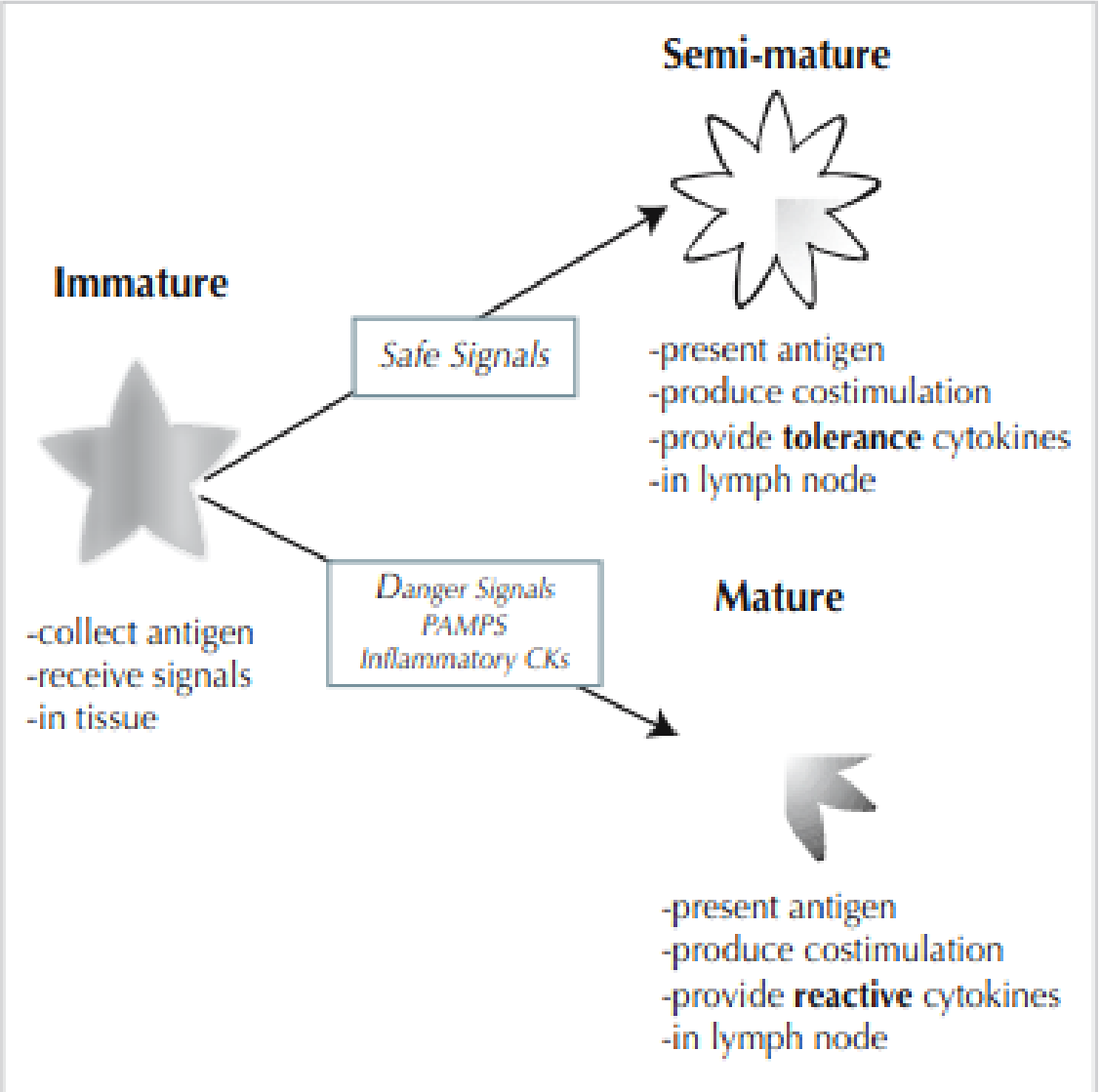

Figure 2.5: Dendritic cell (*in vivo*) maturation stages [18].

There are known limitations of the traditional DCA algorithm, namely that it can only classify data after processing the entire dataset, heavily impacting the size of datasets that can be classified with the DCA. This also means that there are no online or real-time classification capabilities. As pointed out in [18], the Dendritic Cells in an anomaly detection system should be part of a whole [system] and not entirely isolated. Thus, subsequent research around DCA focuses on eliminating these limitations, either directly increasing the sizes of viable datasets or changing the architecture to allow online classification or its use in more distributed environments.

2.2.1.1 The Cursory Dendritic Cell Algorithm

The Cursory Dendritic Cell Algorithm (CDCA) is a version of the DCA that has three main features when compared to the "traditional" DCA: it is supposed to be modular, it should offer near

real-time classification capabilities, and it should also allow for a data-driven approach for the pre-processing stage [41].

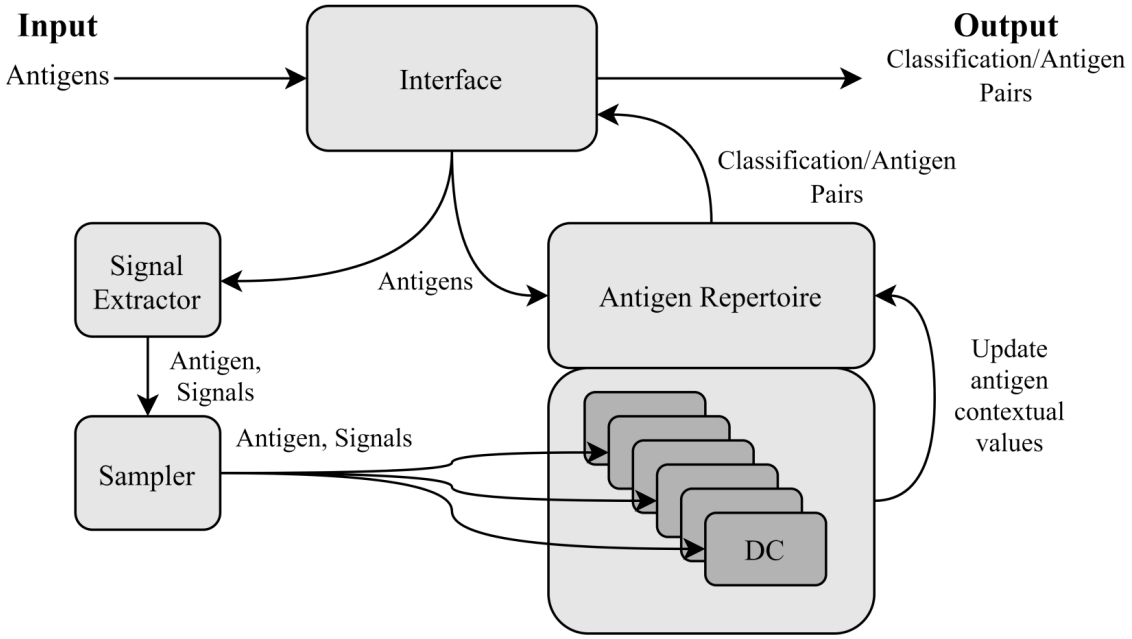

Figure 2.6: CDCA components overview [41].

Figure 2.6 shows the main components of the CDCA. The Signal Extractor is responsible for the signal extraction logic. It outputs danger and safe signals from antigens and associated data. PAMPs and inflammatory signals are not used in the CDCA, though the reason for that is not explicit. The Sampler is responsible for distributing antigens and associated signals across the available dendritic cells. The Antigen Repertoire is responsible for storing all antigens received by the system, which can then be queried by the Interface to get data and context to achieve classification.

One of the main purposes of the data-driven approach is to solve the classic limitations of the DCA: problem modeling. Usually, the DCA requires expert knowledge to map antigens to appropriate signals. However, being that one of the purposes of the CDCA is to provide online classification, being able to easily select a range of data analysis algorithms is a suitable way to solve this limitation. In the paper presenting and defining the DCA, the example algorithm used was *KMeans* [41].

In order to provide online classification, the CDCA classification flow works as follows: firstly, every time a DC matures, the MCAV values for all its antigens are computed. If every DC storing a specific antigen has already matured, but the MCAV value is still below the predetermined threshold, it is considered safe. However, if, at some point, the MCAV value passes the threshold, the antigen is classified as anomalous. This means that a data point can be classified as normal or anomalous before processing the entire dataset, but the value can be updated after the whole processing [41].

This is the version of the DCA that we are going to use to make performance comparisons against the proposed solution in Chapter 5. We are not going to use the CDCA in the implementation of the proposed solution, and the main reason for that is due to the availability of the source code of the CDCA, and implementing it from scratch would take too much time. Also, using a more complex version of the DCA might compromise the clarity of this first specification of a distributed DCA for a blockchain environment.

Chapter 3

Related Work

3.1 Distributed Dendritic Cell Algorithm

[10] proposed a distributed DCA based on the Spark framework for distributed data processing. Its main purpose was to solve the classical DCA memory inefficiencies since there must be m copies of each data point, where m is the antigen multiplier. This makes applying the classical DCA to Big Data problems infeasible. The algorithm works by making a Distributed File System (DFS) available to the algorithm and partitioning the data in disjoint sets. Then the data is processed in a manner similar to the MapReduce framework. It was tested in a high-performance computing facility across data sets of 10,20,30, and 40 million data points and obtained speed-ups of up to 16 times. However, this algorithm works under specific partition conditions and might not be suitable for real-time IDS in a decentralized system unless specific nodes are assigned specific types of antigens and signals.

Figure 3.1 shows the overall schema of the MapReduce framework.

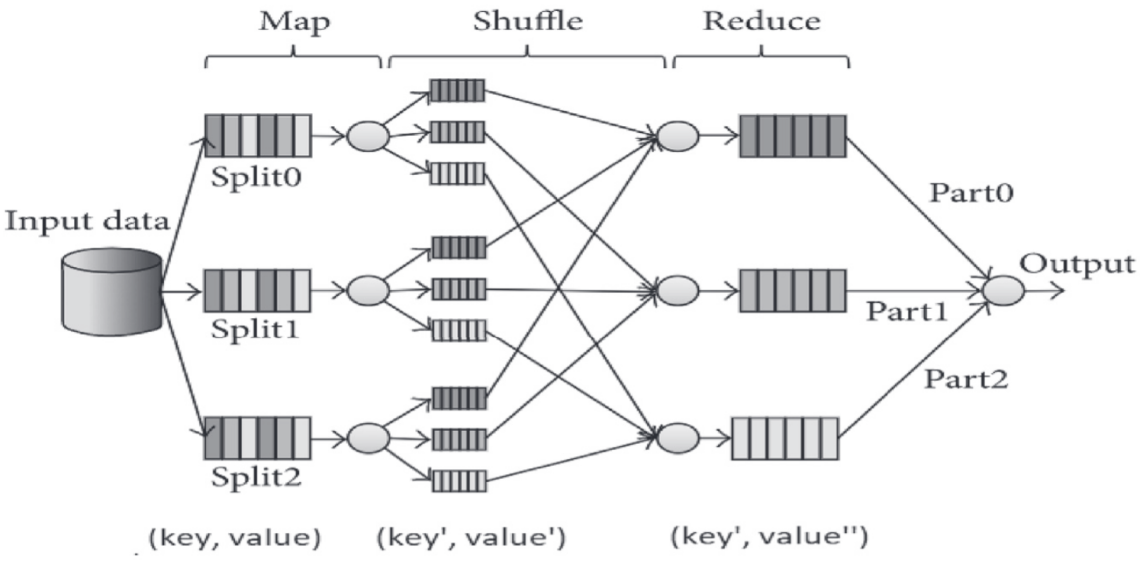

Figure 3.1: MapReduce framework [10].

The overall steps of the algorithm are:

- **Initialization phase** In this phase, a dimensional reduction is performed, and each input is assigned to its category: Danger Signal (DS), Safe Signal (SS), or PAMP.
- **Detection phase** In this phase, a signal dataset is generated. The signal attributes from the first phase are combined with the antigens.
- **Context assessment phase** The DCA processes each input and generates three values that accumulate over time. This value is then used to decide if the antigen is anomalous or not.
- **Classification phase** The DCA calculates the context value for all copies of the antigens and measures how much of the cells are fully mature.

Figure 3.2 visually represents the flow of the sp-DCA.

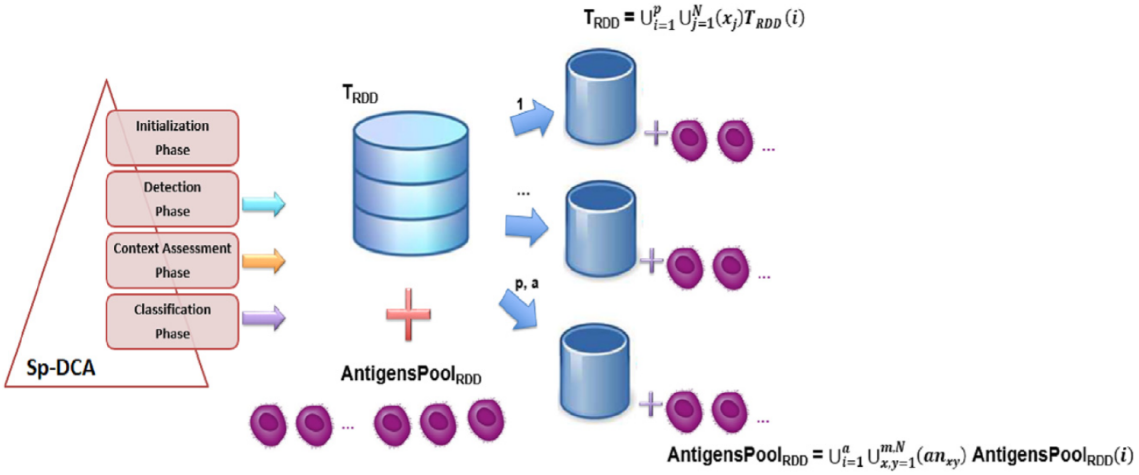

Figure 3.2: sp-DCA flowchart [10].

