Technische Universität München

Department of Informatics

Bachelor's Thesis in Informatics

Machine Learning-Based Adaptive Anomaly Detection in Smart Spaces

Franǫis-Xavier Aubet, B. Sc.

Technische Universität München

Department of Informatics

Bachelor's Thesis in Informatics

Machine Learning-Based Adaptive Anomaly Detection in Smart Spaces

Machine Learning Basierte Adaptive Anomlie Erkennung in Smart Space

Author	Franҫis-Xavier Aubet, B. Sc.
Supervisor	Prof. Dr.-Ing. Georg Carle
Advisor	Dr. Marc-Oliver Pahl, Stefan Liebald,M. Sc.
Date	April 9, 2018

Informatik VIII
Chair for Network Architectures and Services

I confirm that this thesis is my own work and I have documented all sources and material used.

Garching b. München, April 9, 2018

Signature

Abstract

The increase in computing power allowed the rise of the Internet of Things (IoT). Due to the nature of IoT-service it is vital to secure them, as they can impact the privacy and the safety of the user. In this work, we propose an approach to secure sites of cooperating IoT-services, known as smart space. Our solution monitors continuously service interactions, and learns online the normal behavior of services. Using the learnt model of the normal behavior it can detect anomalous accesses and block them in real-time. We use online machine learning to handle concept drift. However, the learned model is human understandable to allow the user to understand the behavior of the services. We show the performance of this approach, by evaluating it in regard to the resource usage and the detection performance.

Zusammenfassung

Steigende Rechenkapazitäten ermöglichten das Entstehen des Internet of Things (IoT). Die Grundzüge von IoT-Diensten machen eine Absicherung dieser unumgänglich, da sie eine große Auswirkungen auf die Privatsphäre und die Sicherheit der Endbenutzer haben können. In dieser Arbeit wird ein Ansatz zur Absicherung von IoT-Netzwerken präsentiert. Die vorgestellte Lösung überwacht den Datenaustausch zwischen Diensten kontinuierlich und lernt dabei deren reguläres Verhalten. Unter Einsatz des erlernten Verhaltensmodells lassen sich ungewöhnliche Aktivitäten entdecken und in Echtzeit blockieren. Die Benutzung von Online-Machine-Learning erlaubt dabei den Umgang mit Concept Drift. Das erlernte Modell ist dabei verständlich für Menschen und erlaubt dem Endbenutzer, das Verhalten der Dienste zu verstehen. Abschließend wird dieser Ansatz durch Evaluierung der Ressourcennutzung und Erkennungsleistung bewertet.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

In the past years the computing power of small chips has increased many folds. This allowed the rise of the Internet of Things (IoT). IoT-devices are a new kind of devices able to connect to another and to run programs. These devices can cooperate to offer a wider range of features.

A smart-space is a place like a house or an airport where all or most of the electrical devices are IoT-devices. They are linked via a common network through which they can communicate with each other. These devices can be traditional house devices, like a light or a thermostat, but also sensors to understand the environment or smart-door locks for example. We could imagine a house where the central heating and the opening of the windows are controlled by small computers that read the temperature from sensors placed in each room. All the different small computers controlling the windows, the central heating or measuring the temperature need to communicate with each other to take decisions. The programs running on these small computer are called services.

Securing the communications between the IoT devices is essential. These devices measure and gather a lot of information about the user life, thereby entering his privacy. Moreover, the smart-devices are able to act on their environment, and for example an attacker could unlock the door of peoples' house. Which is a danger for the user safety. Therefore, it is important to secure the access to IoT-devices for privacy as well as for safety reasons.

The existing mechanisms used to provide access security are based on policies that regulate which device is allowed to communicate with which other one. [10] However these access policies are created by hand. This requires a lot of work from the system administrator, and forbids to adapt the policies automatically. [11]

We propose an approach to detect anomalous communications between the different

services of smart-spaces in an automated way. We could imagine a scenario where an attacker takes control of the controller of the kitchen window and asks for the temperature of all the rooms in the house even if the user does not want to share this information. The module detects these connections and blocks them. The goal of this module is to add a security layer that does not exist yet. This layer could detect intrusions in the system or the failure of IoT-devices.

In this work we consider the following research question:

How can anomalous connections between the services of a smart-space be detected the most accurately in the shortest time?

1.2 Methodology

At first we have a short look at the methodology followed throughout this work. In this work we solve our problem using Machine Learning, to do so we follow a methodology inspired by the CRISP-DM model. The CRISP-DM model was developed to provide a process model for conducting a data mining project. [12] The project is divided in six phases: business understanding, data understanding, data preparation, modeling, evaluation, and deployment. [12] The connections between this phases are shown in figure 1.1. As we present in the outline, the different parts of this work will follow this methodology.

- business understanding: The goal is to understand the application domain, and to use this knowledge to create a data mining problem definition and identify requirements.
- data understanding: The initial data is collected and examined. Moreover the data is described in order to be sure that it satisfies the requirements.
- data preparation: During this phase the collected data is used to create the dataset that will be used to create the model.
- modeling: In this phase the modeling technique is selected, a model is created and it is assessed.
- evaluation: The model is evaluated according to the requirements defined in the business understanding phase. If the model does not fit the requirements the process should begin again at the business understanding.
- deployment: Once the model is created it has to be deployed in a way that it can be used.

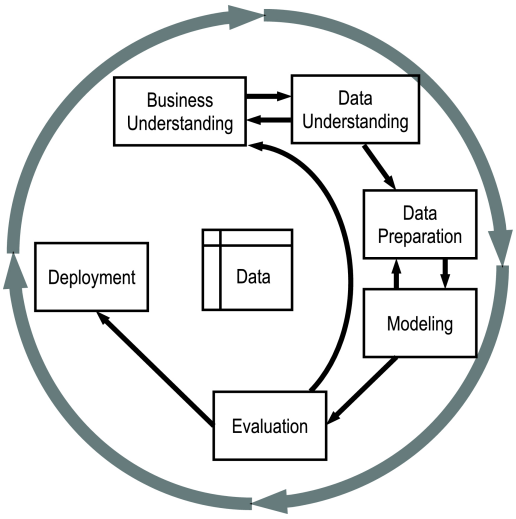


Figure 1.1: Phases of the CRISP-DM model. (reused with permission from [13])

1.3 Outline

This thesis is structured as follows. In Chapter 2 we will go through the business understanding phase and present the fields that we will use in this work. In Chapter 3 we will review similar work from different application domains. In Chapter 4 we will first go through the data understanding phase, then we will have a design loop to test different models. This loop will contain the data preparation, modeling and evaluation phases. In Chapter 5 we will see some points of the deployments. Then in Chapter 6 we will evaluate the final solution created. Finally in Chapter 7 will present some concluding remarks and possible future work.

Chapter 2

Analysis

In this chapter we introduce our application domain, present our problem and explain the different fields that are used to solve this problem. First we present the smart-space architecture that we are using and the access control mechanism that it uses in section 2.1. This leads to the description of the problem we tackle in this work in section 2.2. Following the description of the problem, we present the different fields and concepts that are used to in this work: Intrusion detection systems in section 2.3, anomaly detection in section 2.4, machine learning in section 2.5, clustering in section 2.6 and finally considerations about the data needed in section 2.7.

In the last sections we analyse the particularity of IoT traffic in regard to anomaly detection in section 2.8. Then we introduce network flows 2.9 and graph representations 2.10.

At the end we sum up the requirements and introduce our research question.

2.1 The Smart-space architecture

We detect anomalies in the connections between the services of a smart-space system. The smart-space architecture that we enhance with an anomaly detection module is the Distributed Smart Space Orchestration (DS2OS) that has been developed by Marc-Oliver Pahl since 2008. First we introduce the architecture of the system, then we have a closer look at the existing system to regulate connections. This helps us to identify the challenges that we face during the design. Finally we introduce an example scenario that helps us illustrate explanations later during the analysis.

2.1.1 Architecture

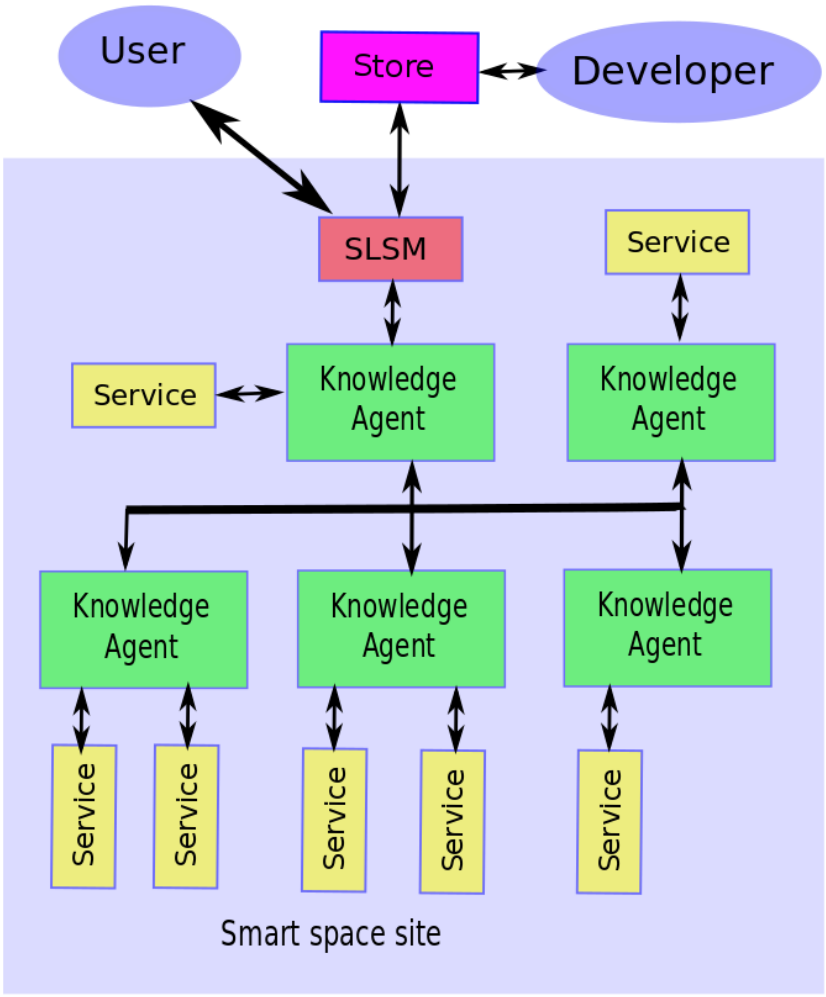


Figure 2.1: The structure of the DS2OS architecture.

At first, we look at the global distributed organization of this system to have a broader view than just the area in which we are developing our system. The global organization of the DS2OS can be understood via a comparison with mobile-phones. Each site, place where a smart-space is deployed, has a set of services comparable to applications on a

mobile-phone. Depending on the needs of the user, different services can be deployed. The deployment process of services in our smart space system is comparable to the one of mobile applications. Developers can create services and then publish them on a trusted store, the S2Store, which then distributes the services to the users who want to have them. [14]

Our application domain is on the site level, therefore we present the architecture of the DS2OS within a smart-space. The distributed computing nodes are connected via a middelware, the Virtual State Layer (VSL). The expectation is that most of the devices within a site are equipped with computing units capable of hosting services. Thereby, each of these equipped devices is a node in the network. The VSL is a self-organizing peer-to-peer system that acts as edge-based distributed operation system. Edge-base is to be opposed to cloud based, it means that the computing happens on the nodes and not on remote servers. The VSL has the particularity that the service logic and the service state are separated, the VSL manages the service state and the services run the logic. [15]

Figure 2.1 shows the structure of the DS2OS. There are two types of entities: humans, represented by circles, and programs, represented by rectangles. The entities inside other entities are part of them. The dark arrows represent the connections between the entities. In the following subsections we present different components of the VSL that we need to consider for this work.

2.1.1.1 Site Local Service Manager

The Site Local Service Manager (SLSM) is a service that manages the other services in the Smart Space Site. It is the component that communicates with the outside of the Smart Space. The User controls the smart space via the SLSM using a smart phone app or a web-interface. In addition the SLSM is connected to the S2Store where the developers publish services. If the user commands it, the SLSM can download services from the store. The user can configure new services via the SLSM. [15]

2.1.1.2 Knowledge Agent

The Knowledge Agent (KA) is responsible for the state of the services running on the same SHE. The state is represented in the form of Context Nodes. In this way, each KA has the values of a subset of all the Context Nodes of the Smart Space Site. However each KA is aware of all the Context Nodes of the Site, the values of these nodes is obtained by accessing the Context Nodes of the other KAs. Each Context Node has an unique address. Listing 2.1 shows an example address.

```
/kaName/serviceName/variableName
```

Listing 2.1: An example Contect Node address

Figure 2.2 represents the organisation of an example KA. The Context Repository is the tree containing all the Context Nodes on the local node. At the bottom of the diagram the types of accesses are shown. A get access returns the value of the Context Node at the given address. A set access sets the value of the addressed Context Node to the value given in the call. [15]

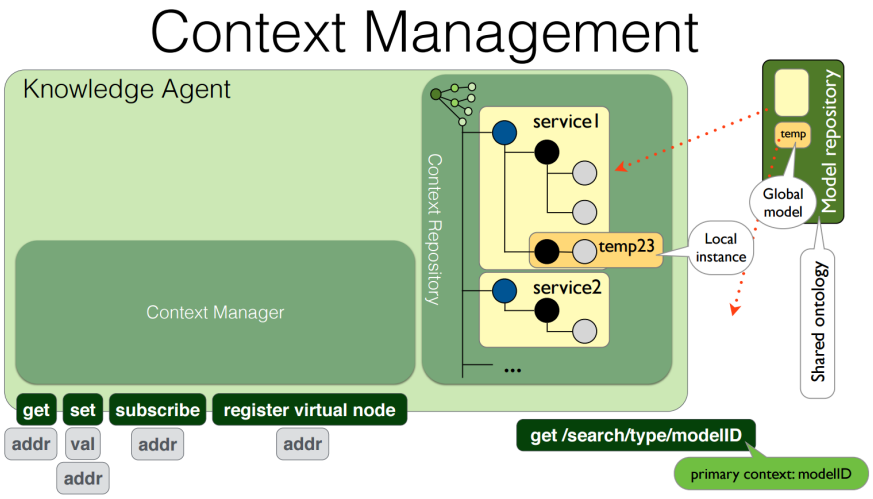


Figure 2.2: The structure of the Knowledge Agent. (Figure taken from [1])

2.1.2 The existing access control

The DS2OS already uses a system of policies to regulate the access to the Context Nodes. We first present it and then analyse its advantages and drawbacks.

The Knowledge Agent (KA) is the part that interest us, as it provides access to all services' data. The VSL information model is composed by a hierarchical structure of Context Nodes containing different attributes. The services have to belong to a certain group to be allowed by the KA to access the information. An example VSL data model can be seen in Listing 2.2. Each attribute of the WindowController service has different access policies. The *state* can be read by all the services and only the WindowController has write access rights. The *lastOpening* can only be accessed by the WindowController. Finally the *openedWindowHouse* can be read by any other service and can be written by all the services of the group *Controller*. [14]

```
<WindowController>
<state type="basic/text" reader="*" writer="" />

<lastOpening type="basic/number" reader="" writer="" />

<openedWindowHouse type="basic/number"  reader="*"
                   writer="Controller" />

</WindowController>
```

Listing 2.2: The example model of a service.