The focus of this work, as mentioned at the beginning of the section, is to solve the problem of scalability when traditional DCAs are used with large datasets. The paper does not explore the suitability of this solution in a blockchain context, and its main distributed component is in the data processing phase only: the data has to be later sent to a single component.

3.2 Agent Based Artificial Immune System

[40] proposed an IDS architecture based on the Danger Theory and on Multi-Agent Systems (MAS). The motivation was to combine both areas to generate a better IDS. The paper proposes four agents:

- **DC Agents** Agents that simulate the functionality of the Dendritic Cell.
- **T-Cell Agents** Agents that calculate thresholds from danger signals detected by dendritic cell agents that exceed a certain threshold.

- **Antigen Agents** Agents that parse inputs from the environment into antigen format.
- **Responding Agent** Agent installed in a central entity that receives signals that exceed a threshold from other agents and classify it as malicious or not. The central entity is then responsible for activating a countermeasure.

Figure 3.3 shows the diagram of the architecture. Hosts only communicate with each other via the TC agents, and these agents are responsible for communicating with the Security Operations Center (SOC).

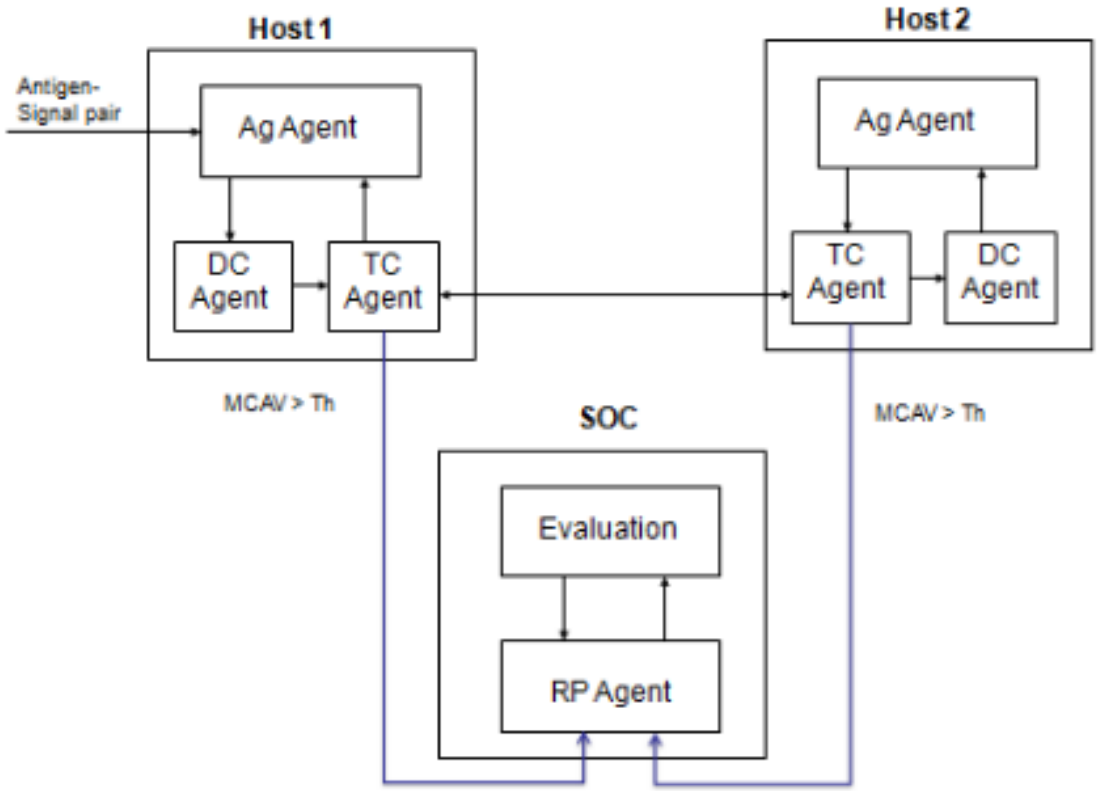

Figure 3.3: ABAIS Architecture diagram [40].

The idea is that TC agents in the different hosts will exchange threshold values between different hosts by sending the values to nearby hosts and receiving values from nearby hosts. The dendritic cells are responsible for analyzing antigens provided by the antigen agent, and PAMPs, danger levels, and safe levels are calculated. Then, results are sent to the TC agents, which are responsible for warning the central entity.

The work describes the self-organizing feature map (SOM) for initial clustering input. The advantage of this solution is that it is a distributed architecture, and the paper already describes a way of exchanging information between agents and hosts. However, the suitability of this system is not explored and not even encouraged since it relies on a central authority for final classification and response.

3.3 DeliveryCoin: Blockchain and IDS

[16] proposes an IDS for a blockchain used in a drone delivery service. The blockchain utilizes Practical Byzantine Fault Tolerance (pBFT) to achieve consensus. Also, there are five network entities: buyer entities, vendor entities, the delivery service, unmanned aerial vehicles (UAV) and antennas. Figure 3.4 shows the high-level threat model of the network.

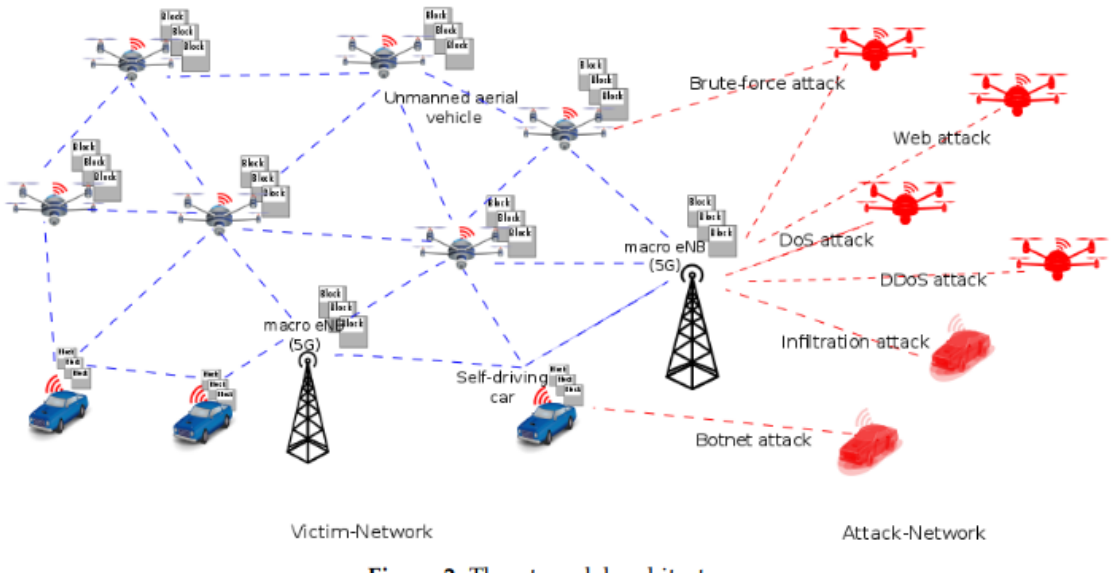

Figure 3.4: Threat model [16].

In this architecture, the *macro eNB* represents a ground component that supports the edge devices (vehicles). The IDS algorithm runs at each of these components. The IDS is a machine learning algorithm that was previously trained with the dataset. Figure 3.5 shows the IDS diagram. There is little detail on fine-tuning details regarding the IDS, but much like other works in IDS research, the data points are network packets.

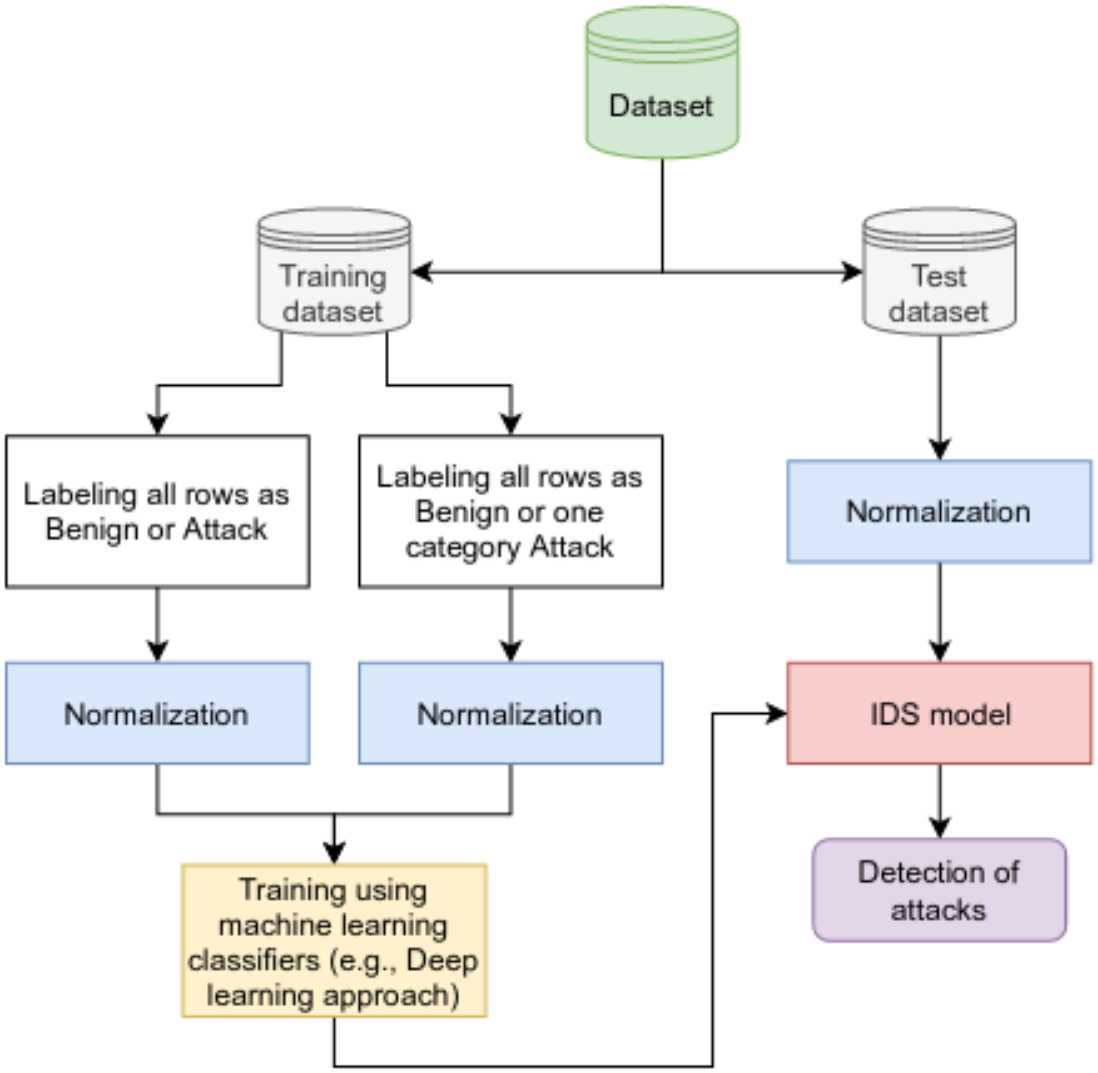

Figure 3.5: IDS model training scheme [16].

Some of the attack scenarios(explaining each one is beyond the scope of this work) used for training are:

- **SSH and FTP Brute force**
- **Infiltration**
- **Web attacks (e.g. SQL Injection)**
- **DOS/DDOS**

This work has some characteristics that are similar to what we propose to do, like an IDS for a blockchain. However, it uses an IDS that is not related to Danger Theory, nor to AIS, for that matter. For some algorithms, it presented accuracy in detection of 90% and higher. This is an interesting result, showing that an IDS in a blockchain environment could be suitable and provide satisfactory detection capabilities.

3.4 Distributed Consensus Mechanism with Novelty Classification Using Proof of Immune Algorithm

[2] proposes a novel consensus mechanism for a peer-to-peer lending system that adapts the DCA. From the abstract: *"Proof of Immune Algorithm was proposed by leveraging the potential dendritic cell algorithm mimicking human immune system to provide consensus among peers involved in the lending process to enable trust"* [2].

In the paper, the Proof of Immune (POI) is introduced. It differs from mainstream consensus algorithms in that the algorithm does not intend to form an equal view from the perspective of borrowers and lenders but accepts that each entity will have a different level of trust in relation to each other entity in the network. It does so by defining a Danger Quotient (DQ) and computing the trust between an entity and another entity. The DQ is based on danger theory and dendritic cells, and it is calculated as Self-Danger Score (SDS) and Neighbourhood-Danger Score (NDS). These are the definitions of such scores:

- *SDS* It is a score based on the participation in transaction events by a given peer. The author characterizes it as a sort of "Karma" of the peer in the network, and it is computed by:

$$SNS = (failure - contexts)/(total - validations) \tag{3.1}$$

  The paper works with a borrower/lender network, but the default values for a peer's SNS are use-case specific.