Thanks to this organization all the information exchanged between the services is contained in the VSL KA overlay. Thereby the service cannot go around the access policies within the trusted KA. Thanks to this system, security by design is introduced [16]. Moreover, it is great for us because it allows to monitor all the connections between the different services of the house. [14]

The main disadvantage of this system is that the rules are static and all the connections are treated alike. Services are treated in the same way whether they where connected before or not. The second issue is that there is no global view of the different connections in the network. A service asking for all the information it can access at once would not be detected. This is the reason why we want to enhance this system with an anomaly detection module.

2.1.3 Challenges

Now that we have understood the way the smart-space architecture works we want to identify the challenges that it implies.

As the DS2OS is scalable and allows any number of service to be managed [17], the detection module should also allow this scalability. The first major challenge linked with this architecture is to identify *where the anomaly detection module should be placed.*

The second challenge linked to IoT edge-based systems is that all the computation is made within the smart space [17]. Therefore the anomaly detection algorithm has to make *minimal demands on system resources.*

The last challenge linked with the use of the DS2OS is that no dataset is available and that *we have to create one to test the algorithm.*

2.1.4 Example scenario

Throughout this analysis, we illustrate explanations with an example scenario of smart-space. This is just for the purpose of explanation. We consider an office with 10 rooms. There is a central heating that is controlled by a service S_0. The temperature of each room is measured by a thermometer that publishes its measurements with a service $S_{1,i}$ with i the number of the room. Finally, each room has a window that can be controlled by a service $S_{2,i}$. The normal behavior of the system is so that the central heating service S_0 accesses all the thermometer services at 5 am to choose a heating policy for the day. Moreover, each window controlling service $S_{2,a}$ decides every 10 minutes if the window should be opened or closed. In order to decide, it accesses the thermometer service $S_{1,a}$ of its room and checks the heating scenario of S_0. This example is used to illustrate proprieties in the rest of the analysis.

2.2 Problem domain

In the previous section we present our application domain, the DS2OS. In this section we analyse the problems we tackle in this thesis.

We go through the different requirements. For each of them we discuss why it is needed and name the concrete challenges and goals. We also present the current situation regarding this requirement.

2.2.1 Detect anomalies

The goal of this work is to detect anomalies in the connection between services of the smart-space architecture DS2OS. The detection of anomalies is made by defining a normal behavior. In this first section of requirement we define the characteristics of this normal behavior to find out what is to be considered anomalous.

2.2.1.1 Service access monitoring

To detect anomalies in the connections between services, the normal behavior has to be defined or learned [11]. To do so, our systems needs to monitor the accesses in the smart-space, in order to know which connection exists in the set of all the connections allowed by the existing set of rules.

Using this the system has to be able to detect anomalies, like abrupt changes in the behavior of a service that could indicate that it is malicious [11]. It is also important to monitor the type of access. If a service only reads the information contained in a node and wants to perform a write access, it should also be considered as an anomaly.

This type of anomaly can be illustrated using our example scenario defined in 2.1.4. In the normal behavior the window controller service $S_{2,a}$ of a room a only accesses the thermometer service of the same room $S_{2,a}$. A connection from this window controller service to the temperature measurement from another room should be labeled as anomalous.

This detection feature can be enhanced with other options. We will go through these different options.

2.2.1.2 Periodicity anomalies

An additional detection criteria that could be taken into account is the frequency with which the connection occurs. If in normal behavior a service contacts another every 10 minutes, it should be considered anomalous that the frequency changes to one access every 2 seconds. This type of anomaly detection can be useful to detect malfunction or Denial of Service (DoS) attacks. [18]

We could even imagine that the anomaly detection module could warn if a connection stops. Learning the frequency of a connection could help the module predict packets. If packets are missing, an alarm could be raised.

2.2.1.3 Value anomalies

The second possible enhancement would be to detect anomalies in the values exchanged in the communications [19]. We can illustrate this situation with our example scenario from 2.1.4. For example, the response to a reading of a thermometer is of 20 °C. If then the reading changes abruptly to −30 °C, it should be labeled as an anomaly. This could be due to a sensor malfunctioning or another error. In any case, we want to be informed of this abrupt change.

2.2.1.4 Access patterns

The last improvement feature could be the detection of the access patterns. The anomaly detection module could learn the frequent access patterns, then label as anomalous group of packets deviating from the learned patterns. For example if a service always reads 3 nodes one after the other and then writes in a fourth node, it could be labeled as anomalous if the writing was done after the reading of only one of the three nodes was performed.

2.2.2 Integrated in the DS2OS and scalable

The DS2OS is a scalable smart space architecture [17]. We want the designed detection module to be placed in this architecture and still allow it to be scalable. Therefore, the anomaly detection should be scalable. This means that it should be possible in a smart-home with ten services as well as in a smart-airport with thousands of connected devices.

2.2.3 Real-time

Anomaly detection can be used to detect attacks after they happened or to prevent them from happening [20]. We want the anomaly detection to be possible in real-time. This would allow the anomaly detection to work like a firewall allowing only the packets that are not anomalous. This is only possible, if the detection does not create latencies. If it is nt possible to prevent attacks, we want the detection to run almost as fast as possible. This requirement might result in a trade-off between accuracy and speed of the detection. This issue will be addressed in the choice of the algorithms.

2.2.4 Adapt to changes of the normal behavior

The anomaly detection has to learn the normal behavior and detect anomalies deviating from this learned normal behavior [11]. However, every smart-space has a different normal behavior, a service accessing five hundred other services might be normal in a smart-airport but is not possible in a smart-home that has less services. Therefore, the anomaly detection should learn the normal behavior of the location in which it is deployed to detect anomalies. It is not possible to create a definition of the normal behavior before the deployment because it would not fit all the different smart-spaces. [21]

The second problem is that the normal behavior of one smart-space can change over time [11]. Therefore the definition of the normal behavior used for the anomaly detection has to adapt to these changes. These changes can be due to updates or the addition of new services.

This is a hard requirement to meet. The goal is that the definition of the normal behavior used for the anomaly detection continuously adapts to the changes of the actual normal behavior.

2.2.5 Human understandable model

It is interesting for the system administrator to understand what is going on in the smart space. Therefore we want to create a human understandable description of the observations made. Moreover, we want to allows the understanding of the anomaly detection to keep the system administrator in the loop. Therefore, we want the created model to be human understandable. Some Machine Learning approaches act similarly to black boxes that output if an access is anomalous or not, but do not allow to understand why it was classified this way [21]. Furthermore, we want to provide basic visualizations of the model created to give the system administrator an insight in the connection relationships of the smart space.

2.3 Intrusion Detection Systems

We first introduce Intrusion Detection Systems (IDS). IDS are security tools that are used to detect intrusions either within networks or on a host. Our problem shares similar challenges with them. First, we look at the different types of IDS and then look at some methods used in this domain. After, this we present an application domain of IDS.

2.3.1 Traditional IDS

There are two types of IDS: host-based and network-based IDS. Understanding the differences between these systems helps us in the design to choose where to place the anomaly detection module. A host-based IDS is placed on each host and analyses the connections that the host has with other peers. A network-based IDS monitors the traffic of the whole network and detects anomalies not just for the hosts but also on the network level. This type of IDS can detect network scans or Distributed Denial of Service (DDoS) attacks that are invisible for a host-based system. Because a network scan would be done by one host sending one scan packet to all the other hosts, each host would just discard the packet. But they would need to view the whole network to interpret these packets as a network scan. [22] We want our system to have the advantages of a network-based system.

There are two detection methods used in IDS, on one side, misuse-based detection and on the other side, anomaly detection. In misuse-based detection, a signature is defined for each type of attack, that is to be detected. If the system calls or the network packets monitored match the signature of one of the attack an alarm is rung. In anomaly detection based IDS the normal behavior of the system is defined and actions that do not fit in this defined normal behavior are labeled as anomalous. The advantage AD-based IDS is that they can detect zero-day attacks, which a rule-based IDS would not be able to detect because it has no signature for these attacks. On the other hand,

AD-based IDS have a high false alarm rate, because not every action lying outside of the normal behavior is in reality an attack. [22] The IDS models are very often trained using supervised learning. This is explained in section 2.5.

We present a anomaly-based IDS for Smart Spaces in this thesis. It uses schemes of both network-based and host-based systems.

2.3.2 Industrial Control Systems

Industrial Control Systems (ICS) are systems that use sensors and actors to control processes in different industries, like: chemical processing, power generation, oil and gas processing, water networks and telecommunications. They share the same type of network as IoT network, since in IoT most of the devices are also controlled automatically, therefore it is interesting to look at IDS in these systems.

Supervisory control and data acquisition (SCADA) system is one of the most used ICS. It is designed to use network communications. We explain its organisation to understand how ICS are built. Figure 2.3 shows the different levels of the SCADA architecture. On level 0 are all the sensor and actor field devices such as temperature sensors or control valves. Then, level 1 contains the modules controlling the devices of level 0 and the processors they run on. On this level, there are programmable logic controllers. On level 2 are the supervisory computers running the SCADA software to controls the modules of level 1. This is the level that corresponds to the Services of the DS2OS architecture. Level 3 controls the production. It is similar to the SHE running the NLSM and KAs to control the Services in the DS2OS architecture. Finally, level 4 is used to schedule the tasks of the system, in the same way as the SLSM in the DS2OS. [23]

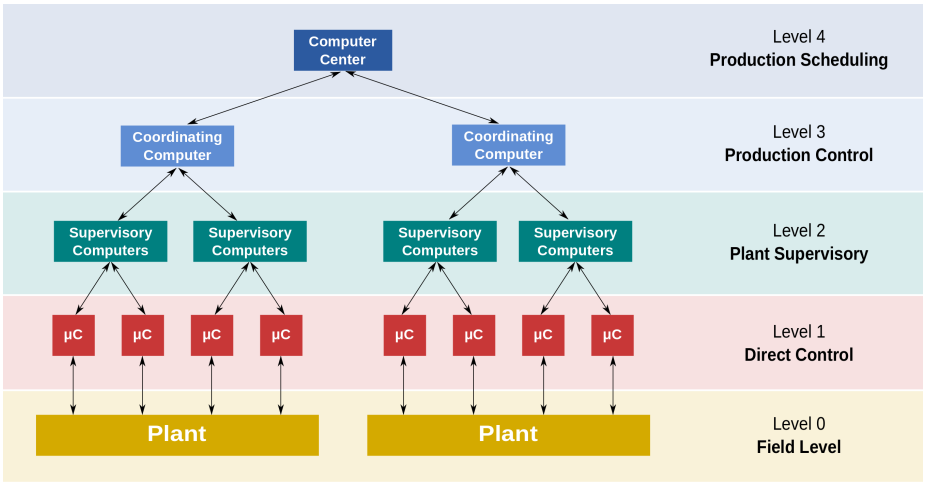


Figure 2.3: The structure of the SCADA

The traffic of these systems is quite different from the one of a conventional computer network. The number of hosts is very stable, once the system is set up the number of hosts should not change. Moreover, nodes do not change their connection partners very often, the controller of a warehouse always communicates with all the devices in it but not to the devices of other warehouses. Finally, the modules communicate with each other at frequent intervals because they run control loops and are not used by users. [24]

2.4 Anomaly Detection

We want our tool to perform Anomaly Detection, we present in this section the basics of this field.

Anomaly Detection is used in many different fields, its purpose is to detect points or patterns that do not conform to the expected behavior in data. It is interesting to find these points since they can carry more information than the rest of the data, for example a few anomalies in the brain scan of a patient could be the sign of a tumor. In our case, we want to detect anomalies in the connections between the services. Similarly to the brain scan, the normal connections in our network do not really interest us but the anomalies could point out an intrusion in the network or the failure of a service. [2] We introduce the different types of anomalies 2.4.1, then we present some limitations of anomaly detection 2.4.2, and finally we explain how the detection of anomalies can be assessed 2.4.3.

2.4.1 Types of anomalies

We state the different types of anomalies that can occur and that we would like to detect. It is essential to identify clearly each type of anomaly since different detection methods are needed for each of them. We use the example scenario of section 2.1.4 to illustrate each type of anomaly.

2.4.1.1 Point Anomaly

First, there are the point anomalies. In case, one data point can be seen as anomalous in comparison to the rest of the data, it is a point anomaly. This type of anomaly is the most simple, as it is just the fact that one data point does not concord with the rest. [2] We can see an example in figure 2.4, points O_1 and O_2 are point anomalies.

The anomalies in the access of services defined in 2.2.1.1 are an example of point anomalies. Each access that lies outside of the normal behavior is anomalous.

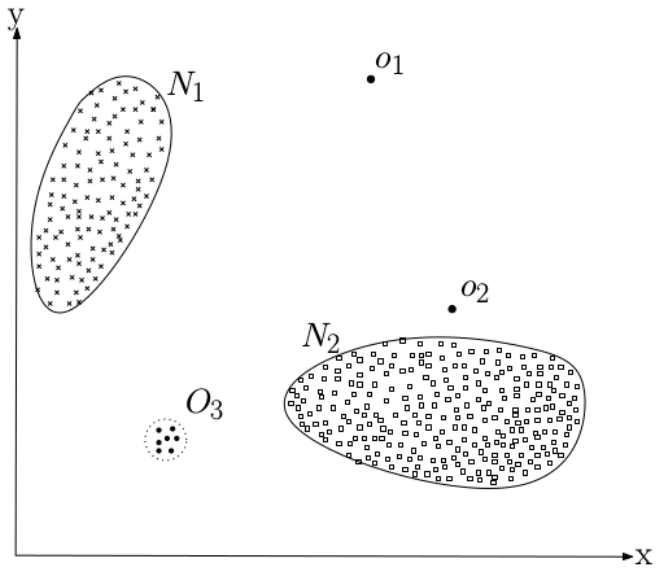


Figure 2.4: Example of anomaly detection. We can see normal regions in the data as well as some outliers. (adapted with permission from [2])

2.4.1.2 Contextual Anomaly

A more complex type of anomaly is the Contextual Anomaly. If a data point is considered as an anomaly in a specific context and only in this context, it is a contextual anomaly. The context can be, for example, the date or the outside temperature. [2]

If the central heating system of our scenario were to access the temperature of the thermometer services at 1 pm, it would be considered as a contextual anomaly. Even if the central heating system can normally communicate with these services, in the normal behavior it accesses to this information only at 5 am. Therefore, this situation is not a point anomaly because it is considered anomalous only due to the context: it is 1 pm and not 5 am.

2.4.1.3 Collective Anomalies

This last type of anomaly considers data instances with respect to the whole dataset. Data instances might not be anomalies by themselves, but the fact that they occur together is an anomaly. [2]

To illustrate this, we can look at the connections from the window services to the central heating. If the one window service begins to connect to the heating system every 5 seconds instead of every 10 minutes, it is a collective anomaly. The connection from one window service to the heating system is not an anomaly but as the frequency changes

dramatically, this set of connections is considered as a collective anomaly.