- *NDS* It is a score based on successful requests from borrowers and from lenders. It has the primary purpose of identifying trustworthy borrowers for lenders and maintaining an overall healthy activity in the network. It is computed by:

$$NDS = (\sum acceptances)/(total - requests) \tag{3.2}$$

From the above definitions, 3.1 and 3.2:

$$DQ = SDS/NDS \tag{3.3}$$

$$Trust = 1 - DQ \tag{3.4}$$

The results obtained were as follows:

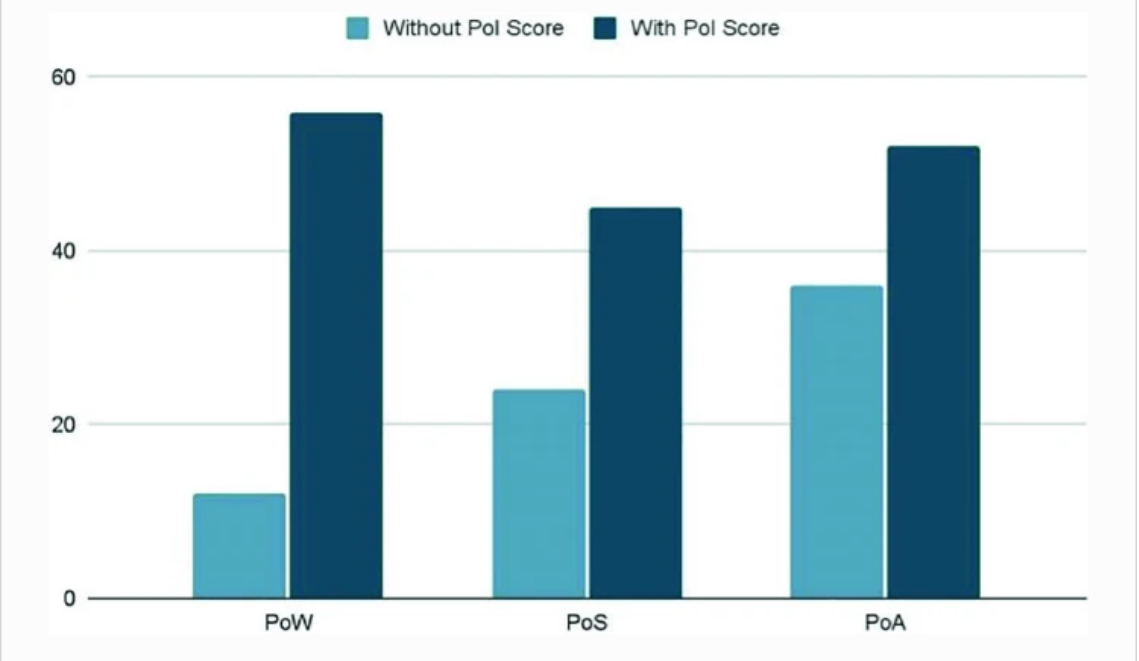

Figure 3.6: Performance of PoI with standard consensus approaches [2].

Significant improvements in the ability to detect suitable borrowers were obtained, and the POI can also be used to build consensus on other domains besides banking.

3.5 Comparison

Table 3.1: Related Work comparison.

	DCA	**Distributed**	**Blockchain Environment**	**IDS**
Distributed DCA	Yes	Yes	No	Yes
Agent Based AIS	Yes	Yes	No	Yes
DeliveryCoin	No	Yes	Yes	Yes
Proof of Immune	No	Yes	Yes	No

Table 3.1 shows us that, even though there is research around some of our requirements, it seems that a DCA-based IDS in a blockchain environment has not yet been studied. It is also important to note that, even though the *Proof of Immune* paper references dendritic cells, it does not implement the DCA but instead tries to mimic the behavior of dendritic cells regarding danger signals: it is not properly a DCA adapted to a consensus algorithm.

Overall, while there is some research concerning the distributed properties of the DCA, most of them do not cover the suitability of the DCA in a blockchain environment, or it is not used as an IDS but rather as a consensus mechanism. There is still a gap in the understanding of the suitability of the DCA to a fully decentralized environment of the blockchain as a satisfactory IDS. In the next section, we define a concrete use case for a problem and start the outline of a possible solution.

Chapter 4

Proposed solution

4.1 Requirements

As demonstrated by the state-of-the-art analysis, and with the context and motivation in mind, there is a gap in research studying the DCA in blockchain environments. That said, we propose with this work an architecture for the DCA with the use of blockchain networks in mind.

Firstly, as mentioned in the introduction, Industry 4.0 is characterized by highly distributed systems, so an IDS that matches these distributed topologies would be ideal while also removing the reliance on a central entity for data processing. This first requirement is going to be achieved by distributing the data processing and classification tasks across the nodes in the network.

Secondly, industrial blockchains are usually permissioned, resulting in a lower number of nodes when compared to public blockchains. This characteristic allows us to use the blockchain itself as a communication layer, reducing the number of dependencies and guaranteeing that the communication is closely attached to transactions and actions in the blockchain layer. This is going to be achieved by implementing the communication between nodes as events that are emitted from the blockchain.

Lastly, in order to reduce overall classification error, besides distributing the classification tasks between nodes, the nodes should achieve a consensus regarding the classifications by utilizing their local classifications as a vote. This would help reduce classification error (as it is going to be demonstrated in Section 5) while also allowing the voting process to happen if a minority of nodes fail (not tested in this work, but it is a characteristic of the architecture). This is going to be achieved by executing a majority voting with the active nodes in the network to achieve a final classification.

To summarize, these are the requirements of the proposed solution:

- The architecture should distribute the data processing across the multiple nodes in the network.
- The nodes should communicate using the blockchain as a medium.

- The nodes should be able to communicate their local classification in order to reach a consensus for a final classification, reducing classification error in the process.

4.2 High-level description of the solution

The proposed solution has the following high-level architecture:

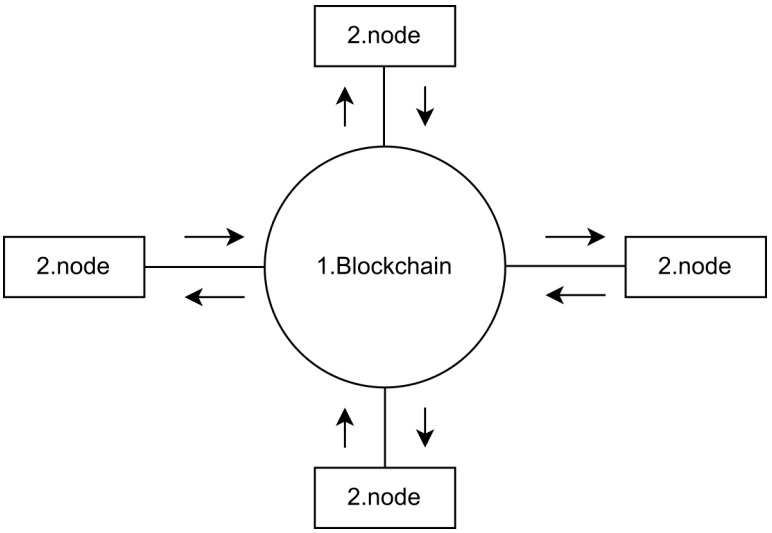

Figure 4.1: High-level architecture of the solution.

In this solution, the blockchain is the only communication channel between the nodes: the nodes use blockchain events t communicate with each other. A node is an abstraction of an entity that interacts with the network by sending data points to the network to be classified by other nodes. Naturally, the nodes could be thought of as devices that send sensor data to the network. Every time a node pushes data to the blockchain, a network event is triggered and captured by the other peers (how exactly this process works is dependent on the blockchain layer, and how it works in our specific layer is going to be described in the next section). After capturing the necessary data from a particular node, a node will then classify the data from each node with separate local DCA instances, and send the result of its local classification as another network event, initiating the voting process. After all the nodes receive the classifications from every required node, the voting process processes all the classifications, generating a final classification.

4.3 Implementation

4.3.1 The blockchain layer

Ideally, the architecture should be blockchain agnostic, and it would be up to the nodes to properly implement adequate communication with the blockchain to allow the nodes to retrieve data published to the chain by other nodes. Thus, the blockchain technology to be used is going to be

selected based on how much its properties and Application Programming Interface (API) facilitate the mapping between the high-level architecture and the actual implementation.

4.3.1.1 Hyperledger Fabric

4.3.1.2 Overview

Hyperledger Fabric is an open-source blockchain framework developed by the Linux Foundation. One of its primary design principles is to provide a modular architecture, allowing for flexibility and customization. These characteristics make it well-suited for a variety of use cases. It is a permissioned blockchain with secure and private transactions between multiple parties.

Hyperledger Fabric provides tools and APIs to enable developers to build applications that interact with network events. The available Software Development Kits (SDKs) provide a set of APIs that enable developers to subscribe to network events emitted from smart contracts besides common blockchain interactions ([22], [23], [24]).

4.3.1.3 Smart contracts

Hyperledger Fabric's smart contracts, known as chaincode, are one of the most important aspects of the framework. Chaincode is responsible for executing transactions and for the implementation of the business logic of the system. Particularly, Hyperledger Fabric supports chaincodes in Golang, Javascript, and Java.

The chaincode can be accessed and invoked by any authorized participant on the network. Every time a transaction (in our context, the publishing or retrieval of an data point or classification) is submitted, it is validated by the ordering nodes and passed to the endorsing peers after that. The endorsing peers, which are the peers responsible for validating transactions , are then responsible for executing the chaincode and generating a response containing the result of the transaction and a digital signature. In case that the transaction is successful, it is returned to the ordering peers to be added as a new block on the blockchain.

Chaincode is designed to be modular, allowing developers to create logic that can be easily integrated with the blockchain network. It can also be updated and/or replaced without network disruption.

Smart contracts provide a number of benefits over traditional contracts: immutability, meaning that once a chaincode is deployed, it cannot be changed. It is also self-executing, meaning that it can automatically execute when certain conditions are met, such as when a certain date is reached or an event occurs. This makes chaincode suited for industrial settings, such as supply chain management, for example ([22], [23], [24]).

4.3.1.4 Events

Hyperledger Fabric provides a robust event system enabling the reception of real-time network events. Events in Hyperledger Fabric are generated by different layers and components of the

network, such as endorsing peers, orderer peers, and chaincode execution. These events can be used to monitor the state of the network and set triggers for specific events.

Hyperledger Fabric events are published through channels, a logical construct that allows for private communication between multiple parties, meaning that generated events are not public. Clients can subscribe to specific channels and receive notifications when events occur on the subscribed channels. Events can be filtered and searched based on a number of criteria, such as event type, transaction ID, and block number.

Chaincode events, which are more relevant to our purposes, are events triggered by the execution of chaincodes on the network (only a chaincode that emits an event explicitly in its logic generates a chaincode event). These events allow applications to react to specific events emitted by the business logic. Transaction events are generated when a transaction is successfully validated and committed to the ledger. Block events are emitted when a block is added to the network. These events contain metadata associated with their respective source, and in the case of chaincode events, they may contain a payload. These events can include information about the health of the network, such as node status and resource usage, or contain information about actions being performed by nodes that need to be captured by the rest of the network.

In conclusion, the event system provided by Hyperledger Fabric provides a powerful and flexible way to monitor the network and trigger event-specific actions. The system enables clients and applications to receive real-time notifications about network events, which can be used to create more efficient and effective blockchain solutions([22], [23], [24]).

4.3.1.5 Decision overview

Although the blockchain itself doesn't have an impact on the architecture, the Hyperledger Fabric's properties would greatly facilitate the implementation of the communication since it provides chaincode events, which serve as a direct mapping to the high-level architecture's communication: a node will send data to the blockchain (via chaincode), and this chaincode will trigger an event, that will be captured by other nodes, enabling the classification and voting process.

4.3.2 Our chaincodes

The following listings show the chaincodes used in this work. In all the listings displayed below, the *ctx* parameter refers to the internal hyperledger context, which is not important for our purposes.

Listing 4.1: Put Antigen.

```
1 async put(ctx, key, value) {
2     await ctx.stub.putState(key, Buffer.from(value));
3     ctx.stub.setEvent("putEvent",Buffer.from(value));
4     return { success: "OK" };
5 }
```

Listing 4.1 represents the chaincode that nodes use to send a reading to the blockchain. It also sends a "putEvent" event that nodes listen to and react to. It allows nodes to react to new readings instead of having to poll the ledger and check for new readings. There is also a "putMultiple" version of this chaincode to allow sending multiple readings in a single transaction.

Listing 4.2: Get Antigens.

```
async list(ctx) {
    const allResults = [];
    const iterator = await ctx.stub.getStateByRange('', '');
    let result = await iterator.next();
    while (!result.done) {
        const strValue = Buffer.from(result.value.value.toString()).toString('utf8');
        let record;
        try {
            record = JSON.parse(strValue);
        } catch (err) {
            console.log(err);
            record = strValue;
        }
        allResults.push(record);
        result = await iterator.next();
    }
    return JSON.stringify(allResults);
}
```

Listing 4.2 represents the chaincode that nodes use to list the readings in the blockchain ledger. It allows peers to get all current readings stored in the ledger. Its main purpose is to facilitate testing, so we do need to wait for every peer to generate each reading individually· If, for any reason, an asset that cannot be parsed to JSON was submitted to the ledger, the *catch* clause will guarantee that the record is still retrieved as a string. This is mainly due to testing and debugging purposes, and it is not particularly relevant to the overall flow of the architecture since dealing with errors is not the purpose of this work.

Listing 4.3: Classify antigens.

```
async classify(ctx, key, value) {
    ctx.stub.setEvent("classificationEvent",Buffer.from(value));
    return { success: "OK" };
}
```

Listing 4.3 represents the chaincode that nodes use to send their votes to the other nodes. The chaincode emits a "classificationEvent" to the other nodes so that they can process that peer's votes. The votes are not stored in the blockchain for usability purposes.

It is worth reiterating that these events are the main reason that led to the decision to use Hyperledger: it facilitates development and allows nodes to react to events instead of polling the ledger.

4.4 The application layer

4.4.1 The classification process

In this architecture, each node is responsible for running the DCA locally, with its own data and the data originated from the other nodes in the network. The distributed component takes place as a voting system in which each node shares its local classification of the data of every other node with all the nodes to achieve a consensus regarding each individual data point. The following diagrams visually represent the steps of the classification process.

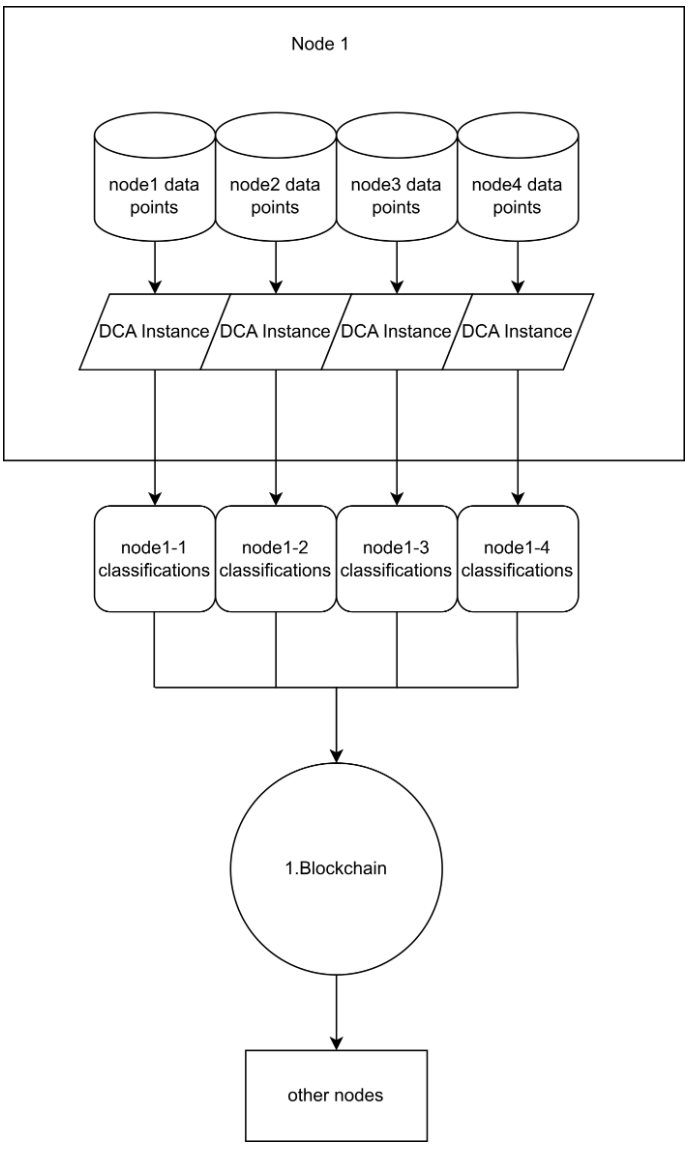

Figure 4.2: First step: a node sending its local classifications to the network.

Figure 4.2 represents the first step to be taken by each node in the network: First, it will classify its own data points with its local DCA. After that, it will retrieve the data points of each other node separately and classify each set with the DCA as well. The data points from another

node can be retrieved in one of two ways: when a node emits data to the network, an event with the data as the payload is emitted and captured by the other nodes in the blockchain. The other way to retrieve data is to query assets from the distributed ledger, since the data points are stored as assets in the blockchain. The classification mode will determine how the data points will be retrieved, and these modes will be explained later in this section.

After it has classified the data points of each node, it is going to send its local classifications to the ledger, and each classification will generate an event with the classifications as the payload, and these events will be processed by the other nodes in the network. After that, the node waits, until itself has received all the classification events from all the other nodes in the network, and it will then compute the votes and generate the final classification.

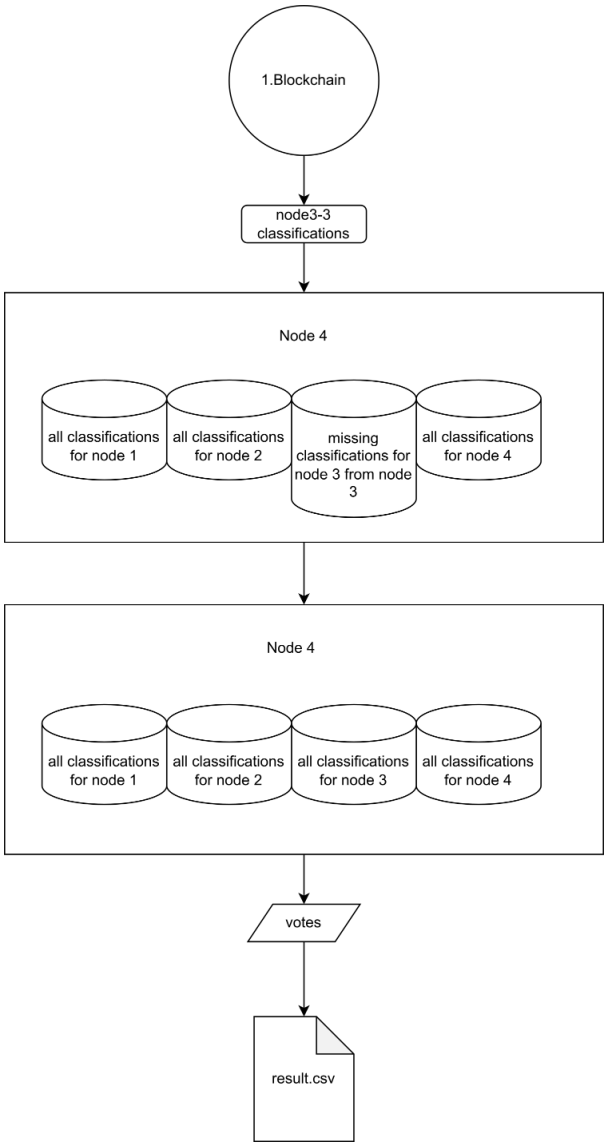

Figure 4.3: Second step: a node executing the voting process after it has received all classifications from all peers.

Then, Figure 4.3 represents what happens after a node finishes receiving all classifications from the other nodes and it has also finished its own classifications: it then uses the classifications to vote on the consensus for each data point and generates the result file.

There are two modes of classification. The first one is the "normal" mode. In this mode, each node listens to the network events, and after a predetermined amount of data points from a specific peer is received, the classification process for that peer's data points begins. This mode is more suited for real environments, where each node emits data constantly in real-time. This does not solve the DCA's limitations: After a classification or after a specific DCA instance's DCs expire their lifetimes, the maturation process must be done again. The peers listen for data in real time, but the classification process only takes place after enough data has been captured.

The other mode is the "dataset" mode. In this mode, a dataset is stored in the blockchain ledger, by committing a series of transactions issuing the data points as assets, with the necessary metadata such as the node of origin and ID, to allow retrieval and classification of these data points. After that, each peer will get the necessary data from the ledger at once and classify the data points in the dataset. This was developed to help with validating the proposed solution with CPS datasets, while the "normal" mode is more suited for a real testbed/scenario.

4.4.2 Voting process

This architecture does not depend on a single type of voting. The following pseudo-code (2) illustrates how a node would register its vote and the vote of other peers.

Algorithm 2 Basic voting.

```
1:  function ADDVOTE(electionId, voterId, vote)
2:      temp ← electionMap.get(electionId)
3:      if temp.size = votesPerElection then
4:          throw "Maximum votes reached!"
5:      temp.set(voterId, vote)
6:      areElectionsOver ← true
7:      for all [electionId, voteMap] in electionMap do
8:          if voteMap.size ≠ votesPerElection then
9:              areElectionsOver ← false
10:     if areElectionsOver then
11:         emit-event("election-over", electionMap)
```

In the above algorithm, the *electionId* refers to the peer whose data points are being voted for, *voterId* refers to the peer that is adding that specific vote, and *vote* is a list of all antigens from peer <*electionId*> that were classified by peer <*voterId*> 's DCA and its respective classification. The value *votesPerElection* is an aspect of the type of voting taking place. If there is already the required number of votes for a specific *electionId*, any further votes received will throw an error and will not be considered. Further handling of this scenario is out of the scope of this work and would depend on the particular use case.

In this work, it is equal to the number of nodes. That is, an election (there is one election per peer) is over when all peers have received the classifications from all other peers. This value would not need to be equal to the number of nodes. For example, if there are *N* nodes in the network, with *N* greater than or equal to 4, each node may need to only receive classifications from *N/2* other nodes (or any even number for larger networks), or there may be an entirely different voting logic.

One particular example that might be worth mentioning, as it was considered by the authors in the early stages of this work, is by having each node receive only the classifications of the "closest" nodes. This definition of *close* could be the XOR between two nodes: as demonstrated in [34], the XOR can be used as a distance metric. Alternative voting logic like this one might be needed in more complex/larger networks, in which receiving classifications from all peers might be too time-consuming. This idea was then later abandoned because testing different voting alternatives or focusing too much on them is outside of the scope of this work.

4.4.3 The result format

The entire classification process generates one file that has the following items:

- An ID column, in the format *antigen-peer<peer-number>-<antigenId>*
- A value column representing the value that was used as antigen for that particular point.
- One column per peer, with the column name being the peer ID and the value being the label attributed by that peer to that particular data point.
- A *final label* column, representing the consensus about that particular point.
- A *correct label* column, representing the actual value of that particular point.

For columns with label values, that are two possible values are *Normal* and *Anomaly*. If the votes for a particular data point result in a draw, there are no particular criteria taken into consideration: implementation-wise, if there is a draw, the first label in an array will be chosen as the final label.

Also, there is something important to keep in mind: since all nodes share their respective classifications with the network, this means that by default, the result generated by every node is the same since every node has the classifications made by every other node, so, by default, unless some behavior is added to each peer, we only need to make one of the peers generate the result file. The following figure shows a couple of lines of such a result file generated by our first test, which will be discussed further in the next section.

```
antigenId,value,peer0,peer1,peer2,peer3,final_label,correct_label
antigen-peer2-0,87,Anomaly,Anomaly,Anomaly,Anomaly,Anomaly,Anomaly
antigen-peer3-0,88,Anomaly,Anomaly,Anomaly,Anomaly,Anomaly,Anomaly
antigen-peer0-2,90,Anomaly,Anomaly,Anomaly,Anomaly,Anomaly,Anomaly
antigen-peer2-2,36,Anomaly,Normal,Normal,Anomaly,Normal,Normal
antigen-peer0-3,69,Normal,Normal,Normal,Normal,Normal,Normal
```

Figure 4.4: Example result format.

4.4.4 The DCA

As stated in the previous section, the architecture is composed of multiple nodes, and each node uses the traditional local version of the DCA. The exchange of data points and events through the blockchain and the communication of the results from each peer is what makes the distributed component of the architecture.

That being said, over the years, multiple variants of the DCA have emerged, each with its own advantages and shortcomings. Our architecture is not bound to a single version of the DCA: As long as the researcher is able to get the data from the blockchain to be used in the algorithm, that version of the DCA will be able to be used with the architecture.

In this work, we are using a more traditional version of the DCA, similar to its first conceptualization: it does not have online capabilities, like the cursory DCA, nor is it completely deterministic, like the deterministic DCA.

Listing A.1 in the appendix shows the implementation of the DCA that we used. This version is an implementation by [8]. The final code that we used is not presented because it was adapted to an object-oriented paradigm, and the visibility and clarity would be decreased in relation to the original version we present here.

4.5 Architecture Review

As mentioned in Section 1.4.2, to study the feasibility of using a DCA-based IDS in a blockchain environment, the proposed architecture has to be *distributed*. The developed architecture fulfills this requirement: the IDS does not depend on a single entity, and the multiple nodes communicate with one another to achieve a final classification. This is made possible by each node receiving the data points from each other node, classifying the data points from each node individually, and communicating the classifications to the network. Then, after each node has received the classifications from the required amount of nodes (in our case, the required amount of nodes is equal to the number of nodes), the final classification of each data point is decided by the majority class from individual classifications (voting process).

However, the optional requirement of minimal redundancy was not reached. This is due to two main reasons: The first reason is that when a node emits a datapoint to the network, it generates an event (also containing the datapoint itself as the payload). Even if some nodes ignore this event, the event is processed, so the overhead of transmitting every event to every node exists. This

is a "limitation" of the technology being utilized (Hyperledger Fabric). This could be solved by creating more than one hyperledger channel and assigning nodes to a new one once the available channels are full, with the purpose of limiting the number of events each node receives and better spreading the architecture communication.

The second reason is not caused by the architecture itself but by the implementation. Although the architecture itself does not depend on this specific voting system, we implemented a voting system that requires every node to classify the data points of every other node, including itself.