2.4.2 Limitations of Anomaly Detection

A simple approach to detect anomalies would be to define a normal behavior and label as an anomaly anything that deviates from this behavior. However, due to some factors the problem is more complex than it could seem. First, the border of the normal behavior might not be precise, a point lying very close to the border in the anomalous zone could be normal. Moreover, noise can cause this crossing of the border. Then, the normal behavior might change over time and the border would have to be set again. Another problem is that an attacker could understand what lies in the normal behavior zone and change the shape of its attacks. Finally, the lack of labeled data to verify and train models is a big problem. For these reasons, this problem is not trivial and many different techniques of Anomaly Detection exist. [2] We want to evaluate how each of these problems is relevant for this work.

2.4.2.1 No precise borders

The problem of setting a precise border between the normal and the anomalous behaviors is mostly found in continuous spaces. It does not affect the detection of anomalous accesses between the services because the services have defined IDs. However, the detection of contextual anomalies is based on the timestamp. In our example of contextual anomaly, we could ask when it begins to be an anomaly: is it an anomaly if the heating service asks for the temperature at 5:10 am? This is only 10 minutes after the normal time. In this case, a threshold has to be set to define the delimitation of the normal behavior and the anomalous one. [2]

2.4.2.2 Noise in the data

It is an important issue in general in anomaly detection, as it is hard to tell noise from anomalies apart. Our application case has the advantage that most of the data that we collect does not suffer measurement imprecisions as it is a discrete environment. However, we face this problem for the timestamp collection. [2]

2.4.2.3 Normal behavior might change

A major issue that has to be solved to perform anomaly detection is that the normal behavior might change. [2]

A basic solution to our problem would be to create rules that define what is normal or not. Then everything that is not part of the defined normal behavior would be labeled as anomalous. Like the rules that already exist in the DS2OS. However, the normal behavior might change over time. For example the habits of the person living in a smart house might change, or the number of services might increase.

If the normal behavior evolves but its definition is static, then at some point all the normal connections would be labeled as anomalous, even if they should be considered as normal. To address this issue, we want to use Machine Learning. There are different methods that could help us and we have a look at them in section 2.5.

2.4.3 Assess the detection

Once a detection method has been designed and implemented, it is important to be able to assess its performance. Different evaluation methods can be used.

First, anomaly detection can be seen as a classification of the data instances in two classes: the normal instances on one side and the anomalies on the other. Moreover, there are the two true classes to which the data instances belong. This leads to the four possible outcomes that can be seen in table 2.1. A perfect detection would have no FN or FP data points.

Classified as: \ Is:	Anomalous	Normal
Anomalous	True Positive (TP)	False Positive (FP)
Normal	False Negative (FN)	True Negative (TN)

Table 2.1: The possible outcome of the comparison of the classification of a data instance with its true class.

Using the frequency of these four possible outcomes, different metrics can be used to evaluate the detection [13]. First, the Accuracy: it corresponds to the ratio of correct classifications over all the classifications:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Then, the False Alarm Rate is the ratio of not correctly classified items to the number of items that should be classified as negative:

$$FalseAlarmRate = \frac{FP}{TN + FP}$$

The specificity corresponds to the ratio of correctly classified instances within all the

instances classified as negative. It is computed with:

$$Specificity = \frac{TN}{TN + FP} = 1 - FalseAlarmRate$$

Also called true negative rate.

Finally, the Sensitivity, that corresponds to the ratio of correct classification of positive out of all the items that should be positive, is represented by:

$$Sensitivity = \frac{TP}{TP + FN}$$

Also called detection rate, recall or true positive rate. [13]

2.5 Machine Learning

Now that we understand the principles of anomaly detection, we present understand Machine Learning (ML) to see how it can be used. Arthur Samuel that coined the term "Machine Learning" in 1959 defined it as “field of study that gives computers the ability to learn without being explicitly programmed.” [25]

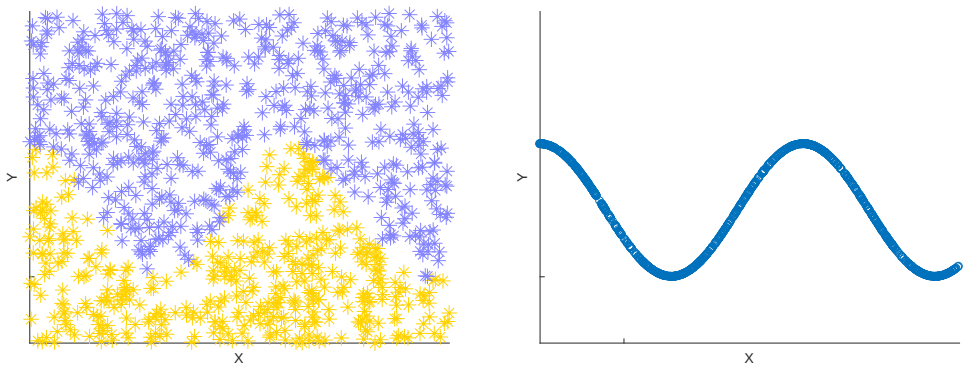
We use machine learning to learn a model defining the normal behavior. This model is used to detect abnormal behavior. There are two possible usages of machine learning that can be used to learn this model, we begin to look at the classical one and then understand the possibly more appropriate one.

2.5.1 Classical usage

First, we look at the basic principles of the classical usage of ML. As we have seen, anomaly detection is the classification of the data instances in two classes: normal instances and anomalies. Therefore, we look at how classification can be done with ML. In figure 2.5b, there is an example of two classes in a coordinate system. We, humans, would assume that there is an underlying function that describes the limit between the two classes. With the help of ML, this function can be approximated. Here we could find its equation manually, however most of the time, ML is used in application domains in which humans cannot find this underlying function. In such cases, ML is particularly useful because there would be no means to perform a classification without it.

We have a look at the characteristics of this use case.

There are clear defined phases to create the model in this application. First, the training dataset is fed to a ML algorithm that extracts relevant informations to create a model. This is the training phase. In the classification example from figure 2.5 the model learned



(a) Labeled training dataset. (b) Learned function.

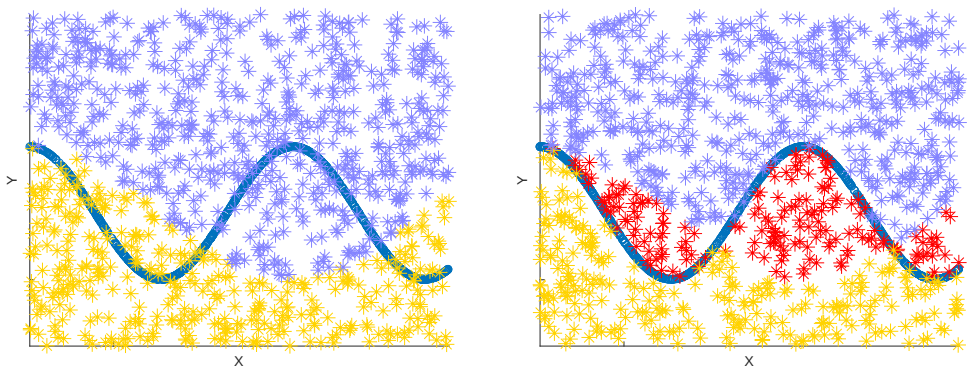
Figure 2.5: A typical usage of Machine Learning.

is the description of the classes divided by the function. Then, the created model is tested on a dataset that was not used for training. This is a way to assess the performance of the model. Finally the model is deployed and used. In the example, the model would be used to classify new unlabeled data instances. Once the model is created, it is not changed.

There are three different types of algorithms used to find an approximation of the underlying function. In the example from figure 2.5, there was a labeled dataset from which the model was learned, this is supervised learning. In this case, the ML algorithm only creates a model that reproduces the characteristics of the dataset. Then, there is unsupervised learning, there the ML algorithm extracts interesting characteristics, such as patterns or structures, from the dataset. This can be done with an unlabeled dataset. Finally, there is semi-supervised methods that can be used when a portion of the dataset is labeled. [13]

However, this training approach has a major limitation: it assumes that the distribution of the classes is static. This training approach cannot be used in applications where the properties of the target variable, which the model is trying to predict, change over time in unforeseen ways. This change is called: Concept Drift. Figure 2.6 illustrates this situation. The model is created using the classes from figure 2.5. Then the model is deployed. However, after some time the distribution of the classes changes, this change can be seen in figure 2.6a. This results in a lot of false classifications that can be seen in figure 2.6b.

As mentioned in 2.2, the normal behavior of our smart-space changes over time. This concept drift forbids us the use of classical ML; therefore, we have to use online Machine Learning.



(a) Concept drift. (b) False classifications.

Figure 2.6: The problem of Concept Drift.

2.5.2 Online Machine Learning

In some applications, the data points arrive continuously and it is not possible to store them all. Moreover, the model has to be updated if the underlying function generating the data instances changes. In such cases, online Machine Learning has to be used. In this section, we introduce the principle challenges of online ML and then see the possible ways to address them.

2.5.2.1 The Challenges

Online learning algorithms are used on data streams. Therefore, they need to operate under some unique constraints, we present them here.

The first challenge is the *one-pass constraint*. As data instances keep arriving, they cannot all be saved. In the most extreme cases, each data point can only be processed once. As most traditional ML algorithms need multiple pass on the dataset, specific algorithms have to be designed to address this constraint. [21]

The second challenge of machine learning on data streams is *concept drift*. In data streams, the underlying function generating data at a time step t could be a different function than the one that generates the data instances at time step $t+1$. The assumption that turns this characteristic into a challenge is that concept drift is unpredictable, the way the underlying function will change is not known. [21] In our system, the normal behavior will change over time and the way it will change is unknown.

The speed of the concept drift is also unknown, it can be either sudden or gradual. A sudden drift would correspond to a jump from generating function f to function g. This would correspond to the launch of a new service in our smart-space architecture or

maybe the change from winter- to summertime. A gradual transition occurs when the transition between f and g is smooth. This could for example be a gradual change in the hour of a particular communication between two services. [3]

A further challenge is the stability-plasticity dilemma. The model that is used should be updated using the data collected until time step t to classify the instances arriving at $t+1$. The classifier has to remain stable, while considering irrelevant events and be plastic when there is a change in the important data. This is known as the stability-plasticity dilemma. [3]

2.5.2.2 Reservoir Sampling

A frequently used method for ML algorithms on data streams is to use reservoir sampling, also known as windowing.

The principle is to use a window in which the last incoming data points are saved. This allows to have a small dataset containing current data instances. Almost any classical ML algorithm can run on this small sample, as long as the algorithm can be run fast enough. This method also has the advantage of providing a solution to concept drift since only the most recent data points are used by the ML algorithm. [21]

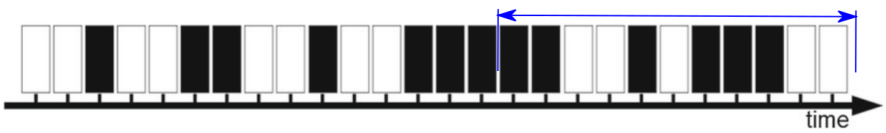


Figure 2.7: The example of a sliding window in a data stream. (adapted with permission from [3])

Figure 2.7 shows an example of a sliding window over a data stream. The black and white boxes represent data instances. The blue arrow represents the sliding window. The last 11 data instances are saved in the sliding window. When a new data instance will arrive, the oldest one will be discarded to free space for the new one.

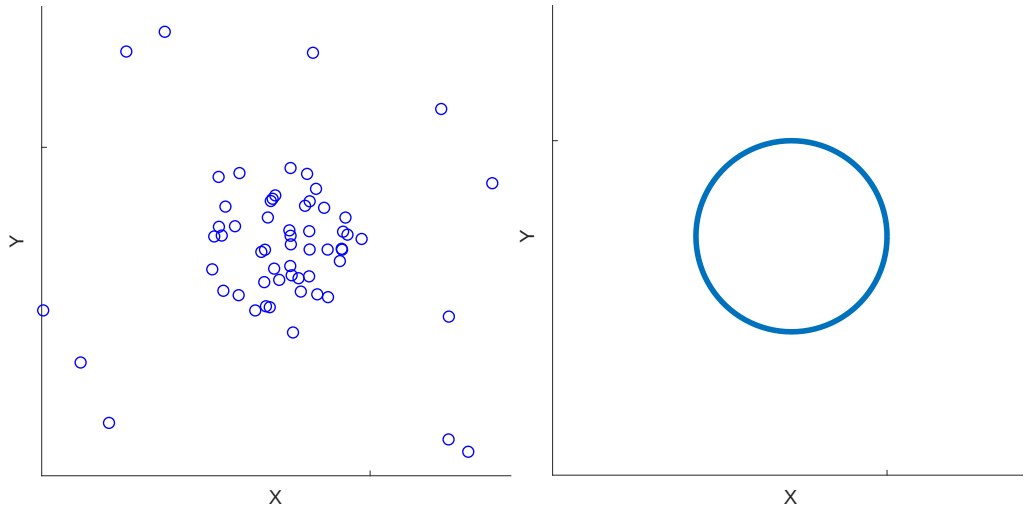
The choice of the window size is particularly difficult. The choice results in a trade off due to the stability-plasticity dilemma. If the chosen size is too big, it might contain stale data points that would hinder the creation of an accurate model. On the other hand, if the window size is too small it might not contain enough data points to be of statistical relevance. Some times, this choice is dictated by the ML algorithm used. The computation time increases, when the number of points in the window increases.

2.6 Clustering

Clustering algorithms are a subclass of ML algorithms. As we mainly use clustering algorithms in this work, we present them in this section. This allows us to illustrate the ML principles presented in the previous section.

2.6.1 The basics

Clustering algorithms are used to find aggregations of data points: clusters. Clustering is most often done on unlabeled datasets to gain insight in the structure of the data. Figure 2.8 shows an example data point distribution with the cluster that the algorithm would extract.



(a) The data points. (b) The created cluster.

Figure 2.8: An example of clustering.

There are several different methods that can be used to determine what is defined as a cluster. A common approach is to use the distance between the points, the points of the clusters are closer to one another than to any other points. It is also possible to use the density of the points in the different regions: the density of points is higher within clusters. [21]

2.6.1.1 K-means

K-means is a very commonly used clustering algorithm. It partitions the data points into k clusters. Each observation belongs to the cluster with the nearest center. The algorithm is composed of two steps: first the assignment step, then the update step. This two steps are repeated one after the other until the convergence of the clustering. [21]

In the assignment, step every data point is assigned to the nearest cluster. The distance to a cluster is computed using the center of the cluster. Frequently, the euclidean distance is used. However, other metric can be more accurate depending on the dimensionality of the data and the use case. [21]

In the update step the center of each cluster is updated. The new cluster value corresponds to the mean of all the points belonging to this cluster. [21]

2.6.2 Online clustering: CluStream algorithm

As mentioned in the previous section, in this work we use online ML to adapt the changes in the normal behavior. Therefore, we have a look at online clustering and more particularly at the CluStream algorithm.

The CluStream was presented in [26] by Charu C. Aggarwal. It is developed to handle concept drift in data streams. The main principle of this algorithm is the use of *cluster features* to describe the clusters. This particular cluster representation allows to discard the data points once they are handled.