Chapter 5

Testing & Validation

5.1 Test setup

5.2 Methodology

For performing the tests described in this section, we used a computer with an AMD hexacore processor with 12 threads and 3.6 GHz, with 32GB of RAM. We used 4 *hyperledger fabric* nodes running on this computer. Unless specified, the plotted results are an average of 5 runs.

In order to assess the suitability of our architecture, three tests are going to be performed. For the first test, we are simply going to use a dummy dataset with randomly generated values (one range of values for "Normal" and another for "Anomaly" values). For the second test, we are going to validate the solution with the OPC UA dataset [41]. And for the third and final test, we are going to use the SKAB dataset [45]. For every dataset, we are assessing accuracy and f1-score (to account for the distribution of labels in the dataset) for performance. For direct comparison between the performance of the solution and the performance of other work, and the performance of using the DCA locally, other metrics such as False Alarm Rate (FAR), Missed Alarm Rate (MAR), and *Receiver Operating Characteristic - Area Under the Curve* (ROC-AUC) may be used, depending on the dataset. Here is a brief description of these metrics:

- **FAR** It represents the rate of false alarms. It is represented by the ratio between the number of emitted false alarms and the total number of anomalous data points.
- **MAR** It represents the rate of missed alarms. It is represented by the ratio of false negatives and the total number of anomalous data points.
- **ROC-AUC** It represents the trade-offs between True Positive Rate (TPR) and False Positive Rate (FPR).

The main scope of this methodology is to test the suitability of the proposed solution as a first step towards integrating the DCA in blockchain environments. So, for our purposes, four peers are enough to demonstrate the characteristics of the *majority voting system* that we are using. Other

metrics, such as time for classification, are not being compared due to the lack of sufficiently similar work: the CDCA [41] that we are going to use to compare with our solution has a very different classification system and a different environment, so with that in mind, comparing the time between the two approaches might not be very indicative of the proposed solution's potential. Also, scalability tests with a higher number of nodes are considered to be out of the scope of this work since our main focus is to present a first attempt at integrating the DCA with a blockchain environment. The two datasets were selected to provide a frame of reference for comparison since they are tested with another DCA version in an IIoT context.

For this work, the use case that will guide the development process of the proposed solution will be that of a network of devices connected to each other through a blockchain. These devices send data to the network, be it sensor or communication data, and the data is readable by all other devices in the network. The DCA module in each device is responsible for analyzing such data, both from its own device and from the other, classifying it, and sending its results to the other devices as well.

5.3 Dummy dataset

In this scenario, the data "emitted" by the network devices is completely simulated. The feature being used in cell maturation and for the classification is temperature. The values are generated randomly, with a 70% chance of being between 25ºC and 75ºC (Normal) and 30% of being between 76ºC and 100ºC (Anomaly). We present the results for this dummy dataset here only because it was the primary dataset used during development, tuning, and debugging.

5.3.1 Test I

5.3.1.1 Performance metrics

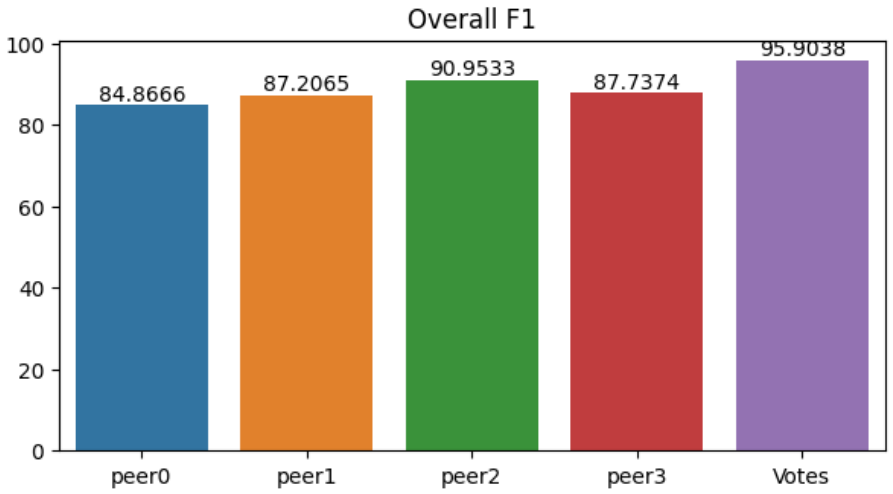

Figure 5.1: Overall f1-score of dummy dataset.

The results show that the fact that different nodes miss classifying different data points can be used to improve the overall rating. This is just a dummy dataset to work as a first proof of concept, and no further conclusions are going to be drawn from these results.

5.4 OPC UA dataset

This dataset was published by [42]. The dataset was generated by modifying packets in a CPPS testbed; it is comprised of normal data points, data points describing a Denial of Service (DoS) attack, man-in-the-middle (MITM) attacks, and Spoofing attacks. There is also a boolean flag indicating if a specific data point is anomalous (any of the three attack types) or normal.

The dataset has 32 features. However, since the version of the DCA that we are using for this work is better suited for single values, we are going to capture metrics using two different single features, one per run. The selected features are *b_pktTotalCount* and *octetTotalCount*. Both these features have a high correlation with the anomaly label.

For both runs, we are going to consider two scenarios: one scenario in which all the data points in the dataset are emitted once by each peer (in other words, the dataset is replicated by each peer) and one scenario in which the dataset is partitioned between the peers, with no particular criteria (thus allowing peers with different label distribution).

5.4.1 Test I - Replicated Dataset

5.4.1.1 Performance metrics - b_pktTotalCount

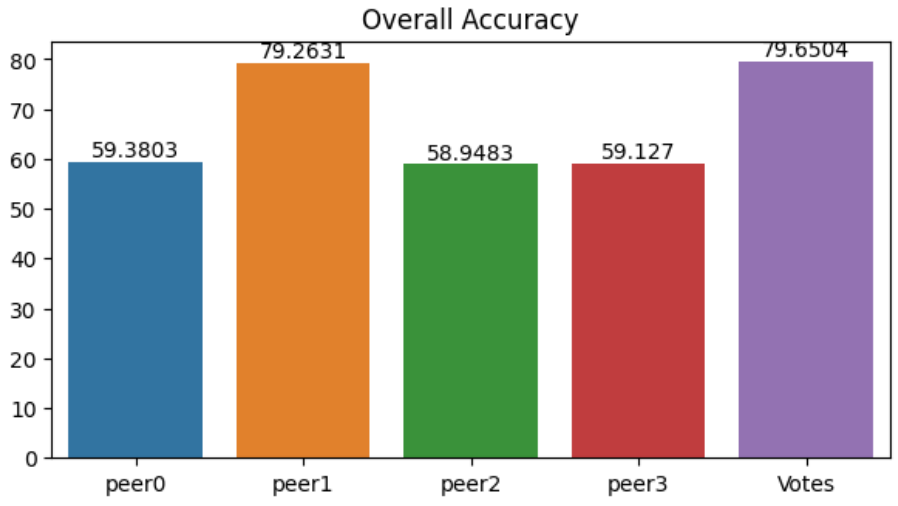

Figure 5.2: Overall accuracy (replicated dataset) using b_pktTotalCount - unique run.

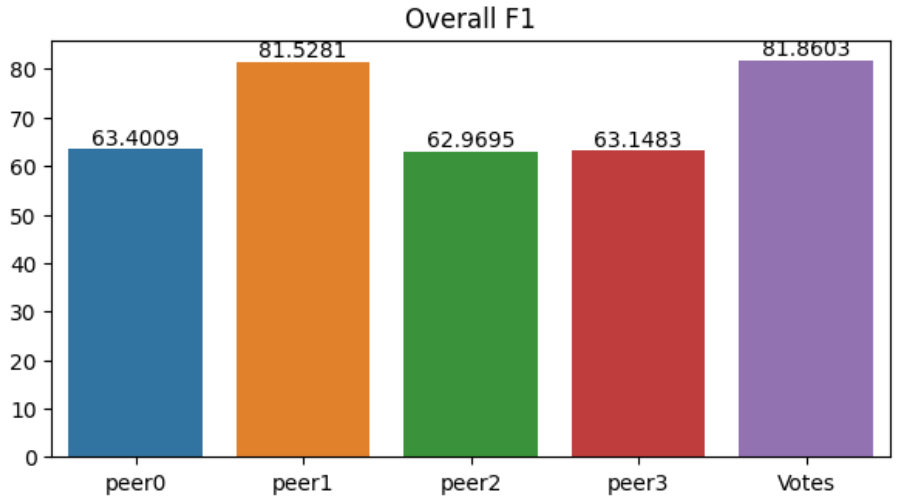

Figure 5.3: Overall F1-score (replicated dataset) using b_pktTotalCount - unique run.

For the binary feature *b_pktTotalCount*, we can see that even though three peers have had relatively poor performances (around 63% on f1 (5.2, 5.3)), one high performing peer was able to raise the voting results to higher performance as well. This might happen because, even though the three least-performing nodes had approximately the same performance, they miss-classify different data points, and in this case, if *peer1* gets the majority of classifications correct, the final results can be corrected. This is in line with what was concluded by [49], that an ensemble can result in higher performance at the trade-off of possibly lower individual performances. This happens because the variance in classifications will dilute the error across the multiple nodes, resulting in the majority voting classifying most of the data points correctly. However, the potency of this effect depends on multiple factors, and more precisely, analyzing its effect in this architecture and maximizing its effect would require further research.

This particular test scenario was not the result of averaging five runs. It is a unique test result that is worth analyzing because it demonstrates a more extreme example of the effect described by [49]. Figure 5.4 depicts the average f1-score for this feature.

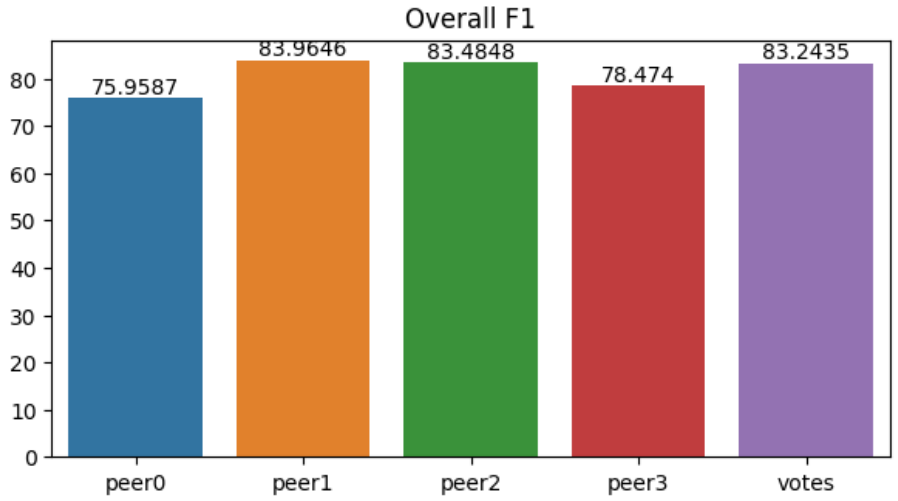

Figure 5.4: Average f1-score (replicated dataset) using b_pktTotalCount feature.

5.4.1.2 Performance metrics - octetTotalCount

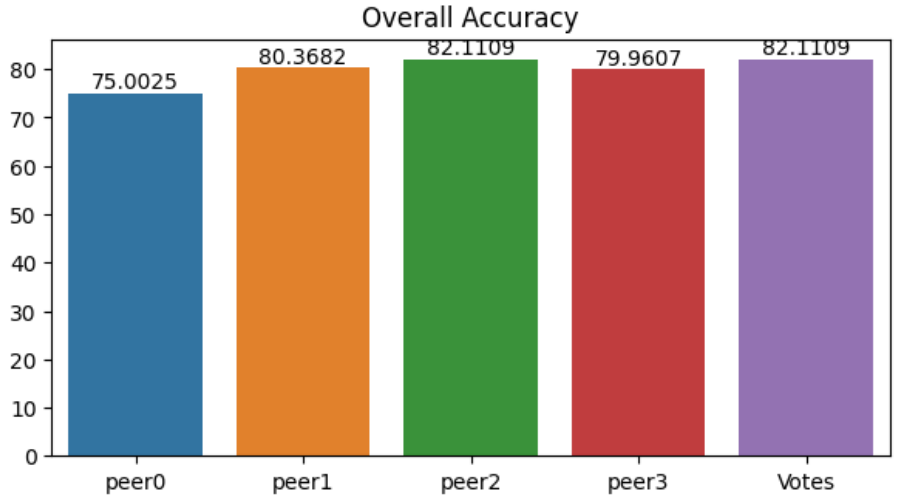

Figure 5.5: Overall accuracy (replicated dataset) using octetTotalCount feature.