The *cluster features* are modified BIRCH clusters [27]. They are defined by:
$(n, CF1^t, CF2^t, CF1^x, CF2^x)$ with n the number of points in the cluster, $CF1^t$ the sum of the timestamps, $CF2^t$ the sum of the square of the timestamps, and for each dimension, $CF1^x$ the sum of the values in the cluster and $CF2^x$ the sum of the squares of the values.

This cluster representation allows to merge two clusters just by adding their respective features. Moreover, for d-dimensional data points, it only requires to save $(2 * d + 3)$ values independently of the number of points in the cluster. Whereas a list keeping track of all the k points in the cluster would require $k * (d + 1)$ saved values: one value for per dimension plus the timestamp for each data point.

During the online clustering, points that are within the *maximal boundary* of an existing cluster are merged with it. A new cluster is created for points that lie outside the *maximal boundary* of all existing clusters. The *maximum boundary* is defined by multiplying a factor t with the RMS deviation of the cluster.

With the help of the $CF1^t$ and $CF2^t$, the staleness of clusters can be computed and clusters that are too stale can be deleted. The goal is to determine the average timestamp of the last m data points. With the help of $CF1^t$ and $CF2^t$, the mean μ and the variance σ^2 of the timestamps can be obtained. It is then assumed that the timestamp follow, a normal distribution. This normal disctibution can be build using μ and σ^2. This allows to compute the average of the last m data points.

2.7 The Data

In the previous sections, we presented how data can be used to perform anomaly detection and how it can use machine learning but we did not look at the type of data needed and the preparation of the dataset. In this section, we address all the issues related to the data. First, we look at why data is needed and in which form. Then, we present the different types and natures of input data. Finally, we introduce feature selection.

2.7.1 Dataset

A dataset is a data collection, a table containing data instances. Typically each line contains the list of features describing one data instance. For the example classification we saw in the ML section, it would be a table containing for every data point an x and an y value. [21]

There are two types of datasets, labeled and unlabeled ones. An unlabeled dataset contains only the features describing the data instances. These datasets are used to train unsupervised ML models. A labeled dataset additionally contains for each data instance the information to which class it belongs. Table 2.2 shows an example of a labeled dataset.

x	y	class
2	3	blue
1	2	blue
2	0.5	yellow
4	1	blue
2.5	2	yellow
⋮	⋮	⋮

Table 2.2: A section of the labeled dataset used to train the model of figure 2.5.

To evaluate the classification of a trained ML model a labeled data set is needed. The model classifies the data. Then classification is confronted to the ground truth of the labeled dataset with the techniques described in 2.4.3. The ground truth is the true classification of the data.

For ML on streaming, data things are different. As the model evolves and detects at the same time, only one labeled dataset is to be created. The detection runs on this model and is evaluated at the same time.

2.7.2 Feature selection

In the previous examples, we always presented datasets with two features, an x and y value. In reality, there are often more than two features. In IDS as in most ML usages, the first step before creating a model is to identify which features should be used [21]. The anomaly detection algorithm monitors these features to detect anomalous behavior. This is a well researched question for IDS for example in [28, 29]. In this section, we present the process of feature selection.

It is the task of the designer of the system to select the relevant features. He has to use his domain knowledge to decide which feature are relevant to fulfill the goal of the ML algorithm. [21]

However, in some cases an algorithm is used to detect the subset of features relevant to the detection. There are different algorithms to select features automatically, they can use for example the entropy. If there is a very high number of features, it might take more computational power to use them all. Furthermore, some features might be highly correlated. In such cases algorithms can be used to detect which features are the most relevant ones and which ones can be ignored. [21] These algorithms are very useful when using a high number of feature with an unknown importance.

The problem we face is that in data streams, the importance of each feature can change over time. In such cases, it might not be interesting to determine on one particular dataset which features are relevant since the relevance of these features might be specific to the dataset.

2.7.3 Nature of the input data

For every anomaly detection application, it is very important to determine the nature of the input data because it conditions the choice of the anomaly detection technique. Often the data points on which the detection is performed are composed by a set of different attributes. In the example of the ML section, the data points are a pair of two numbers x and y. The nature of the attributes determines if a distance metric can be used between the data points. With points on a coordinate system, it is easy to compute the Cartesian distances between the points, but if the attributes are labels, it can be harder. In this section, we present the different types of input data and how they can be transformed one into the other.

We identify two main types of data for our work: *continuous* and *categorical*. Features are continuous if they are described by a continuous numeric value. The x and y values of the previous dataset are continuous. This can also be for example the age of a person, a temperature or a distance. Categorical features depict the belonging to a category. The class label in the previous dataset is categorical. This can also be for example a ZIP

code or a color. [21]

ML algorithms learn from mathematical constructs; therefore, the data instances for learning have to be converted into numerical representations. It is important to find an appropriate transformation. [21]

To convert categorical features in continuous, *binarization* should be used. If x different categories are used, a vector of x binary values is created. Each value corresponds to one category. To represent a category, the binary value of this category is set to 1 and the other values are set to 0. [21]

To convert continuous features in categorical, *discretization* should be used. [21]

2.8 Characteristics of IoT site traffic

To detect accurately anomalies in the traffic within a smart space, we need to identify the particularities of this traffic. In the previous sections, we presented the different methods that we are using. In this section, we propose characteristics of IoT site traffic. Then, we look at how these characteristics could allow to design a more accurate detection.

As discussed, many aspects of our system can be found in Intrusion Detection Systems. Therefore, we could draw inspiration from these systems. However, the traffic of conventional networks and the one of IoT network seems to be different. We analyze this differences, since they allow to design a more accurate detection. [11]

Most of the IoT nodes are controlled by computer programs that act independently. Most of them have a main-loop and perform a few preprogrammed connections every time the loop is ran. This is not the case for all the IoT nodes since some of them are controlled by a user. This characteristic is similar to SCADA systems where most of the devices operate using main loops. Hence, we can draw inspiration from the characteristics of SCADA service.

First, the number of devices in the network is very stable. New services should not be added every other minute. Whereas, in traditional networks we can imagine that the users come and go using mobile computers. [11]

In addition, this number of services should have very stable connection patterns. The heating service should always access the thermometer service and one or two other services. However, in normal P2P networks every host can access every other host. [11]

Finally, the frequency on each communication relationship can be periodic since on access would be done every time the service runs its main loop. The service should then wait for x seconds and run the main loop again. [24]

Through these three characteristics, IoT traffic can be quite different from the one in

conventional networks. The traffic generated by people using computer is not periodic since they do not follow a preprogrammed time schedule. Moreover, any host can access any other host and need not to restrict themselves to a few other hosts.

2.9 Network Flows

We want to monitor connections between two particular end points, such connections are known as network flows. These are used for IDS, firewalls or monitoring. In this section we look at how they are defined and how we use them.

2.9.1 Definition

There are many possible definitions for Network Flows. According to RFC3697, they are a group of packets that share the same source IP address and destination IP address be it unicast, anycast, or multicast. [30]

Cisco router monitor Network Flows with the feature NetFlow. There a flow is defined as a group of packets having the same: source and destination IP address, source and destination port, layer 3 protocol, class of Service and router or switch interface. [31]

2.9.2 Definition of flows in the DS2OS

We use more high level features to describe the Flows. We do not want to look at IP flows in a network but at the way they can be used to describe connections between services.

We define flows as follow:
Definition 1. *A Flow is a group of connections that originate from the same Service and access the same Context Node.*

Different type of operations can be performed on one flow. This is discussed in the design chapter.

2.9.3 Use cases

It can be particularly useful to monitor the flows in addition or instead of Deep Packet Inspection (DPI) in an IDS. DPI is the process of examining the payload of the packet. Flows can be used for example to detect network scans or distributed denial-of-service attack (DDoS attack), because in these two cases there would be respectively a big number very short flows originating from one host to always different destinations or

many flows with a high throughput destined to the same host. The advantage of flows is that they require less computing power to analyse than using DPI.

Network flows have been extensively used to detect the type of applications in a network. For example HTTP traffic can be recognized solely with the flows descriptions. The port number can be enough in this case. Different methods have been used to detect P2P, VoIP, DNS, FTP, email or games. [32]

2.10 Representation of graphs

It is a common approach to visualize and represent the connections between nodes in a network in the form of a graph. The hosts in the network are represented by the vertices of the graph and the flows by the edges between them. Graphs can be represented using different data structures that have different properties regarding the complexity of retrieving particular information. Therefore, in this section we look at the commonly used data structures for the representation of graphs. [8]

2.10.1 The different data structures

2.10.1.1 Adjacency matrix

A graph can be represented as an adjacency matrix. It is a two-dimensional matrix in which the entries represent edges between vertices. The rows represent the source vertices of the graph and the columns the destination vertices. If there exist an edge from a vertex a to a vertex b the entry at the crossing of the row that represents a and the column that represents b will be equal to the cost of this edge. With this data structure additional knowledge has to be stored outside of the matrix. [8]

2.10.1.2 Adjacency list

In an adjacency list, there is a list containing all the source vertices. Each entry in this list contains a list of all the vertices that can be reached from this vertex. For the same connection as before from a vertex a to a vertex b: there will be an entry in the list for the vertex a and b will be in the list of this entry. With this data structure the additional knowledge can be stored inside the entries. [8]

2.10.1.3 Incidence matrix

Graphs can also be saved as incidence matrix. This is a two-dimensional matrix, in which columns represent the edges and rows the vertices. There is an entry at the

crossing position of a column and a row if the vertex of the row is incident to the edge of this column. [8]

2.10.1.4 Hash table

A hash table is a data structure that can map keys to values. It is implemented in the form of an array. A hash function is used on the key to compute an index into the array where the corresponding value is to be saved. Such a data structure can be used to save graphs. We can imagine the key to be the name of an edge and the value to be information about the edge. The efficiency of hash tables depends on their dimension and load. However on average most operations on hash table have a complexity of $\mathcal{O}(1)$. [8]

2.10.2 Comparison

These different data structures can be compared in regard to the time complexity of some basic operations.

Operations:	Adjacency matrix	Adjacency list	Incidence matrix	Hash table
Store graph	$\mathcal{O}(\|V\|^2)$	$\mathcal{O}(\|V\| + \|E\|)$	$\mathcal{O}(\|E\| * \|V\|)$	$\mathcal{O}(X)$
Add vertex	$\mathcal{O}(\|V\|^2)$	$\mathcal{O}(1)$	$\mathcal{O}(\|E\| * \|V\|)$	$\mathcal{O}(1)$
Add edge	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\|E\| * \|V\|)$	$\mathcal{O}(1)$
Remove vertex	$\mathcal{O}(\|V\|^2)$	$\mathcal{O}(\|E\|)$	$\mathcal{O}(\|E\| * \|V\|)$	$\mathcal{O}(1)$
Remove edge	$\mathcal{O}(1)$	$\mathcal{O}(\|V\|)$	$\mathcal{O}(\|E\| * \|V\|)$	$\mathcal{O}(1)$
Find an edge	$\mathcal{O}(1)$	$\mathcal{O}(\|V\|)$	$\mathcal{O}(\|E\|)$	$\mathcal{O}(1)$

Table 2.3: Comparison of the different graph representations. [8]

Table 2.3 compares the computational complexity of different operations in each of the presented data structures. A graph $G = (V, E)$ with V vertices and E edges is used. The complexity presented for the hash table corresponds to the average case. The X represents the dimension of the hash table.

2.11 The Problem domain revisited

We have understood the basis of network flows and anomaly detection. Moreover we have seen how online ML can be used to classify data instances of changing distribution. Our research question follows from these observations:

How can anomalous connections between the services of a smart-space be detected the most accurately in the shortest time?

We go through the requirements again with our research question in mind, this leads us to a concept for the design of our anomaly detection module.

We want to concentrate first on the *detection of anomalies in the service accesses* (requirement: 2.2.1.1). The concept that we designed still *allows the addition of the other AD modules*: periodicity AD, value AD and access patterns AD. This means that the first goal is to detect new accesses between services. To do so we want to create a graph representation of the network in which the vertices are the Services and the Context Nodes. The edges would represent the existing connections, these accesses are allowed. This way new connections can be detected very quickly.

The system should *react to anomalies and learn the evolutions of the normal behavior* (requirement: 2.2.4). As new accesses are anomalies we want to inform the user and possibly wait for his approval. The approval of the user would act as a supervision for the learning. If the access is allowed by the user it is added as an edge of the graph.

The solution has to be *fast and scalable* (requirements: 2.2.2 and 2.2.3). The problem of such graph is that it would need a lot of space to save it if there were many services and connections. In addition the computing power to look for particular connection would be high. Therefore we want to split the graph in subgraphs in order to keep the detection scalable. Each KA would have the subgraph containing all the accesses originating from the services it manages.

This simple but robust concept would handle the anomalous service access detection. Moreover it would allow to add the other modules once it is designed.

Chapter 3

Related work

To create an overview of the state of the art in the different research fields linked with our problem, we examine different areas of related work. We begin with an explanation of how the related work were found in section 3.1. Then we look at related work in the area of Internet of Things in section 3.2, then in the field of Intrusion Detection System in section 3.3, and finally in periodicity mining in section 3.4. We will conclude with an overview of the state of the art in section 3.5.

3.1 Areas of related work

We want to find existing works that address part of our research question and have similar requirements. These work will be used to create an overview of the state of the art in the fields of our research question. The requirements and challenges that we have in common with the other works are the one listed in sections 2.2 and 2.11 of the Analysis.

Different research fields address parts of our research question and have similar requirements. All of them use anomaly detection. However they rarely want to detect the same type of anomalies as we want.

In the Internet of Things, Anomaly Detection is used for different tasks. It has been used to detect misbehaving devices, to detect unauthorized connections, to detect anomalous sensor values or to detect anomalous behavior of the people in the house. We review methods and applications for each of these tasks. These works are similar to ours because they share the same application domain as well as its constraints.

The second major field that shares some of our requirements is network-based anomaly-based IDS. They also need to detect attacks in the form of anomalies. The normal behavior of the network might not change over time but it can. We particularly look at such IDS used in Industrial Control Systems. These systems also share the requirement

that they have to monitor a distributed system of industrial installations. Furthermore in contrary to most other IDS they know the high level protocols that are used and use this knowledge to perform deep packet inspection. In our case we also want to use the payload of the packet and not just look at the network packets and flows.

Finally, as we want to perform frequency anomaly detection, we review the work that make use of periodicity in network connections and in particular in network flows.

3.2 Internet of Things

We want to perform anomaly detection by monitoring the traffic within IoT sites, therefore we look at similar applications in IoT. We begin with a review of the current IoT access control systems in use. After this we look at some applications using traffic monitoring. Finally we analyse how the anomaly detection in the behavior of people is close to our work.

3.2.1 IoT access control

The existing connection management systems for IoT sites are presented here. These systems are used to filter unwanted accesses between devices. They fulfill the same purpose as the existing access control system of the DS2OS presented in section 2.1.2.

Neisse et al. [33] present a smart-space architecture using seckit [10] as access control mechanism. They use hand written rules, however they present two particular functions: a context manager and an enforcement of rules. The *context manager* allows to specify the current context of the smart space and take this into account for the detection of anomalies. They propose a *rule enforcement* method to modify the importance of rules, however it does not allow to create completely new rules. Sarkar et al. [34] present an other smart-space architecture using seckit without introducing further features.

To conclude, these systems are mainly similar to the connection management of the DS2OS that is presented in section 2.1.2. Some of them present interesting extensions, that allow a more fine grained control. However they still use hand written rules that have to be rewritten by the system administrator when the normal behavior changes. Furthermore, they can be set to allow more than what the service requires. Having more permissions than needed reduces the security, because anomalous behavior from these services could be permitted. All in all, they are very different from this work because they do not consider the automatic adaptation to changes at all.

3.2.2 IoT traffic behaviors

Very recently machine learning and IDS approaches have been used to detect anomalous traffic communications on IoT sites.

Meidan et al. [35] present a technique to classify the type of IoT devices based on the traffic patterns. These authors also presented a detection of unauthorized IoT devices based on their traffic [36]. Our approach differs form theirs since we use the device type to create a description of the service behavior and then detect single anomalous connections.

Hafeez et al. [37] present a work very close to ours. They present an approach to detect suspicious device-to-device communications in IoT-device-dominant edge networks. They perform clustering on the traffic patterns to detect anomalous behavior. However, their approach differs from ours in some points, they do not use the periodicity of the connections, they do not propose a human understandable model and they do not address changes in the normal behavior of devices.

DeMarinis et al. [11] compare IoT traffic to regular traffic caused by a laptop. Their initial finding is that IoT traffic may have some characteristics different from other network traffic that would justify the use of white-list policies. As a result of their analysis they highlight the need tor a new type of IoT security architecture and identify its four most important features: it should adapt automatically to changes in the normal communication behavior, the policies should be maintained by one or more independent authorities rather than trusting a single party, it should consider resource limitations, finally it should facilitate the installation of new devices. Even if they do not propose an approach to solve these problems, it is interesting to notice that our method provides a solution to almost all of the problematics they identify.

3.2.3 Smart spaces anomaly detection in the behavior of the people

As we want to use machine learning based anomaly detection in smart-space, we also look at how anomaly detection has been used in smart-spaces for other usages than device to device connections. The main aspects that we want to look at are mainly how these systems cope with the irregular aspects of the human behavior. The habits changes of the smart-spaces inhabitants would result in changes in the connections between the services. Therefore it is interesting for us to understand how a system can adapt to these changes.

Different smart-home experimenting prototypes have been developed like the MavHome [38], the Aware Home [39] or the Gator Tech Smart Home [40]. In the MavHome project a lot of work has been done regarding describing and learning the normal behavior of the inhabitants of the house in order to detect anomalies. The developed method uses

temporal relations to describe how the events or the actions of the user depend from one another [4]. These relations are shown in figure 3.1.

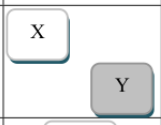
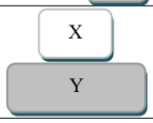
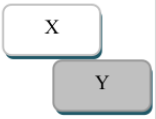
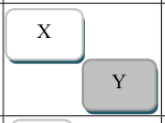
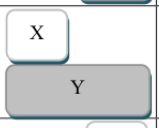
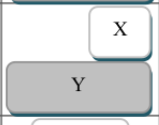
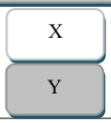
Temporal relations	**Visualization**	**Interval constraints**
X **Before** Y Y **After** X		Start(X)<Start(Y); End(X)<Start(Y)
X **During** Y Y **Contains** X		Start(X)>Start(Y); End(X)<End(Y)
X **Overlaps** Y Y **Overlapped-by** X		Start(X)<Start(Y); Start(Y)<End(X); End(X)<End(Y)
X **Meets** Y Y **Met-by** X		Start(Y)=End(X)
X **Starts** Y Y **Started-by** X		Start(X)=Start(Y); End(X)≠End(Y)
X **Finishes** Y Y **Finished-by** X		Start(X)≠Start(Y); End(X)=End (Y)
X **Equals** Y		Start(X)=Start(Y); End(X)=End(Y)

Figure 3.1: The temporal relations. (adapted with permission from [4])

The goal of the method is to use data mining to extract frequent relations and to predict the behavior of the inhabitant. A hight level concept of activities is used to describe the actions of the user. For example: the user is watching a movie. The different activities are collected and frequent relations are extracted. For example, the algorithm could learn that the user very often makes popcorn before watching a movie. Then the learned relations are used to predict the behavior of the user. If the predicted probability of an action is very different from the action happening the systems outputs an anomaly. For example if the user has not made popcorn before watching a movie even though the predicted probability was 0.97 it could be seen as anomalous. [4]

This approach is still used in most of the cases like in [41]. And even when it is not directly used the anomaly detection of behaviors in smart-homes only focuses on the

hight level activity concept like in [42]. The problem is that this concept is actually quite far away from the anomalies that we want to detect in our work. The focus is really on the activity of the humans in the house. Moreover, apart from the consideration of temporal relations it seems that traditional machine learning and data mining are used for the anomaly detection. Therefore we cannot gain knowledge about the way the user habits might change over time. For these reasons we cannot gain allot of knowledge through these work.

3.3 Intrusion Detection Systems

We design a module that detects anomalies in the service accesses. As discussed in section 2.9.2, these accesses are similar to flows in IP-networks. Therefore we will look at the different type of anomaly detection methods in general and then at which methods are used in ICS.

3.3.1 Anomaly-based IDS using flows

Many different intrusion detection systems have been designed using anomalous flow detection. Their main advantage is that they can run quicker than IDS using deep packet inspection. We will have a look at some of these methods.

For example Artificial Neuron Networks (ANN) have been used by Jadidi , Muthukkumarasamy, and Sithirasenan [43]. They created an Anomaly-based IDS that can run in real-time. To do so they used a Multi-Layer Perceptron neural network with one hidden layer. Moreover they use their network to identify the most relevant features of the dataset. They achieved a very hight accuracy of 99.43%.

Winter, Hermann, and Zeilinger [44] propose an approach to detect anomalous flows with a One-Class Support Vector Machines. Traditionally AD-based IDS are trained using the normal traffic. On the contrary, here, they trained the one Class SVM with the malicious network data. The advantage of this idea is that the detection suffers of a lower false-positive rate. However we may think that it cannot detect new attacks.

There are many more methods that have been used, sadly their application domain is quite different to ours. Therefore they fail many of our requirements. There main problem is that they all use classical Machine Learning to first train the model and then deploy it.

3.3.2 Flow-whitelisting in SCADA

In Industrial Control Systems the traffic is different, therefore different techniques have been developed to perform better on this type of traffic. The two main differences relevant for the anomalous flow detection are that the number of hosts and their connection with each other are very stable.

To respond to this particular characteristics RRR Barbosa, Sadre and Pras [45] presented the idea to whitelist the flows that are allowed and to forbid the others. RRR Barbosa has worked a lot on securing the connections in SCADA networks.

Their first approach is rather simple. First, in a learning phase the flows are monitored and added to the whitelist, then in the detection phase the created whitelist is used to filter the flows. This approach is based on two assumptions, first that there is no attacks during the learning phase otherwisebecause else it would be learned as part of the normal traffic. Then that all the normal flows are present in the learning phase, if they only appear afterwards they would be labeled as anomalous. This means that this system does not address changes in the normal behavior. The whitelisted flows are saved in a matrix. Considerations are made about the duration of the learning phase, but these are very dependent on the application network. A additional module is also added to classify the types of anomaly in four classes. The drawback of this method is the potentially high false alarm rate.

Different improvements have been proposed for this method.

Leef and Addanki [46] applied this method to regular traffic with the help of different machine learning algorithms: Decision Tree, Random Forest and Linear SVM. There results are not as good as the ones of [45] because this approach is harder to use on traditional traffic.

Lemay, Rochon and Fernandez [47] designed a technique to transform this white list in a black list in order to be able to use it with traditional rule-based IDS. This is helpful but they did not make major improvements on the concept.

Kang, Kim, Na, and Jhang [48] improved the classification of the type of attacks using deep packet inspection.

These improvements allow a slightly better detection. However, they do not propose a new concept, or solve the dependence to the two assumptions of the first method.

3.4 Periodicity Mining

We want to detect for each flow if there is a frequency at which the packets are exchanged. If this is the case packets that do not conform to this frequency should be labeled as

anomalous. We want to access how anomalous each packet is with the help of an anomaly value. The goal would be to attribute a confidence value to the mined frequency. Then if the packet is very different from a frequency with a high confidence value then it is very anomalous.

To achieve this, periodicity mining is needed. In this section we will have a look at the different techniques of periodicity mining and how they have been used. There are three main types of techniques: some are based on spectral analysis, others are using the inter-arrival times of the packets and the last ones are based on automates. As periodicity in traffic is often caused by programs, it is used mostly in botnets detection and in anomaly detection in ICS.

Botnets are a large number of compromised machines that follow the orders of a small number of bot masters via a Command and Control channel (C&C). Due to the preprogrammed behavior of the bots the traffic in the botnet is periodic. Therefore it can be sufficient to detect if there is periodicity in the traffic to detect a botnet. Thus, some of the methods are used to detect if there is periodicity more than to find out which periodicities there are. However in some cases it is trivial to obtain this additional information.

On the contrary, in Industrial Control System the different devices tend to send updates or ask for information in predetermined intervals. Therefore the traffic is very periodic. The goal in such networks is to detect deviations from the normal periodic behavior because these deviations can indicate anomalies or intrusions. This is also the way we want to use periodicity mining.

3.4.1 Spectral analysis

The goal of this method is to use the tools of the spectral analysis to discover periodicity in the data. An example is Fast Fourier Transform that can be used to find the different frequency components of the signal. Figure 3.2 shows how it can be used, it is easier to assess the periodicity of the signals in the spectral domain. These methods often use the number of packet arrivals of each flow in each time intervals as the signal.

3.4.1.1 Detect periodicity

Barford, Kline, Plonka and Ron [49] were the first to propose such a method. They used wavelets to transform a time series of the number of new flows per time interval to the spectral domain. They applied it to a moving window of fixed size. Then detected anomalies by detecting new peaks in the spectral domain that are higher than a certain threshold.

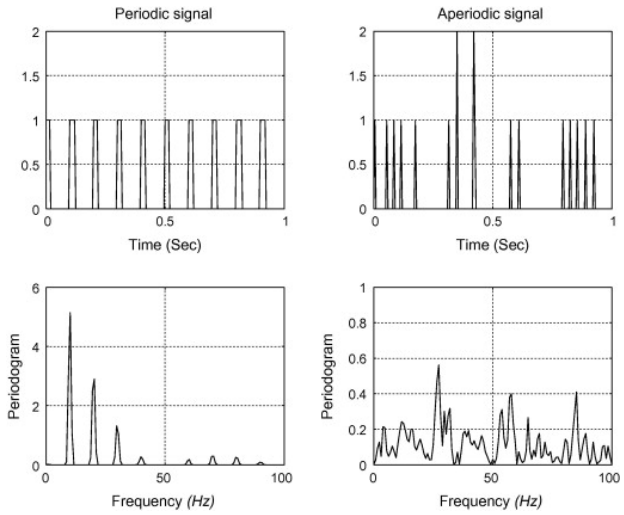


Figure 3.2: The comparison of a periodic and an aperiodic signal in time and spectral domains. (adapted with permission from [5])

AsSadhan and Moura [5] have created a sequence x[n] with the number of the packets per 100 ms time interval. Then they have confronted two hypothesis: either the sequence x[n] follows a Gaussian distribution, or it has a periodic component plus Gaussian noise. To find the correct hypothesis they compare the highest pick in the spectral domain with the other components to obtain a ratio. If the ratio is large enough it means that there is a clear periodic component. They used a poisson distribution to model the x[n] sequence.

3.4.1.2 Detect aperiodicity

Barbosa, Sadre, and Pras [6] present the first prove of concept of frequency anomaly detection in the connections of SCADA. Figure 3.3 shows the different modules of their system. First comes the Traffic Capture module, it collects the packet that use SCADA protocols. Then the Flow Creation module, aggregates the packets in flows. If the flow is new its normal behavior has to be learnt, this is the role of the Periodicity Learning module. If the flow is known the new packets are compared with the learned frequency fingerprint of the flow. If the packet does not match the fingerprint it is labeled as anomalous. They have shown the feasibility of the approach but they did not automate it.

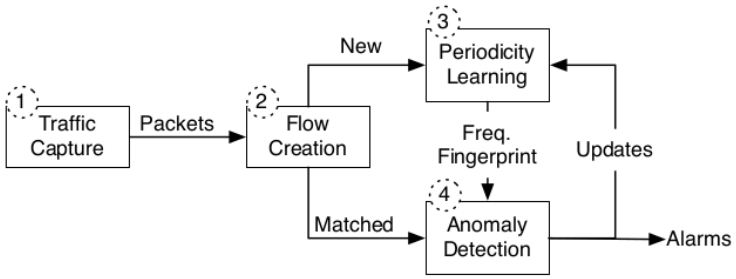


Figure 3.3: The different modules of the frequency anomaly detection developed in [6]. (adapted with permission from [6])

Cheng, Kung and Tan [18] proposed a method to use spectral analysis to identify malicious DoS attack traffic. They use the number of packets per time slot to create the signal. Then they transform it in the spectral domain to detect if there is periodicity or not. The DoS traffic is not periodic, whereas the normal TCP traffic often has its round-trip time as period. The advantage of their method is that is can differentiate which packets are anomalous and which fit the TCP normal traffic frequency. This allows to discard only the unwanted packets.

3.4.2 Inter-arrival times

The idea is to use the value of time duration between two packets to detect if there is a particular periodicity.

Hubballi and Goyal [50] present an approach using the standard deviation of the inter-arrival durations for each flow of a host. If the standard deviation is below a certain threshold the flow is considered to be anomalous. The limitation of this system is that it is only able to detect frequencies of packets that have always the same time interval: a frequency of one packet every ten minutes will be detected, but two packets with ten seconds interval every ten minute will not be detected as a frequency.

Bilge et al. [51] created a model to detect botnets. They trained a random forest classifier to classify flows as botnet commands or normal traffic behavior. Therefore the focus is to detect if there is a periodicity. Most of the features fed to the classifier are created using inter-arrival durations: the average standard deviation of the flow, the average standard deviation from the last 300 seconds plus the minimum, maximum and median inter-arrival times.

3.4.3 Automata

The last approach does not only consider the period between the arrival of two packets, it also considers the order in which the packets arrive. Such systems use deep packet inspection and have knowledge of the used communication protocol. This allows to extract the type of operation performed with each packet. With domain knowledge, an automaton is created that describes in which order the different requests should arrive and with which time interval. Any deviation from the behavior described by the automaton is seen as anomalous.

Goldenberg and Wool [52] propose to model ICS traffic using the Modbus protocol with such an automaton. Their approach creates the automaton given a training set without any anomalies. However their approach encounters difficulties when dealing with cycles with multiple periods. Furthermore, their approach does not take into account the duration between two packets into account.