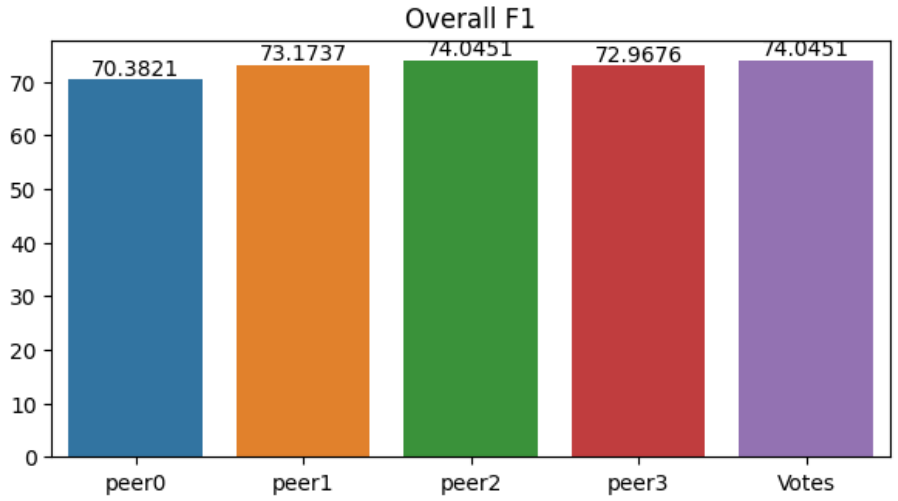

Figure 5.6: Overall F1-score (replicated dataset) using octetTotalCount feature.

In this particular scenario, when peers have relatively similar performances, there is no significant difference between the overall results and the voting results. It is important to note that this characteristic of the architecture goes both ways: with a different dataset or using a different feature of this dataset that had a lower correlation to the label if the individual nodes themselves end up with poor performance due to that fact, the vote is also going to have no positive effect over the final classifications.

Regarding individual results, it is possible that the difference between classification performance for both features is entirely due to the nature of both features: the first one, *b_pktTotalCount* being binary, and the second one *octetTotalCount* being numeric (integer numbers only). [17] mentions that the DCA is sensitive to changes in context, but it is less precise on context change points. It may be that abrupt changes in the context, such as with binary features, translate to better classification performance.

5.4.2 Test II - Partitioned Dataset

5.4.2.1 Performance metrics - b_pktTotalCount

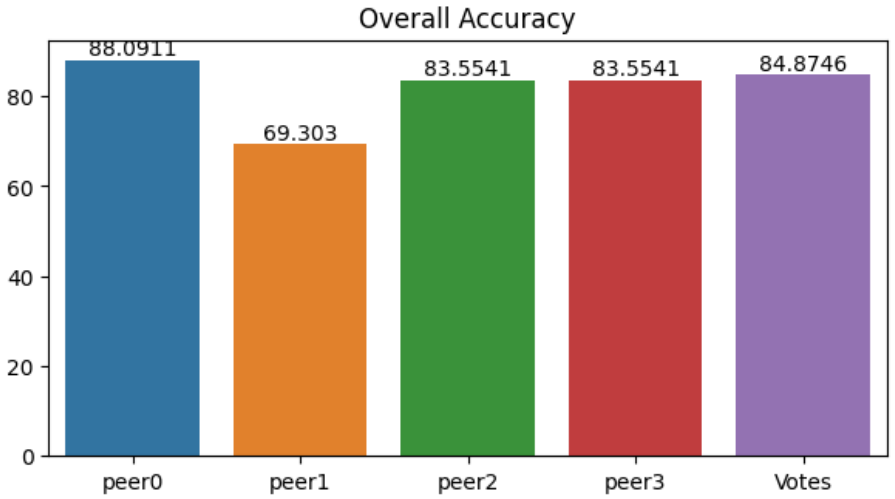

Figure 5.7: Overall accuracy (partitioned dataset) using b_pktTotalCount.

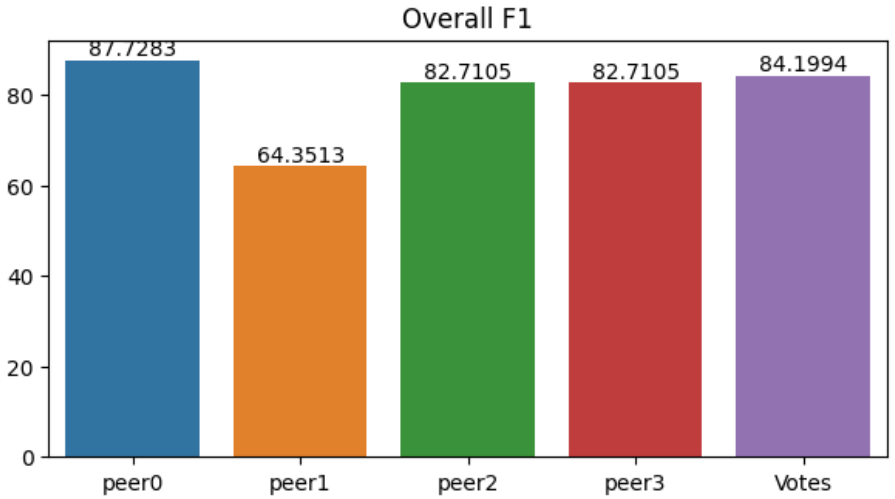

Figure 5.8: Overall F1-score (partitioned dataset) using b_pktTotalCount.

Now, with the partitioned dataset, we get the highest performance across individual nodes, of around 82% to 87% (5.7, 5.8). Particularly, we can see in this scenario that a poorly performing peer, in this case, *peer1*, does not affect the overall performance of the final results. This is straightforward: if the majority of peers have a high performance, it follows that the final votes will also have a high performance. But here, we can infer another characteristic: even a malicious node intently classifying anomalous behavior as normal (it is not the case here, but it might be a real scenario), the other nodes will be able to detect malicious behavior, as the voting results will yield anomalous classifications for the malicious node. Moreover, as long as the majority of

nodes are not malicious and have a decently performing DCA, anomalous behavior can be detected on the network. In this particular scenario, one of the nodes will perform poorly because when the dataset is partitioned into four parts, one of the peers will be responsible for simulating the behavior of the more unbalanced partition.

5.4.2.2 Performance metrics - octetTotalCount

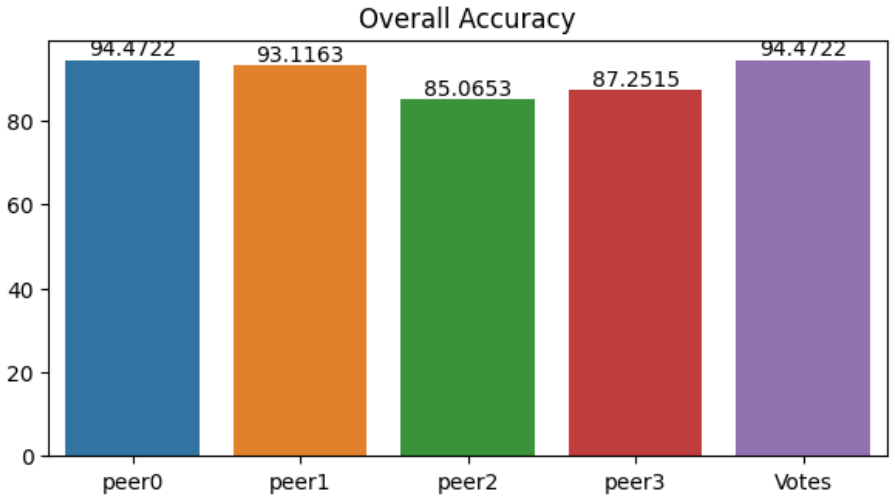

Figure 5.9: Overall accuracy (partitioned dataset) using octetTotalCount feature.

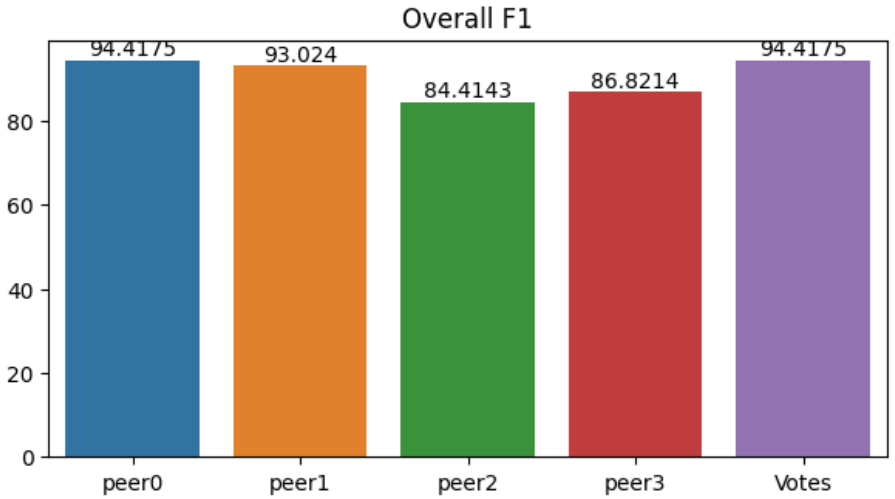

Figure 5.10: Overall F1-score (partitioned dataset) using octetTotalCount feature.

Here, we can see that partitioning the dataset does not necessarily compromise overall performance. This is desirable due to two reasons: firstly, fewer data points per node means faster classification times, fewer transactions, and less network overhead for the same performance. Of course, this may also be a consequence of the features being highly correlated with the label -

more work would need to be done to verify the relation between partitioning the dataset and the characteristics of the dataset and its features.

It is also possible to verify that this test scenario yielded better results for the *octetTotalCount* feature, as opposed to what happened in the *Replicated dataset* section. This might be an indication that the nature of the features (binary or real) might have less to do with overall performance than the maturation process of the DCs. In the partitioned scenario, each DCA instance has access to a partitioned pool of data with different distributions. This can mean that the distribution of types has an impact on performance, and extra effort might be needed to expose each node in the network to a pool that has a sufficient distribution of types to guarantee better performance.

5.4.3 Local DCA and other works

The next plot shows the accuracy and f1-score observed when running the DCA on a single machine. In this context, it only makes sense to compare against the partitioned dataset since, in the replicated scenario, each peer is already running the whole dataset locally.

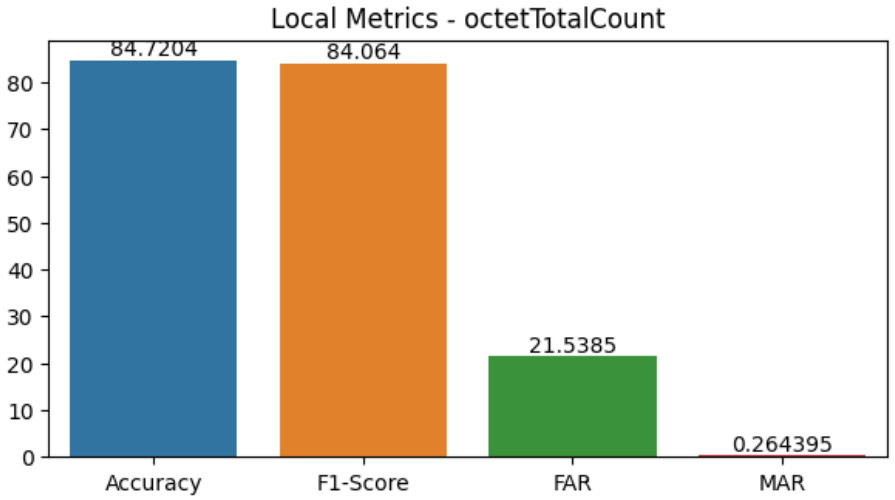

Figure 5.11: Local metrics - *octetTotalCount*.

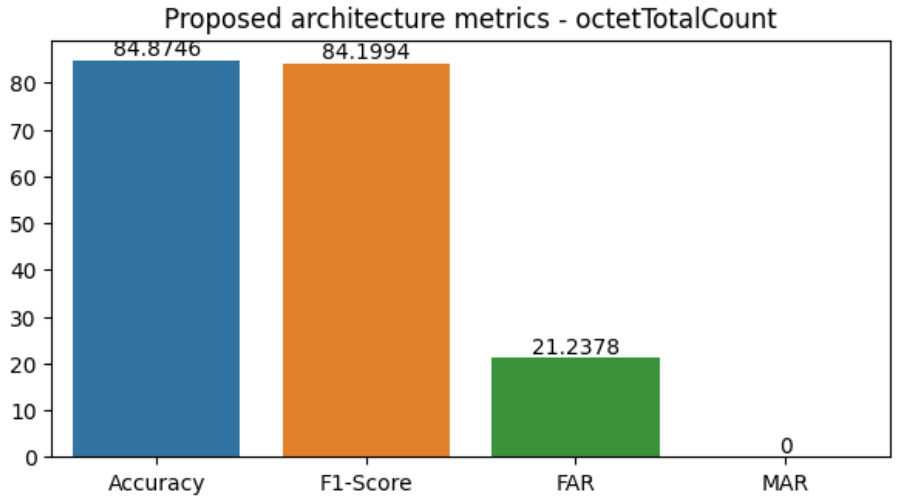

Figure 5.12: Proposed architecture metrics - *octetTotalCount*.

By comparing 5.12 and 5.11, we can see that generally, the performance of the votes is very similar to running the classification locally, with slightly better metrics. Even though even if the overall performance is similar, the proposed solution still allows classifications without a central entity, and the voting system can also compensate for poor individual node performances, as shown in previous tests. Here, we can also verify that the main downside of usual DCA implementations had not been alleviated: a relatively high false alarm rate. Overall, it will depend on the specific use case. If it can cope with false alarms relatively well, this would not be a problem.

For a final comparison, [41] observed Area Under Curve (ROC-AUC) values between 64% and 99%. The proposed solution yielded results between 82% and 94%. Not as high results, but with a higher lower bar. As shown in the previous results, the higher lower bar can be attributed to the voting system, which can compensate for individual lower performances.