Barbosa, Sadre, and Pras [53] propose an approach using message repetition and timing information to automatically learn the periodic patterns of the traffic. These learned periodic patterns can then be used to detect the traffic that does not fit in them. They are able to learn periodicities on different time scales, the different periodicities of each command in a flow even if there are small timing variations. The learner module, that extracts the sequences uses a system of candidate: when a sequence of periodic requests is detected a candidate is created. Then each new request that fit in this period is added to the candidate. Finally the different candidates are confronted to one another. They are evaluated using the duration from the first packet to the last and the number of packets fitting their description.

This approach is promising even though it has some general limitations. It requires the knowledge of the protocol, as well as a not too complex patterns, to avoid in over complicated automates. The second limitation is that, as for now, these approaches require normal traffic to be trained and do not allow to learn the automaton online.

3.4.4 Comparison of the approaches

In conclusion, using inter-arrival times seams to be easier. But the existing approaches have more difficulties to detect complex frequencies or frequent burst of packets. In the case of frequent bursts of packets the average inter-arrival duration would be very short because of the burst. Therefore it is easier to detect this kind of periodicity using spectral analysis techniques. Spectral analysis techniques on their side suffer from imprecise periods. [53] Automates can be useful and more robust than the others, with a well known protocols and an application domain that does not suffer concept drift.

It is also important to mention that all these approaches focus on the periodicity within

flows. This allows them to be scalable to bigger networks because hosts can implement these methods, it does not have to be a network level monitoring.

3.5 Conclusion

To compare the different related work we use table 3.1. It allows us to have an overview of the sate of the art. We use different categories to classify the papers we found. First, one category for each type of anomaly we want to detect: flow monitoring and frequency AD. Then we want to know for which usage the different methods were designed: IoT, IoT access control, IDS, IDS for ICS. We do not look at the papers that we found for anomaly detection in the behavior of people because they are quite different to our approach. Next we want to know if the detection did already run in real time and if the detection can adapt to a changing normal behavior. Some detection methods were only tested on datasets, this means that even if they could run in real time it is not presented. We also want to know if the presented design addresses the issue of scalability and the creation of the dataset used to access the detection.

The analysis of table 3.1 reveals that only about a third of the related work are taken out of the area of IoT. Furthermore, very few related work handle concept drift. All the related work that handle it, are used in frequency anomaly detection. There, the use of a sliding window allows to handle concept drift more easily.

Papers:	Flow AD	Frequency AD	IoT	IoT access control	Conventional IDS	IDS for ICS	Runned in real Time	Can handle concept drift	Address scalability	Address the dataset creation
Neisse et al. [33]	..	..	..	✔	..	..	✔	..	✔	..
Sarkar et al. [34]	..	..	..	✔	..	..	✔	..	✔	..
Pahl et al. [17]	..	..	..	✔	..	..	✔	..	✔	..
Meidan et al. [35]	..	..	✔	..	..	..	..	..	..	✔
Meidan et al. [36]	..	..	✔	..	..	..	..	..	..	✔
Hafeez et al. [37]	✔	..	..	✔	..	..	..	..	✔	✔
DeMarinis et al. [11]	..	..	..	✔	..	..	..	..	..	..
Jadidi et al. [43]	✔	..	..	..	✔	..	..	..	..	✔
Winter et al. [44]	✔	..	..	..	✔	..	..	..	..	..
Barbosa et al. [45]	✔	..	..	..	..	✔	✔	..	..	✔
Leef et al. [46]	✔	..	..	..	✔	..	..	..	..	✔
Lemay et al. [47]	✔	..	..	..	..	✔	✔	..	..	✔
Kang et al. [48]	✔	..	..	..	..	✔	..	..	..	..
Barford et al. [49]	..	✔	..	..	..	✔	..	..	✔	..
AsSadhan et al. [5]	..	✔	..	..	..	✔	..	..	..	..
Barbosa et al. [6]	..	✔	..	..	..	✔	..	..	..	..
Cheng et al. [18]	..	✔	..	..	✔	..	..	✔	✔	..
Hubballi et al. [50]	..	✔	..	..	✔	..	..	✔	✔	..
Bilge et al. [51]	..	✔	..	..	✔	..	..	..	..	..
Barbosa et al. [53]	..	✔	..	..	..	✔	..	..	..	..

Table 3.1: Comparison of the different related work.

Chapter 4

Design

In this chapter we discuss the design of our anomaly detection system. First we present the concept and the architecture of our approach in section 4.1. Then we explain our feature selection process in section 4.2. We present the two principal components of our architecture in sections 4.3 and 4.4. We show our first approach in section 4.5, followed by our final approach in section 4.6. Finally, we present our approach to detect frequency anomalies in section 4.7.

4.1 The architecture - Echidna and the Sphinxes

In this section we give an overview of the architecture. First, we remind the requirements and then we present the different modules used.

The system that we want to build should follow the requirements set in section 2.2.

- Detect anomalies in: service accesses and period (requirements: 2.2.1.1 and 2.2.1.2)
- Allow the addition of the optional modules. (requirement: 2.2.1)
- Allow the learned models to be visualised and understood by humans. (requirement: 2.2.5)
- Be integrated and scalable inside the distributed architecture of DS2OS (requirement: 2.2.2)
- Learn and adapt to the behavior of the network (requirement: 2.2.4)

To answer the first requirement, the detection of anomalies in the service accesses, we want to monitor the connections between the services. A solution to do so could be a service in the VSL middleware to which descriptions of the accesses between services would be sent. This service would perform the detection on the collected connection descriptions. The advantage of this method is that this module could monitor the whole network. Anomalies on the network level like scans of the network could be detected. But on the other hand this solution is not very scalable, if there are hundreds of services

exchanging thousands of connections per seconds the detection might be to much to handle for one service.

Another possibility would be to implement an anomaly detection module in every KA. This module could detect unexpected connections to the services it manages. This would be a scalable solution since the number of KA rises with the number of services; therefore the number of services monitored by the anomaly detection module would never be higher than the capacities of the KA. However it would not have a view of the whole network.

We propose a solution using a combination of both approaches. There is one anomaly detection module on each KA called the Sphinx. In addition, we use one coordinator module called Echidna. The Sphinxes of the whole smart space are coordinated by the Echidna, this allows the detection to be both scalable and network-based. Figure 4.1 shows an example of this organization.

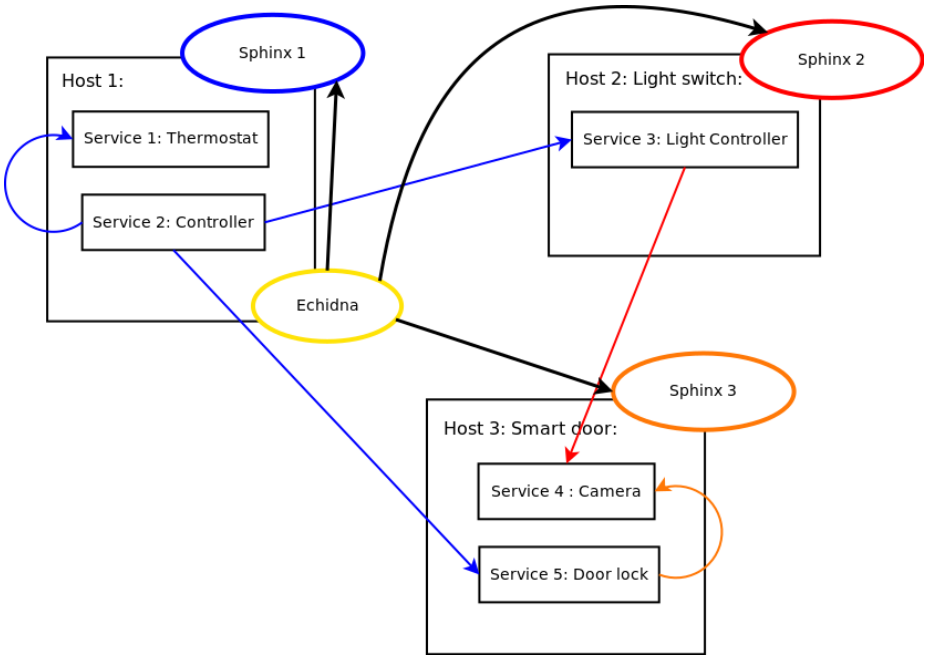


Figure 4.1: The different detection components in an example smart space.

In the rest of this section we have a look at the two modules, their functions and their places in the architecture. For each we argue why it is the best choice in regard to our requirements.

4.1.1 The Sphinx

One instance of the Sphinx is present on each KA. Its role is to monitor service accesses that are routed through this KA and to detect anomalies in them.

Monitoring and detecting anomalies on each KA allows to have a system that is scalable. Independently of the number of services in the smart space, every KA only hosts as much services as it can control. This means that the Sphinx always monitors a limited number of services. Moreover this module can be directly implemented within the KA. This speeds up the time needed to access the normality of a connection.

This design choice should not lead to major detection problems. The system is not completely a network-based IDS, because only a subset of all the connections between services are treated by each KA. The Sphinx acts as a network-based IDS for the subset that it monitors and thereby is able to detect all the attacks that such a system could have detected. The problem is that some attacks might need a view of the whole network to be detected or analysed. This is why there is also the module Echidna presented in part 4.1.2.

In this organization each Sphinx monitors a subset of all the existing accesses. We have to make sure that every access is monitored by a sphinx and that if possible no access is proceeded by two Sphinxes, which would consume unnecessarily additional computing resources. We decide that each Sphinx monitors only the accesses originating from services hosted on the same KA as the Sphinx. This organization can be seen in figure 4.1, where the color of each edge indicates the Sphinx that monitors it. Every service has to be run on a KA and only one KA; therefore, each access is monitored by exactly one Sphinx.

4.1.2 Echidna

The Sphinxes do not allow to have a view of the whole smart space, because they only monitor the connections from there services. To overcome this problem we design the module Echidna that coordinates the Sphinxes and centralize the information they gather. Furthermore, Echidna provides the interaction point with the user. This module is designed in section 4.4.

4.2 Feature Selection

Before the design of the different modules, we discuss the features that are extracted from the DS2OS connections. With this section we arrive at the *data preparation phase* of our methodology (section 1.2). It is needed before going to the modeling phase. We explained the purpose of this phase in section 2.7.2.

We need different features than traditional IDS. In conventional IDS the features that are used are often from the Network and Transport Layer of the OSI model, for example IP address or port number. However in our case these features are quite useless because all packets are exchanged on similar ports and many services can use the same IP address. Therefore we use features of the application layer, we want to use the service ID instead of the IP address. We use the features describing the service accesses in the DS2OS.

4.2.1 Selected features

To represent each connection we use a list of features that describes each of its aspects. In this subsection we discuss the features which were chosen and why. Some features are not present for every connection, when a feature does not exist for a connection the field of this feature is filled with ”*none*”.

4.2.1.1 Accessing service ID

Every service has a unique identifier. The accessing service ID is the unique identifier of the service from which originates the access. This feature is similar to the source IP address of an IP packet. This feature is needed to describe an access. Moreover, this feature is present for the description of every connection.

4.2.1.2 Accessing service address

If a service is registered by a KA, it has an address in the knowledge organisation of the VSL. This feature is the address of the service from which originates the connection. If a service is not registered, this feature is ”*none*”. This feature can be useful to detect if there is a KA on which more anomalies happen than on others.

4.2.1.3 Accessing service type

Each registered service has a model that describes the organisation of its knowledge nodes. This is the type of the service. All the temperature sensor services of the SmartSpace would have a similar type, for example "*temperatureSensorService*". This feature is useful to detect is a particular type of service in the whole Smart Space generates the same type of updates. It also allows to compare the behavior of similar services. Once again, the value is ”*none*” if the service is not registered.

4.2.1.4 Accessed node address

This is the equivalent of the destination address of a connection. It is the other end of the connection, the address of the knowledge node that is accessed. This is feature has to be present for every connection as it defines one of the end points.

4.2.1.5 Accessed node type

Each knowledge node has a type, for example *text*, *number*, *list* or *composed*. We also want to save this information.

4.2.1.6 Accessed service address

This is the address of the service to which the accessed knowledge node belongs. It is useful to log it in order to detect which anomalies come from which service.

4.2.1.7 Accessed service type

As for the type of the type of the accessing service, it can be useful to log the type of the accesses service.

4.2.1.8 Operation

There are different type of accesses. If, for example, we want to label as anomalous the fact that a service that was only reading a value begins to change it, then we need to use this feature. The different type of accesses are discussed in subsection 4.2.2.

4.2.1.9 Value

Be it a read access or a write access, a value is exchanged. A thermometer service would write the temperature value its Context Node, and other services would read this value. For the value anomaly detection as defined in 2.2.1.3 it is needed to monitor the value exchanged to detect anomalies in comparison to the other values exchanged in this flow.

4.2.1.10 Timestamp

Each access is made at a certain point in time, this point is described by the timestamp. This is a vital feature to perform periodicity anomaly detection. Moreover for value anomaly detection it can be used to weight the values, it is normal to have a higher difference after a longer time. The timestamps used are in Unix time.

4.2.1.11 Normality

As we want the dataset to be labeled, we add this feature to describe if the connection is normal or not. However, this feature is not used once the system is deployed.

4.2.2 Type of operations

There are a number of different operations that services can perform. For each of them the information that are logged into the dataset are quite different. In this subsection we go through each of them and explain what they do and what information we extract from it.

4.2.2.1 Read and write

These are the most simple operations, a service reads or write a new value in a Context Node. All the features listed before are used.

4.2.2.2 Subscribe and un-subscribe

Services can subscribe to particular Context Node. Once subscribed, when the Context Node is modified, they receive a notify-callback. The subscription operations contain no *value*. Apart from this, all the features are present. The accessing service in this context is the service that is subscribing or un-subscribing to the accessed node.

4.2.2.3 Notify-callback

Once subscribed, if the Context Node changes, a notify-callback connection is issued. There is no *value* in this packet either. The accessing service is the service that had subscribed to the Context Node. The accessed node is the Context Node.

4.2.2.4 Register and un-register virtual node

Services can create virtual nodes, to do so they have to register them by issuing a register virtual node operation. It can be unregistered with an un-register virtual node operation. There is no *value* for these two operation either. The accessed node in this case is the virtual node that is being registered or un-registered.

4.2.2.5 Register and un-register service

Services are to be registered in the VSL before they can be used. These operations are done at the creation and the deletion of a service. There is no accessed node, nor value. All the fields are set to "*none*" except the service id, its address and type if known, the operation and the timestamp.

4.2.2.6 Subtree operations

Services can perform operations on whole subtrees, they can lock them and then either commit the changes made or discard them. In this case the accessed node is the parent node of the subtree on which the operation is performed.

4.2.3 Summary

All in all, there four features that have to be used to describe each connection: *serviceID*, *accessednodeaddress*, *operation* and *timestamp*. The other features are present dependent on the operation type. The list of all the features is shown in table 4.1

Feature name	Type	Example values
Accessing service ID	Text	service1
Accessing service address	Text	/kaName/service1
Accessing service type	Categorical	sensorService
Accessed node address	Text	/kaName/service2/temperature
Accessed node type	Categorical	number
Accessed service address	Text	/kaName/service2
Accessed service type	Categorical	sensorService
Operation	Categorical	$read$, $write$
Value		37.5, sunny, {$beer$, $lettuce$}
Timestamp	Continuous	847690962,1513093731
Normality	Categorical	"normal","anomalous"

Table 4.1: List of the attributes defining a connection instance with their type.