5.5 SKAB dataset

The SKAB dataset [45] is a dataset used for bench-marking anomaly detection. It contains 11 features. This dataset was particularly hard to handle with our specific DCA implementation: the version we are using is not very suited to classify real values (which compose all the features of this dataset). So, instead of implementing another version of the DCA and redoing all of the tests with the previous dataset or at least comparing its performances, we decided to add a new feature to this dataset for the purposes of facilitating the testing process: we cluster the data points with the *K-means algorithm* and add a binary feature that is true if the data point is in one of the clusters with the highest anomaly to normal rates, and 0 otherwise. Also, for this dataset, we are only going to run tests with the replicated dataset. The reason for that is the number of data points. While the OPC UA dataset contains over one hundred thousand data points, the section of SKAB we are using contains only around thirty thousand points. This means that if we were to run the replicated dataset, that would not be enough data on each node to get reasonable results, and we

would see what was described in the previous section: a poor voting result due to overwhelming poor individual performances.

5.5.1 Test I - Replicated Dataset

5.5.1.1 Performance metrics

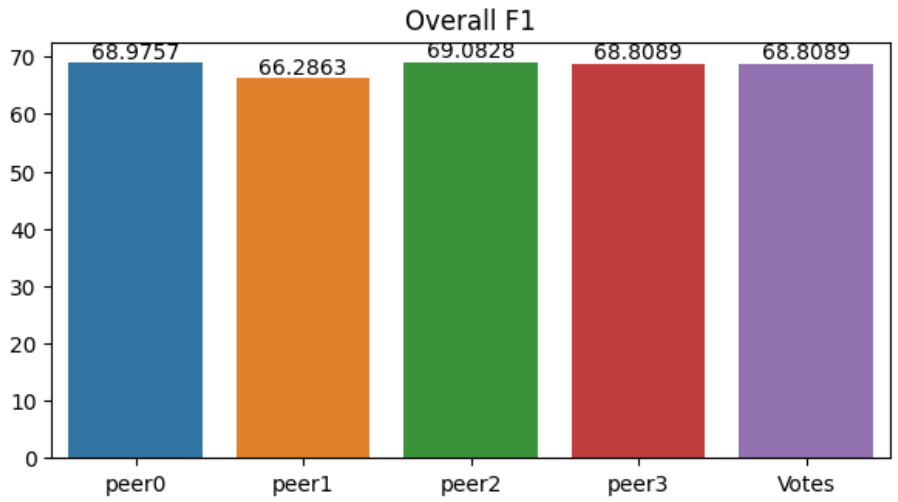

Figure 5.13: Overall F1 - SKAB.

5.5.2 Comparison with other IDS

Table 5.1: Comparison with SKBA results obtained by [41] and other anomaly detection algorithms.

	F1	**FAR[%]**	**MAR[%]**
Proposed solution - Replicated	0.69	22.4	9.15
CDCA_100_10_20_original	0.71	28.27	15.61
CDCA_100_10_5	0.72	29.18	15.24
CDCA_100_10_20	0.72	37.95	5.71
CDCA_100_5_20	0.72	38.27	5.75
LSTM-AE	0.68	14.24	35.56
T-squared+Q (PCA)	0.67	13.95	41.16
MSCRED	0.64	13.56	41.16

Table 5.1 shows a direct comparison between the performance of the proposed solution and the performance of the Cursory Dendritic Cell Algorithm (CDCA) captured for the SKAB dataset by [41]. It also contains other IDS displayed in the scoreboard available on the Kaggle page [45]. From the results, comparing directly with the CDCA, we see a slightly lower F1, with a relatively lower False Alarm Rate (FAR) in relation to all scenarios, along with a lower Missed Alarm Rate (MAR) in two of the four scenarios.

5.6 Result analysis

The proposed solution proved to be similar to other versions of the DCA, with slight differences in the overall classification performance. We observed a higher low bound in classification performance in the OPC UA dataset when compared to the same metric (ROC-AUC) observed by [41]. This could be attributed to the capacity of the voting system to compensate for individual poor performances. We also observed lower FAR rates when compared to the CDCA scenarios tested by [41] and lower MAR rates than two of the four scenarios. Also, compared to other IDS, although we observed a higher FAR rate, we observed a significantly lower MAR rate. These results can be explained by the fact that the majority of the nodes have to either classify an anomalous point as normal or classify a normal point as anomalous.

When comparing to the local DCA (same version), with the proposed solution partitioning the dataset across nodes, we also observed a similar performance, meaning that at least in some scenarios with some datasets, individual node's dendritic cells can indeed go through the maturation process with only a sub-pool of the total dataset.

We were able to observe that, from the point of view of measuring the performance of the classification, the proposed architecture is able to compensate for individual poor performances while also being capable of penalizing the best performers. However, from the point of view of the network, this means that even if there are nodes that are behaving in an anomalous manner, if the majority of nodes have a coherent baseline of normal behavior, the anomalous behavior can be detected.

Furthermore, the voting system can be thought of as an ensemble technique, given that there is some randomness in the utilized DCA version. [49] showed that an ensemble model can reduce overall testing error with the trade-off of possibly increasing individual error, which is shown in some of the tests presented in this chapter. This also explains the cases where the overall performance was equal to the highest-performing node, even though the other 3 nodes performed worse.

After analyzing the observed results, we can qualitatively say that the DCA can indeed be used in the proposed architecture as an IDS in a blockchain environment. The main advantage presented by this work is being able to lift the restriction of running the classification process in a single centralized machine. The proposed solution enables the use of the IDS in a blockchain, which, given the provided contextualization, might be useful for IIoT environments that are increasingly relying on these systems. The main limitations are: the need for expert knowledge for

the DCA still exists here, as demonstrated in the DCA research referenced in this document, and it can prove a substantial effort depending on the use case. Also, there is no support for real-time classification. However, both of these problems relate to the DCA, and as it is going to be pointed out in Section 6.2, adapting the architecture for other DCA versions might be a way to overcome these limitations.

Chapter 6

Conclusions & Future Work

6.1 Conclusions

A new solution for using the DCA in blockchain environments has been developed. This architecture utilizes an implementation of a more traditional DCA version, combined with blockchain events, to communicate individual classifications and achieve a distributed IDS. The architecture utilizes a *majority voting system* to get a consensus on the final classification of individual data points. The architecture could also be extended to other voting systems and DCA versions.

The proposed solution was tested with two datasets simulating IIoT scenarios. The tests with the first dataset, OPC UA, were compared against running the DCA locally and with the CDCA by [41]. The tests revealed that the proposed solution could compensate for individual poor performances, although it did not get as high classification results as the CDCA. At the same time, we obtained a significantly higher lower bound, which can demonstrate that the performance can be further improved by fine-tuning algorithm parameters and/or improving the partitioning process, such as the partitioning process used by [10]. The second dataset yielded competitive results also when compared to the CDCA. However, if not consistently better performance metrics, the proposed solution achieved the capability of not relying on a central machine to perform classification. This means that once the data points are sent to the network, failing or disconnected peers could also have their data points classified. However, limitations of the proposed solution were also observed: the usual high FAR typical of anomaly-based IDS is still present, even if we observed a lower FAR in the case of the SKAB dataset when compared with the CDCA, the rates are still considerably high when compared with the non-DCA based models made available in the SKAB Kaggle page.

The main drawbacks of the proposed solution are: it does not offer online or near-real-time classification capabilities, like the CDCA, and the proposed solution relies on the individual nodes to have a DCA that has access to an accurate normal baseline, meaning that there is still the need of expert knowledge.

Based on the state-of-the-art analysis and the work presented in this document, there is also the possibility that the need for the dendritic cell population to mature again once its lifetime is over

might be a considerable limitation to the integration of the DCA with blockchain environments since the local DCA in each node will end up needing multiple maturation processes over the existence of the network. Further research needs to be conducted in order to assert the impact of this in the DCA-blockchain integration. Due to the lack of sufficiently related work, the testing methodology had a narrow scope. And as it is going to be described in Section 6.2, improvements can be made to the methodology to understand better the qualities and limitations of DCA-based IDS and blockchain integration.

Overall, the proposed solution achieved the goal of providing a distributed DCA-based IDS for a blockchain environment. The quantitative results, however, do not provide sufficient evidence for the suitability or lack of suitability of the proposed solution on their own, based on the comparisons made with other local DCA-based approaches tested on the selected datasets. However, considering that there are possibilities of further studying this architecture with other DCA versions and that the solution already provides a distributed DCA-based IDS, the proposed solution can be assessed as a generally positive step towards integrating the DCA in blockchain environments.

Furthermore, having IDS suited for blockchain environments could have a positive impact on Industry 4.0, which is relying more than ever before on blockchain technologies for its guarantees of confidentiality, integrity, and availability, by helping them to mitigate attacks on these systems without having to rely on centralized systems for this part of its security.

6.2 Future Work

More tests with the architecture could be performed with different versions of the DCA. As mentioned in the document, there have been numerous versions of the DCA developed to overcome the limitations of the original algorithm and improve performance. As such, analyzing how the architecture could perform with other versions of the DCA, such as the cursory DCA, with online classification capabilities, can provide more evidence of the advantages and disadvantages of using the DCA in a blockchain environment. Another possible exciting integration would be the distributed DCA by [10]; with its reliance on distributed data storage, it might be a natural candidate to be tested with the proposed architecture.

The architecture could also be extended to leverage more properties of the DCA algorithm: One possibility that was even envisioned in the early stages of this work is exchanging dendritic cells, using an external node's dendritic cells in the classification process could prove useful to amplify the incorporation of normal baseline across nodes. The architecture could be extended to, for example, expose the dendritic cells to data points of all other nodes for the classification of the data points of a particular peer, as opposed to what happens in the current version, that is, using that node's data points in the maturation process. The idea is that the architecture could be extended to distribute more aspects of the algorithm.

Another particularly interesting idea might be extending the architecture and classifying the nodes themselves. Although the architecture can be used to detect anomaly behavior coming

from devices in the network in a distributed manner, it classifies individual data points. Thus, to improve the overall capabilities of the architecture, it might be extended to also classify the node as "Normal" or "Anomaly," depending on the collective behavior of its data points and other conditions. This would work in a kind of multi-level classification.

Due to the lack of standard datasets used to test IDS in blockchain environments specifically, proper comparisons with other approaches to the same problem are still not possible. Also, testing the architecture in a real environment could provide more reliable data on the proposed solution's true capacities. As mentioned before, a test in a CPS testbed was envisioned, but due to time constraints, we could not execute it.

Also, as mentioned in Section 6.1, more research is needed to verify the impact that multiple maturation processes have on the overall classification capabilities of DCA-based IDS in blockchain environments. The methodology could also be extended to perform isolated tests on the voting system in order to provide more evidence to answer the second research question. However, what would be an appropriate manner to approach these tests is not yet clear.

Appendix A

Relevant code

A.1 DCA reference implementation

Listing A.1: Original DCA Reference implementation [8].

```
from random import randrange
from random import random


def rand_in_bounds(min, max):
    aux = min + ((max - min) * random())
    return aux


def construct_pattern(class_label, domain, p_safe, p_danger):
    set = domain[class_label]
    selection = randrange(len(set))

    pattern = {}
    pattern['class_label'] = class_label
    pattern['input'] = set[selection]
    pattern['safe'] = (random() * p_safe * 100)
    pattern['danger'] = (random() * p_danger * 100)

    return pattern


def generate_pattern(domain, p_anomaly, p_normal, prob_create_anom=0.5):
    pattern = {}
    if random() < prob_create_anom:
        pattern = construct_pattern("Anomaly", domain, 1.0 - p_normal, p_anomaly)
        print(">Generated Anomaly {}".format(pattern['input']))
    else:
```

```
29          pattern = construct_pattern("Normal", domain, p_normal, 1.0 - p_anomaly)
30
31      return pattern
32
33
34  def initialize_cell(thresh):
35      cell = {}
36      cell['lifespan'] = 1000.0
37      cell['k'] = 0.0
38      cell['cms'] = 0.0
39      cell['migration_threshold'] = rand_in_bounds(thresh[0], thresh[1])
40      cell['antigen'] = {}
41      return cell
42
43
44  def store_antigen(cell, input):
45      if cell['antigen'].get(input) is None:
46          cell['antigen'][input] = 1
47      else:
48          cell['antigen'][input] += 1
49
50
51  def expose_cell(cell, cms, k, pattern, threshold):
52      cell['cms'] += cms
53      cell['k'] += k
54      cell['lifespan'] -= cms
55      store_antigen(cell, pattern['input'])
56      if cell['lifespan'] <= 0:
57          initialize_cell(threshold)
58
59
60  def can_cell_migrate(cell):
61      if cell['cms'] >= cell['migration_threshold']
62      and len(cell['antigen']) > 0:
63          return True
64      else:
65          return False
66
67
68  def expose_all_cells(cells, pattern, threshold):
69      migrate = list()
70      cms = pattern['safe'] + pattern['danger']
71      k = pattern['danger'] - (pattern['safe'] * 2.0)
72      for cell in cells:
73          expose_cell(cell, cms, k, pattern, threshold)
```

```
74              if can_cell_migrate(cell):
75                  migrate.append(cell)
76                  if cell['k'] > 0:
77                      cell['class_label'] = "Anomaly"
78                  else:
79                      cell['class_label'] = "Normal"
80      return migrate
81
82
83  def train_system(domain, max_iter, num_cells, p_anomaly, p_normal, thresh):
84      imature_cells = list() # list of imature cells
85      migrated = list() # List of migrated cells
86      migrants = list() # List of possible cells to be migrated
87
88      for c in range(0, num_cells):
89          input_cell = initialize_cell(thresh)
90          imature_cells.append(input_cell)
91
92      for iter in range(0, max_iter):
93          pattern = generate_pattern(domain, p_anomaly, p_normal)
94          migrants = expose_all_cells(imature_cells, pattern, thresh)
95          for cell in migrants:
96              imature_cells.remove(cell)
97              imature_cells.append(initialize_cell(thresh))
98              migrated.append(cell)
99
100     return migrated
101
102
103 def classify_pattern(migrated, pattern):
104     input = pattern['input']
105     num_cells = 0
106     num_antigen = 0
107     for cell in migrated:
108         if cell['class_label'] == "Anomaly"
109         and cell['antigen'].get(input) is not None:
110             num_cells += 1
111             num_antigen += cell['antigen'][input]
112
113     if num_antigen > 0:
114         mcav = float(num_cells) / float(num_antigen)
115     else:
116         mcav = 0
117
118     if mcav > 0.5:
```

```
119             return "Anomaly"
120         else:
121             return "Normal"
122
123
124 def test_system(migrated, domain, p_anomaly, p_normal, num_trial=100):
125         correct_norm = 0
126         for i in range(0, num_trial):
127             pattern = construct_pattern("Normal", domain, p_normal, 1.0 - p_anomaly)
128             class_label = classify_pattern(migrated, pattern)
129             if class_label == "Normal":
130                 correct_norm += 1
131         print("Finished testing Normal inputs {}/{}".format(correct_norm, num_trial))
132
133         correct_anom = 0
134         for j in range(0, num_trial):
135             pattern = construct_pattern("Anomaly", domain, 1.0 - p_normal, p_anomaly)
136             class_label = classify_pattern(migrated, pattern)
137             if class_label == "Anomaly":
138                 correct_anom += 1
139         print("Finished testing Anomaly inputs {}/{}".format(correct_anom, num_trial))
140
141         return [correct_norm, correct_anom]
142
143
144 def execute(domain, max_iter, num_cells, p_anom, p_norm, thresh):
145         migrated = list()
146         migrated = train_system(domain, max_iter, num_cells, p_anom, p_norm, thresh)
147         test_system(migrated, domain, p_anom, p_norm)
148         return migrated
```