As explained in section 2.7.2, there are feature selection algorithms that can be used to select the features that should be used. However, in data streams, the relevance of the features can change over time. Therefore, we use all the features available.

4.3 The Sphinx

The Sphinx is the module that is on present on each KA. Its role is to monitor the service accesses that are routed through this KA and to detect anomalies in them. It means that all the connections originating from the services run on the KA are monitored.

In this section we present the design of the Sphinx. First we explain its name, then we explain how accesses are monitored and how their descriptions are saved.

4.3.1 Naming

As Phil Karlton said: "There are only two hard things in Computer Science: cache invalidation, naming things, and off-by-one errors." We solve here the problem of "naming things" for our module on each KA.

The mythological Sphinx was a beast send by Era to the city of Thebes. She asked a riddle to all the passing voyagers and would allow them passage only if they answered correctly. [54] In a similar way, we want our system to question the normality of all the passing accesses; therefore we name it the Sphinx.

4.3.2 Keeping track of observations

The Sphinx monitors all the accesses originating from the services hosted on the same KA. These accesses are seen as they occur. We want the sphinx to keep track of what kind of access have been made prior the the one been monitored.

To keep track of the accesses observed, we summarize them in flows. We use the definition given in section 2.9.2: A flow is a group of accesses that originates from the same Service and access the same Context Node. Each edge in figure 4.1 can be seen as a flow.

We describe each flow with its characteristics in an edge data-structure, and we organize all these flow descriptions in a flow-list. The flow list is described in the next section and the edge data-structure in the section after.

4.3.3 The Data-Structure of the flow-list

The monitored flows could be saved as a simple list but the time complexity needed to find out if a flow is allowed or not would be too high. Therefore we need to define and create the most appropriate data-structure to model the flow-list. We look first at the defining requirements in the following section. Then we consider the different options in section 4.3.3.2. Finally we present our solution in section 4.3.3.3.

4.3.3.1 Requirements

In this section we review the requirements of the data-structure used to represent the flow list.

First, we want to use data-structures used to represent graphs. One of the advantages of using flows is that it is a human understandable approach. As flows are actually connection relationships between services of a same site. This means that they can be represented as edges of a graph connecting the services. It would make the approach more human understandable. Moreover data-structures that can be used to represent graphs are well studied. For these reasons we want to represent our list in the form of a graph.

Then, we have to analyse what kind of operation is done frequently to select data-structures on the time complexity of these operations. The graph is only created once, at the start of the system. Therefore the complexity of this operation is not very important. If a new service is started or stopped, a vertex has to be added to or deleted from the graph. This is not a very frequent operation. Therefore it is not the most important complexity. If a new flow is detected, an edge is to be added to the graph. This case arises more often. However the operation that is needed the most frequently is to find an edge. For each incoming access, the module looks for the corresponding edge. This is an operation that happens by order of magnitude more often than the other in a normal usage of the DS2OS. Therefore the computational complexity of finding a given edge is the factor of choice for the data-structure.

Finally, it would be helpful to be able to save the flows using the edge data structure. Moreover it would be great to have a class used to represent the vertices of the graph and one for the edges of the graph. This organization would be very useful for the additional modules.

4.3.3.2 The different options

The different data structure that are appropriate for this task are listed in section 2.10.

We have to find and prove the computational complexity of finding an edge in the graph for each data-structure. Even if the table 2.3 shows the computational complexity of finding an edge, it is not the cost that applies in our case. The table shows the complexity of finding an edge between two vertices 2 and 3 of a graph given that their storage positions are known. This operation has a complexity of $\mathcal{O}(1)$ in an adjacency matrix because only the value stored in the matrix at position (2,3) has to be read. For an adjacency list if the storage position of the vertex 2 is known, but once at this position the algorithm has to iterate through the list of adjacent vertex to find out if there is such an edge. For a hash table only the hash of the two vertices has to be computed and then

the corresponding index in the table is read.

In our case the storage place is not known, therefore the complexity is different. The storage positions is known when the vertices of the graph are labeled with different integer. However in our case the vertices are named accordingly to the services that they represent. In the case of the adjacency matrix it means that the vertex name would first need to be mapped on the index of the corresponding row or column. This operation has a complexity of $\mathcal{O}(n)$, where n is the number of elements in the list, because it corresponds to finding a given element in a list. [8] As we would have two lists of vertices the complexity would be of $\mathcal{O}(2|V|) = \mathcal{O}(|V|)$ for a graph $G = (V, E)$.

Contrarily to the adjacency matrix, the hash table can perform as good with the names of the services as with number. Only the hash function is to be changed and this is enough to have a working hash table with name for the vertices. Moreover, the computational complexity of finding an edge stays the same. [8]

Operations:	Adjacency matrix	Adjacency list	Hash table
Find edge with name	$\mathcal{O}(\|V\|)$	$\mathcal{O}(\|V\|)$	$\mathcal{O}(1)$

Table 4.2: Comparison of the computational complexity of finding an edge using the name of the vertices that it connects for a graph $G = (V, E)$.

Using the previous discussions we can fill the table 4.2. The best option seems to be the hash table because finding an edge in it has a computational complexity of $\mathcal{O}(1)$. However the organization of an adjacency list would be helpful to fulfill the first requirement: the organization of the flows as an adjacency list is easier to understand for humans than a hash table.

4.3.3.3 The hash list

To conciliate the advantages of the two approaches we want to use a hash list as presented in [55]. The idea is to use an adjacency list but with hash tables instead of lists.

We use a first hash table that associates a the name of a service with the class instance that describes it. This description then contains a second hash table that associates the destination service of the flow to the instance of the edge data-structure that describes this flow. This way, similarly to an adjacency list, each vertex has in its description the other vertices to which it is connected.

However the complexity of finding a particular edge is of $\mathcal{O}(1)$. First the corresponding origin vertex has to be found, then the corresponding destination vertex. These both operations require $\mathcal{O}(1)$, this means that finding an edge require $\mathcal{O}(2) = \mathcal{O}(1)$.

We changed the implementation of the hash table to allow to iterate over all the elements present in the table. When a new element is saved in the table, its key is also added in a list containing all the keys of the elements in the table. This allows to iterate over all the elements in the table if needed. However, this operation has the same complexity as iterating over a list. Therefore it should only be used when specifically needed.

To conclude, the hash list data-structure and its implementation allows us to fulfill all the requirements defined in section 4.3.3.1.

4.3.4 The edge data-structure

The previous system allows us to create an instance of the edge data-structure for each flow that is observed. This data-structure is used to save all the information concerning a particular flow. In this section we review what these describing variables are. The information contained in this data-structure is extracted from the connection packets of the described flow.

4.3.4.1 The defining variables

The two defining features are the origin and the end vertices of the edge. These correspond to the service ID of the service from which the connection originates and the *accessesnodeaddress*. These two features define the source and the destination of the operations.

In addition, a counter of the number of accesses observed on this flow is saved. As well as the first and last observed timestamps.

4.3.4.2 Additional variables

There are several additional features that can be useful for a more complete detection. We present them here.

First the address of the two services playing a role in the connection can be useful. The *accessingserviceaddress* is the address of the source of the connection. As we have seen in the creation of the dataset, this address might not exist if the service is not registered. The *accessedserviceaddress* is the address of the service from which a knowledge node is being accessed.

Then, the types of the two services can be saved. We have then the *accessingservicetype* and the *accessedservicetype*, again, the first one might not exist. The type of the accessed Context Node is also saved. It is also interesting to know if the accessed knowledge node is a virtual node. This information is also saved in the edge data-structure.

4.3.4.3 The operations

If a service is allowed to read a particular node it is not necessarily allowed to change its value. Therefore, the edge data-structure should also keep trace of the operations performed in this flow. As described in the sub-section 4.2.2, there are different type of operations that services can perform.

We will classify the operations in two types: on one side the operations that only read the value of the node and on the other side the operations that modify the value of the node. The first group is composed of the operations: *read*, *subscribe*, *unsubscribe* and *callback*. The second group of operations is composed of the *write* operation, as well as the *registervirtualnode* and *unregistervirtualnode* operations.

The operations that have been monitored on a flow are saved in a list in the edge data-structure.

4.4 Echidna

One instance of the Sphinx is deployed on each KA. This means that each Sphinx in the smart space only monitors a subset of the services in the smart space and their connections. As explained in section 4.1, we use an additional module, Echidna, to coordinate the Sphinxes. This one module to rule them all is presented in this section.

We present all its roles and the problematic linked with them in subsection 4.4.2. Then we consider its place in the architecture in subsection 4.4.3. Finally we address how it handles user-interaction.

4.4.1 Naming

Again, we face the problem of "naming things".

Even if it was Era that send the sphinx to Thebes, Echidna is the mother of the Sphinx. [56] As the mother of the Sphinx Echidna looks after her daughters and and coordinate them, we choose to call this module Echidna. Plus, I found the name Era to be too short.

4.4.2 The Roles and requirements

The Echidna has different roles that we define here.

First, it provides a network-level anomaly detection. It means that it is able to detect anomalies that can only be seen by looking at the whole smart space. This function is presented in section 4.6.

Then, Echidna handles the communication with the user. This is done via the visualization of the connections in the smart space and an Android Application used to handle the user input needed that can be needed for different detection methods.

Echidna also handles the communication with the store. If the detection should be changed in some way, it is to be communicated to Echidna.

This hierarchical architecture allows the system to be scalable to any number of services and, at the same time, to have the advantages of a network-based IDS.

The main requirement for this coordinator module is that it should not have to perform tasks that consuming more computing ressource the more services there are in the smart space. Echidna has to be scalable: it should be able to function no matter the number of service in the smart space.

4.4.3 The place in the architecture

In this subsection be present the place of Echidna in the detection system and how it interacts with the Sphinxes.

Echidna is started as a service in the DS2OS. There is only one Echidna module in each smart space. Echidna looks for new Sphinxes in the smart space every two seconds and starts a communication with every detected Sphinx.

All the detection state, shared in the whole smart space, are handled by Echidna. The sensibility level used in the first approach 4.5.2.2, for example is set globally and sent to the Sphinxes.

Echidna needs some of the information detected by the Sphinxes. It needs all the edges and vertices created by each Sphinx, and the anomalies that they detect. The communication happens through the VSL. The Sphinxes have knowledge nodes to publish their state. Each Sphinx has a list of the services it monitors and a list of the edges it monitors. These edges are only the edge originating in the monitored services. This way each connection is monitored only by one Sphinx. The communication via the VSL is explained more in detail in section 5.2.

4.4.4 Antigone

To allow user-interactions we designed a small android application. It is in direct connection with Echidna. It is a basic interface that can be extended.

4.4.4.1 Naming

Antigone helped her dad, Oedipus, to travel once he became blind. [56] In a similar way, this application allows the user to interact with the smart space even if he cannot see it.

4.4.4.2 Roles and design

Antigone has two roles, it notifies the user when an anomaly is detected in the smart space and it allows the user to give feedback to the detection.

When an anomaly is detected by a Sphinx, it is send to Echidna, that sends it to Antigone. The user receives a notification on his smart phone to show where the anomaly is located and between which services. This way the user is kept in the loop and he can understand what happens in the smart space.

Then, the user can decide to confirm that it is an anomaly or decide to allow this behavior. This information is sent to Echidna, that sends it to the Sphinx that detected the anomaly.

The way the user interaction is taken into account is described more in detail in the sections 4.5 and 4.6.

4.4.5 Visualization

To allow an even better understanding of what is going on in the smart space, we propose a visualization of the connections in the smart space. An example of this visualization is shown in figure 4.2.

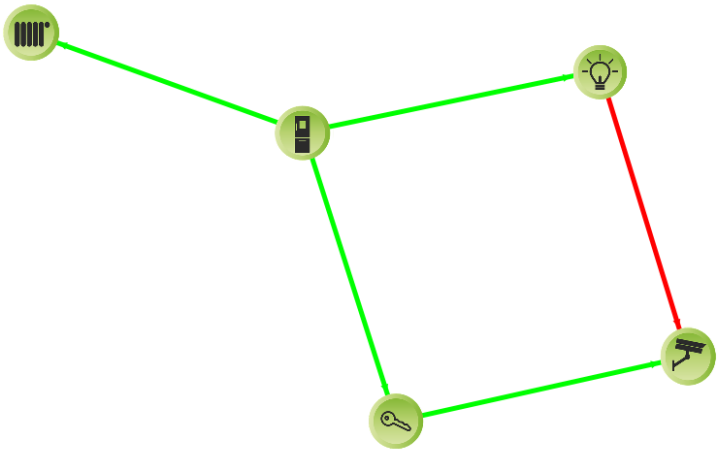


Figure 4.2: The visualization of the connections in the smart space.

This is a first look at the system, the user can learn more when clicking on the edges. Then more information about the flow are displayed. When an anomaly is detected on an edge, it is colored in red, to inform the user.

4.5 First approach: static white-list

We present in this section the first approach that we designed. It builds up on the architecture we presented in the previous sections. The goal of this approach is to use a flow white-list to allow or forbid accesses. This approach has a lot of limitations and is not the final approach we present, however it contain the basis on which we built the final approach. We presented this approach with a poster at the Passive and Active Measurement conference in Berlin the 26/03/2018 [57].

We argue in section 4.5.1 why a white-list should work and how we use it. Then, we describe in section 4.5.3 how the white-list is created. Finally, we present the limitations of this approach in section 4.5.4.

4.5.1 Flow white-listing

We want to create a model to detect anomalous connections. It has to be created by learning and adapted over the time.

As discussed in section 2.8, the connections in a smart space are very stable. This can be taken into account to design a model that is more accurate. This characteristic is shared with ICS, as seen in section 2.3.2, therefore it make sense to look at what has been developed to address this issue in ICS. Moreover, we reviewed different approaches for access monitoring in section 3.3.

We use a flow-based approach to monitor accesses. Similarly to [45], we use flow white-listing . We represent the smart space network as a graph, where the services are vertices that are connected by edge representing the allowed flows. These flows are the white-listed ones. However, we will not consider IP flows but connection flows as defined in section 2.9.2.

Flow white-listing has many advantages. First, list and graphs are well studied data-structures, which allows us to have an optimal detection and training speed. Then, this model allows a simple deterministic classification as long as the model follows the concept drift accurately. Finally, this model can be visualized quite easily.

The reason we can use this method is because the traffic is very stable. However, new services could be added or new connections could be normal, this would be the concept drift in this application domain. The normal behavior can evolve over time and we need to define a way to address this by adapting the white-list, this is discussed in section 4.5.3.

4.5.2 Use the flow-list as white-list

The white-list is placed on the Sphinx in order to filter the accesses that it monitors. The Sphinx already has a list of all the flow that it observes. Moreover, we prove in section 4.1.1 that all accesses in the smart space are monitored by one Sphinx that keeps track of them in its flow-lists. Therefore, we extend the observation flow-list of the Sphinx to be able to use it as a white-list. We add a boolean in the edge data-structure that represents if the edge is allowed.