A.2 Blockchain interface code

Listing A.2: Blockchain interface.

```
1  public async classificationHandler(event:ContractEvent) {
2          if(event.eventName != "classificationEvent"){
3              return;
4          }
5          try{
6              let classification = JSON.parse(event.payload!.toString("utf-8"),reviver);
7              if(classification["Origin"] == this.peerId ||
8              classification["RunSequence"] != this.runSequence)
9                  return;
10             this.electionExecuter.addVote(
```

```
                    classification["For"],
                    classification["Origin"],
                    classification["labels"]
                );
            } catch(e) {
                console.error("Handling error!");
                console.error(e)
            }
    }

    public async putAsset(key, value){
        let result = await this.chaincodeHandler.contract!.submitTransaction("put",key,value);
        return result
    }

    public async getAsset(key){
        let result = await this.chaincodeHandler.contract!.submitTransaction("get",key);
        return result
    }

    public async getAllAssets(){
        let result = await this.chaincodeHandler.contract!.submitTransaction("list",);
        return result
    }

    public async getAllAssetsWithPagination(bookmark){
        let result = await this.chaincodeHandler.contract!.submitTransaction(
            "listWithPagination",
            bookmark
        );
        return result
    }

    public async deleteAllAssets(){
        let result = await
            this.chaincodeHandler.contract!.submitTransaction("deleteAll","dummy");
        return result
    }

    public async deleteAsset(key){
        let result = await this.chaincodeHandler.contract!.submitTransaction("delete", key);
        return result
    }

    public async classify(key,value){
```

```
55          let result = await this.chaincodeHandler.contract!.submitTransaction("classify",key,value);
56          return result
57      }
58
59      public async putMultiple(value){
60          let result = await this.chaincodeHandler.contract!.submitTransaction("putMultiple",value);
61          return result;
62      }
```

References

[1] Editorial introduction. *Seminars in Immunology*, 12(3):159–162, 2000.

[2] S. Adarsh, V. S. Anoop, and S. Asharaf. Distributed consensus mechanism with novelty classification using proof of immune algorithm. In Deepak Gupta, Ashish Khanna, Siddhartha Bhattacharyya, Aboul Ella Hassanien, Sameer Anand, and Ajay Jaiswal, editors, *International Conference on Innovative Computing and Communications*, pages 173–183, Singapore, 2023. Springer Nature Singapore.

[3] U. Aickelin, P. Bentley, S. Cayzer, J. Kim, and J. McLeod. Danger theory: The link between ais and ids? volume 2787, pages 147–155. Springer Nature, 2008.

[4] Uwe Aickelin and Steve Cayzer. The danger theory and its application to artificial immune systems. 2008.

[5] Mark C. Ballandies, Marcus M. Dapp, and Evangelos Pournaras. Decrypting distributed ledger design-taxonomy, classification and blockchain community evaluation. *Cluster computing*, 25(3):1817, 2022;2020;2018;.

[6] Aniruddha Bhattacharjya. A holistic study on the use of blockchain technology in cps and iot architectures maintaining the cia triad in data communication. *International journal of applied mathematics and computer science*, 32(3):403–413, 2022.

[7] Dhruba K Bhattacharyya and Jugal Kalita. *Network Anomaly Detection: A Machine Learning Perspective*. 04 2013.

[8] Jason Brownlee. *Clever algorithms: nature-inspired programming recipes*. Jason Brownlee, 2011.

[9] Steve Cazyer and Uwe Aickelin. A recommender system based on the immune network. 2008.

[10] Zaineb Chelly Dagdia. A scalable and distributed dendritic cell algorithm for big data classification. *Swarm and Evolutionary Computation*, 50:100432, 2019.

[11] Hong Chen, Kecheng Su, and Wandong Gao. The analysis of blockchain digital currency product innovation based on artificial immune algorithm. *IEEE access*, 10:132448–132454, 2022.

[12] Giovanna Culot, Fabio Fattori, Matteo Podrecca, and Marco Sartor. Addressing industry 4.0 cybersecurity challenges. *IEEE Engineering Management Review*, 47(3):79–86, 2019.

[13] Leandro De Castro and Jon Timmis. *Artificial immune systems: A new computational intelligence approach*. 06 2002.

[14] Hervé Debar, Marc Dacier, and Andreas Wespi. Towards a taxonomy of intrusion-detection systems. *Computer Networks*, 31(8):805–822, 1999.

[15] F. Esponda, S. Forrest, and P. Helman. A formal framework for positive and negative detection schemes. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(1):357–373, 2004.

[16] Mohamed A. Ferrag and Leandros Maglaras. Deliverycoin: An ids and blockchain-based delivery framework for drone-delivered services. *Computers (Basel)*, 8(3):58, 2019.

[17] Julie Greensmith and Uwe Aickelin. The deterministic dendritic cell algorithm. In *International conference on artificial immune systems*, pages 291–302. Springer, 2008.

[18] Julie Greensmith, Uwe Aickelin, and Steve Cayzer. *Introducing Dendritic Cells as a Novel Immune-Inspired Algorithm for Anomaly Detection*, volume 3627, pages 153–167. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[19] Julie Greensmith, Uwe Aickelin, and Jamie Twycross. *Articulation and Clarification of the Dendritic Cell Algorithm*, volume 4163, pages 404–417. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[20] Stuart Haber and W Scott Stornetta. How to time-stamp a digital document. *Journal of Cryptology*, 3(2):99–111, January 1991.

[21] Steven A. Hofmeyr and Stephanie Forrest. Architecture for an artificial immune system. *Evolutionary computation*, 8(4):443–473, 2000.

[22] Hyperledger fabric documentation, 2023. Accessed on 2023-05-09.

[23] Ibm developer documentation on hyperledger fabric, 2023. Accessed on 2023-05-09.

[24] Hyperledger fabric whitepaper, 2023. Accessed on 2023-05-09.

[25] IBM. Blockchain for supply chain - ibm blockchain (accessed on 2023/01/20), 2023.

[26] IBM. Blockchain security - ibm blockchain (accessed on 2023/06/19), 2023.

[27] Aamir Iqbal, Mohammad Amir, Vinod Kumar, Aftab Alam, and Mohammad Umair. Integration of next generation iiot with blockchain for the development of smart industries. *Emerging science journal*, 4:1–17, 2020.

[28] Markus Jakobsson and Ari Juels. *Proofs of Work and Bread Pudding Protocols(Extended Abstract)*, pages 258–272. Springer US, Boston, MA, 1999.

[29] Danial Ritzuan Junaidi, Maode Ma, and Rong Su. Secure vehicular platoon management against sybil attacks. *Sensors*, 22(22), 2022.

[30] Jungwon Kim and Peter J. Bentley. An evaluation of negative selection in an artificial immune system for network intrusion detection. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, GECCO'01, page 1330–1337, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[31] Ralph Langner. Stuxnet: Dissecting a cyberwarfare weapon, 2011.

[32] Hongwei Li, Rongxing Lu, Liang Zhou, Bo Yang, and Xuemin Shen. An efficient merkle-tree-based authentication scheme for smart grid. *IEEE Systems Journal*, 8(2):655–663, 2014.

[33] Lodovica Marchesi, Michele Marchesi, Roberto Tonelli, and Maria I. Lunesu. A blockchain architecture for industrial applications. *Blockchain: Research and Applications*, 3(4):100088, 2022.

[34] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. pages 53–65. Springer, 2002.

[35] Weizhi Meng, Elmar Wolfgang Tischhauser, Qingju Wang, Yu Wang, and Jinguang Han. When intrusion detection meets blockchain technology: A review. *IEEE Access*, 6:10179–10188, 2018.

[36] Aleksandar Milenkoski, Marco Vieira, Samuel Kounev, Alberto Avritzer, and Bryan D. Payne. Evaluating computer intrusion detection systems: A survey of common practices. *ACM computing surveys*, 48(1):1–41, 2015.

[37] Tim R. Mosmann and Alexandra M. Livingstone. Dendritic cells: the immune information management experts. *Nature immunology*, 5(6):564–566, 2004.

[38] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[39] L. Nunes de Casto and F. J. Von Zuben. An evolutionary immune network for data clustering. pages 84–89. IEEE, 2000.

[40] Chung-Ming Ou, C. R. Ou, and Yao-Tien Wang. *Agent-Based Artificial Immune Systems (ABAIS) for Intrusion Detections: Inspiration from Danger Theory*, pages 67–94. Agent and Multi-Agent Systems in Distributed Systems - Digital Economy and E-Commerce. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[41] Carlos Pinto, Rui Pinto, and Gil Gonçalves. Towards bio-inspired anomaly detection using the cursory dendritic cell algorithm. *Algorithms*, 15(1), 2022.

[42] Rui Pinto. M2m using opc ua. 2020.

[43] Rui Pinto and Gil Gonçalves. Application of artificial immune systems in advanced manufacturing. *Array*, 15:100238, 2022.

[44] Galina Samigulina and Zarina Samigulina. Development of a unified artificial immune system for complex objects control within the framework of the industry 4.0 concept. *Procedia computer science*, 219:824–831, 2023.

[45] SKAB. Skab dataset. https://www.kaggle.com/datasets/yuriykatser/skoltech-anomaly-benchmark-skab, 2020. Accessed: 2023-06-13.

[46] Mohammad H. Tabatabaei, Roman Vitenberg, and Narasimha R. Veeraragavan. Understanding blockchain: definitions, architecture, design, and system comparison. 2022.

[47] Youliang Tian, Ta Li, Jinbo Xiong, Md Z. A. Bhuiyan, Jianfeng Ma, and Changgen Peng. A blockchain-based machine learning framework for edge services in iiot. *IEEE transactions on industrial informatics*, 18(3):1918–1929, 2022.

[48] Keqi Wang, Wei Xie, Wencen Wu, Jinxiang Pei, and Qi Zhou. Blockchain-enabled internet-of-things platform for end-to-end industrial hemp supply chain. 2020.

[49] G. I. Webb and Z. Zheng. Multistrategy ensemble learning: reducing error by combining ensemble learning techniques. *IEEE transactions on knowledge and data engineering*, 16(8):980–991, 2004.

[50] Dominik Widhalm, Karl Goeschka, and Wolfgang Kastner. A review on immune-inspired node fault detection in wireless sensor networks with a focus on the danger theory. *Sensors*, 23, 01 2023.

[51] Jacob Wurm, Khoa Hoang, Orlando Arias, Ahmad-Reza Sadeghi, and Yier Jin. Security analysis on consumer and industrial iot devices. In *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 519–524, 2016.

[52] Shiyong Yin, Jinsong Bao, Yiming Zhang, and Xiaodi Huang. M2m security technology of cps based on blockchains. *Symmetry (Basel)*, 9(9):193, 2017.

[53] Liangpei Zhang, Yanfei Zhong, Bo Huang, Jianya Gong, and Pingxiang Li. Dimensionality reduction based on clonal selection for hyperspectral imagery. *IEEE transactions on geoscience and remote sensing*, 45(12):4172–4186, 2007.

[54] Qingyi Zhu, Seng W. Loke, Rolando Trujillo-Rasua, Frank Jiang, and Yong Xiang. Applications of distributed ledger technologies to the internet of things: A survey. *ACM Comput. Surv.*, 52(6), nov 2019.