4.5.2.1 Different definitions of anomalies

There are different granularities of anomalies that can be identified. If a whole flow is anomalous, then new accesses belonging to the flow should also be considered as anomalous. However if a flow is allowed for read operations, we have to define if write accesses should also be allowed. Similarly, if a service is allowed to read one knowledge node of an other service, if it should be allowed to read its other knowledge nodes.

The problem of these different definitions is that we cannot choose only one and use it everywhere. If we decide to the strictest possibility, if only read is allowed then write is not allowed and if a service can read one knowledge node it is not allowed to read other from the same service. However if in this particular smart space there were to be a lot of such accesses that the user would consider as normal it would lead to a lot of false positive. In the opposite case it would lead to a lot of false negative.

4.5.2.2 Sensibility of the detection

To solve this problem we introduce a global state of the system called *sensibility level*. It is similar to the system proposed in [10]. Depending on the *sensibility level* particular accesses are anomalous or not.

Sensibility levels:

- Level 0: no changes to the flow description are allowed.
- Level 1: different operations than the already seen ones are allowed.
- Level 2: a service is allowed to access knowledge nodes that belong to a service from which they are already accessing other knowledge nodes.
- Level 3: level 1 and 2
- Level 4: level 3 and if a service has already a communication relationship with a particular type of service it can access other services of the same type.
- Level 5: add a level with the rooms?
- Level 6: add periodicity here?

The list above presents the different *sensibility levels*. Level 0 is the most strict one, an access has to be exactly conform to the description of the flow. For example if only write operations are explicitly allowed, read operation are labeled anomalous. This is not the case in level 1, were all type of operations are allowed as long as the flow is allowed.

Level 3 is distinct from level 2 because situations might require the policy of level 2 without the one of level 1.

Level 4 is the most permissive one, what is allowed in level 3 is also allowed here. Moreover, if a service has already an allowed connection relationship to a service of a particular type, it is allowed to access all the service of this type. For example if a service has an allowed flow to a service of type "light", it can access all the lights in the smart space. It is very permissive, but might be needed in some situation.

4.5.3 Learning the white-list

In this section we present how the white-list is learned. The two questions that we answer in this part are: how is the white-list learned? how is the white-list adapted to changes in the Smart-Space?

4.5.3.1 In Related Work

The related work close to this module are described in section 3.3. The different work use a learning phase at the beginning of the system. All the flows observed during this phase are added to the white-list. After the end of the learning phase, any packet that does not belong to any of the learned flows is labeled as anomalous.

This method relies on two assumptions. First, it assumes that all the flows monitored during the learning phase are normal. Then, it assumes that all the normal flows of the system can be observed during the learning phase.

We cannot use the second assumption because new services can be added to the smart space during the runtime. Therefore in this section, we use an assumption similar to the first assumption: all the flows monitored at the beginning of a new service are normal.

4.5.3.2 New services

At the start of each service there is a learning phase. We allow all the accesses that happen during the learning phase. Moreover the flow is added to the white-list.

We chose a learning phase of 20 seconds. This was arbitrarily chosen and we did not look for the best solution because we did not try to optimize this approach but rather

focus on the second one. We did not implement it, but we thought that the learning phase could be prolonged if the service exhibited new activitiess.

The second particular case that needs to be handled are connections toward new services. These connection can come from services that are not in their learning phase anymore and be labeled as anomalous. For example if we have a heating service that reads all the thermometer of the smart space, if we add a new thermometer, connections from the heating service to the thermometer would be anomalous even if they should actually be normal because it is one of the thermometer of the smart space Therefore we allow connections towards new services as long as the originating service has already allowed connections to service of the same type as the new service.

4.5.3.3 User input

We need the user to improve the learning method described in the previous section. The services might change their behavior after the end of the learning phase. Then, new connections would be labeled as anomalous. We want to inform the user that these new connections happen and let him the choice to add them to the white-list or to definitely forbid them.

4.5.4 Limitations

This approach is able to detect anomalous accesses, however it has a lot of limitations. We review these limitation here.

This approach is based on the two same assumptions as the related work. First, it is assumed that all the traffic seen during the learning phase is normal. If this is not the case, then anomalous traffic would be added to the white-list. Then it assumes that all normal traffic is seen during the learning phase. If this is not the case, some of the normal accesses will be labeled as anomalous.

To solve the second problem this approach uses user interaction. This is also a limitation that we identify, the potential number of requests sent to the user is quite hight. An overload of notification might result in a user allowing everything.

Then, this approach is quite hard to extend to new anomaly criteria. Either an access belongs to a flow that is white-listed or not.

We could have solve some of these limitations with different work-around. However we solve them in our final approach, hence we did not spend time to optimize this one.

4.5.5 Conclusion

This first module allows to detect unwanted flows in real time. This is useful to detect changes in the behavior of services. This module still has two major problems that have to be solved: there is a need for user interaction and the assumption is made that all the connections at the beginning of a service are normal. (no new service detection) Moreover, this module is included in the Sphinx. This means that there is one instance of it on each KA. Therefor, each instance of the module only monitors a subset of the connections in the smart space. To handle this problem, we want to have a module that supervises all the sphinx instances, Echidna.

4.6 Communication Model

To limit the need for user interaction and the importance of the learning phase we present a new approach based on a communication model that depicts the behavior of each type of service. A particularity of the DS2OS is that we can extract the type of the service that issues a connection. This means that we can compare the behavior of the instances of this service. For example, we can compare the behavior of all the light controllers in the smart space. This is what we achieve using the communication model presented in this section.

We begin by reviewing the motivations and requirements for this approach in section 4.6.1. Then we present the concept in section 4.6.2. We show how the behavior of each service is described in section 4.6.3 and how this description allows to create a high level description of the behavior of each type of service in section 4.6.4. Finally, we explain how the communication model is used to detect anomalous accesses.

4.6.1 Motivation and requirements

The motivations to design this new approach are driven from the limitations of the first approach presented in section 4.5.4. We want to overcome these limitations.

This new approach follows the same requirement as described in the analysis. However we want our approach to be able to handle concept drift without the need for user-interaction. We also want to gain advantage of monitoring the whole site by combining the findings of the different services. We can perform a more accurate detection when having a site-view.

4.6.2 Concept

To answer these requirements, we propose our final approach.

A communication descriptor is created for each service. It describes the current behavior of this service. It is created by the Sphinx that monitors this service.

The communication descriptors of all the services of one type are merged to created a communication model for this type of service. This merge happens on two levels, first it is done on the site level i.e. a communication model describing the behavior of all the light controller in the smart space is created by Echidna. After this it is done on a global level i.e. the Echidna of each site sends the communication model it has created to the store where they are merged. This allows to have a global view of how services behave.

Finally, in order to use the communication model to detect anomalous accesses, the communication model of the store is sent to the Echidna of each site, that sends it to

the Sphinxes that host a service of the corresponding type.

This is done periodically in order to have always a current description of the behavior of each type of service. The communication models are not discarded, they are updated to keep track of how this service type behaved before.

4.6.3 Communication model

We create a communication model for each type of service. We describe in this section how the communication model is represented.

We selected site independent features to describe the communication behavior as we want to compare the behavior of the different service instances.

Partner type	"light" (.6)	"radiator" (1)	…
Number of partners	2.3 (.6)	3.9 (.99)	…
Same location	yes (.95)	no (.91)	…
Operation types	"read" (.96)	"read" (.38), "write" (.59)	…
Data type	"text" (.98)	"number" (1)	…
Approved by user	"yes" (.98)	"yes" (1)	…
Periods …	[…]	[…]	…

Table 4.3: An example communication model for a service type.

Table 4.3 gives an example communication model for a service type. The communication model is presented in the form of a table with a column for each communication partner type and a row for each feature describing this connection.

Most entries of tablee 4.3 are represented with the actual value and its probability. However behind each table entry a cluster of values is stored. The *Partner type* gives the type of the partner that is described in this column. The *number of partners* feature give the number of communication relationships to different instances of this type. For example our described type, has in average around 4 communication relationships with radiators. The *same location* gives if the partners are often located in the same room or not. *Operation type* gives the operations that are performed on the partner. *Data type* gives the type of the values exchanged. Then, *Approved by user* gives if the user approved it or not. Finally *Periods* gives the know periodicities, this feature will be discussed in section 4.7.5.

We use modified BIRCH clusters [27] to save the values. BIRCH clusters are particularly useful in data stream scenarios thanks to the additive proprity of their *cluster features* and they require low storage for representing large amounts of data points. We use the modified BIRCH clusters of the CluStream algorithm that are presented in section 2.6.2.

In addition to this table we save a *support value*, it represents the number of different instances of this μS type that were monitored to create the *communication model* and for how long they were observed.

An advantage of the communication model is that it can run with any subset of the presented features. This allows our solution to be compatible with IoT sites that do not provide all context information. It also allows to use additional features that would not be available in the DS2OS. Furthermore, it is human understandable as each value is described and it signification is given.

4.6.4 Building communication models

Our approach to learn communication models is inspired by the CluStream algorithm [26] that is presented in section 2.6.2.

A communication descriptor is created every minute for each running service. The communication descriptor is build like the communication model. It is a communication model describing only one instance; therefore, some clusters only contain one value. Table 4.4 represents an example communication descriptor. The categorical attributes, like the operation types, are transformed into vector using binarization [21].

Partner type	"light"	"radiator"	…
Number of partners	2	4	…
Same location	yes (.5)	no (1)	…
Operation types	"read" (1)	"read" (.4), "write" (.6)	…
Data type	"text" (.8)	"number" (1)	…
Approved by user	"yes" (1)	"yes" (.75)	…
Periods …	[…]	[…]	…

Table 4.4: An example communication descriptor for a service.

Every minute, Echidna merges the descriptors of each type of service to update the communication model of this type of service. To update the communication model, for each feature the clusters of the descriptors are merged with the one of the communication model. The additivity of BIRCH clusters allows to simply add the cluster features of the different clusters together [27]. As proposed in [26], all clusters that have their centers within $3 * standard\_deviation$ are merged. All other clusters remain. Note that clusters containing only one point have no standard deviation; therefore, for them, we only consider the standard deviation of the other merge candidate.

The behavior of a μS can change over time. Thereby, clusters can become stale. To notice such state, timestamps are helpful. However, to save storage space, we do not save every single timestamp. Therefore we compute the staleness of the clusters as described in section 2.6.2 and discard too stale clusters.

The same merge procedure is repeated on global level, in the store, the communication models of the different site are gathered and merged. Then they are sent back to the sites. The advantage of this merge mechanism is that every two communication models can be merged following the same low demanding procedure.

Updating the model every minute is an arbitrary choice. There is a trade of between the traffic overhead generated by too frequent updates and the detection disadvantage that is implied by too seldom updates. We chose to update the communication model every minute, and it prove to be efficient in the evaluation.

4.6.5 Access normality of an access

Once the global communication model is distributed on all Sphinxes, it is used to detect anomalous accesses. An anomaly value is computed for each access. It is composed of three parts:

The *Support of the Model* (SoM) represents how much we can rely on the values saved in a *communication model*. This depends on two factors, the number of service instances that were monitored to create it and for how long. If only few service instances were monitored or for a shot time we do not consider the values are relevant. We compute the *support of the model* using:

$$SoM = \frac{modelWeight}{maxModelWeight}$$

Where $modelWeight$ is the total number of descriptors that were merged over time to build this model. As the model is updated every minute, the number of descriptors is proportional to the number of service instances used and the monitoring duration. The support is low at the creation of a new model. This represents the fact that in this case we cannot draw meaningful conclusions yet.

We have to limit the growth of the support to enable updates to the model. Such updates reflect when service change their behavior. The limitation of the groth allows to handle concept drift. At the same time the support has to be big enough to prevent learning anomalous behavior as normal. This problem is known as the stability-plasticity dilemma [3].

We propose a solution in the form of the $maxModelWeight$. This value represents the maximum value that the $modelWeight$ can take. The higher the $maxModelWeight$ the longer it takes to adapt to changed service behaviors. The lower this value, the quicker anomalous behavior becomes part of the model. We use a $maxModelWeight$ of 2000, it is a good trade of and performed well on our training set. However, this value is a parameter to be chosen depending on the preferences of the user.

Our hierarchical model update process brings the advantage that even if an attacker manages to take control of all services of one type within one site, he will not be able to influence the definition of the normal behavior contained in the *global communication model*.

The *Support of the Relationship* (SoR) represents how relevant a communication relationship is compared to the model. A relationship that was in every descriptor used to build a model is more relevant than another that was observed only a few times. In our current approach we consider infrequent relationships as anomalous. We compute it with:

$$SoR = \frac{modelWeight}{relationshipWeight}$$

Where the $relationshipWeight$ is the number of descriptors in which this relationship was present. Here again, to allow updates, both values have a maximum of $maxModelWeight$.

The *Support of Access* represents the difference between the current access and description of the communication relationship. It is computed for each feature identified in table 4.3, using the formula:

$$SoA = \frac{dist(center, newValue)}{3 * standard\_deviation} * \frac{relationshipWeight}{centerWeight}$$

Where: $center$, $standard\_deviation$ and $centerWeight$ are the center, the standard deviation and the weight of the nearest BIRCH cluster in the communication model, and $newValue$ is the value of this feature for the access being evaluated. This value will be near or equal to zero if the new access is conform the normal behavior according to this feature due to the very small distance between the two values.

We combine these values to obtain the *Anomaly Value* as follows:

$$Anomaly\_Value = SoM * max(SoR, SoA)$$

If an access is anomalous either the connection is itself anomalous, for example a light service accessing the door lock, resulting in a high *SoR*; or the access is anomalous within the connection, for example a light service writing the movement value of a movement detection instead of reading it, resulting in one high *SoA*. Hence we take the maximum out of these two values. In both cases, the value has to be weighted with the *SoM*, as high anomaly values obtained by the comparison with a newly created communication model might come from normal accesses.

4.6.6 User inter-action

In this approach too, we can use Antigone to perform user-interaction. The approach itself does not require user-interaction to function or adapt to changes in the normal behavior. However, using user-interaction is a solution to the stability-plasticity dilemma, it can be used to prevent the model from integrating unwanted behavior as normal or accelerate the adoption of changes in the normal behavior. The user-interaction of this approach has not yet be implemented and evaluated, but we present here how it is designed.

We propose to notify the user when a new anomaly is seen but not wait for his answer to output the anomaly value. If the user allows it, the feature *Approved by user* of this communication relationship is changed to "yes".

Integrating this information to the communication model has many advantages. The statistics of the approvals of other users can be shown to the user to help him to decide. Moreover, if a new behavior has been approved by man other users in other smart spaces, we can decide that this behavior is normal without requiring the approval of all users.

4.6.7 Conclusion

To conclude, we compare the presented approach with our first approach. This new approach uses a network-level and global view to improve the detection. This results in many advantages.

In the first approach the learning relied on a learning phase. In this approach, there is a similar mechanism brought by the *SoM*. However it is quite different in the sense that only at the start of the first instances of a service type, the relevance of the model is low. When a new instance of a know service type is started, anomalous accesses can directly be detected. Moreover this approach does not rely on the two assumption of the first on because the model is updated during the runtime.

This approach does not require user-interaction to function. The approach can be enhanced to use it but it is not needed. Whereas the first approach relied in user-feedbacks to adapt.

Finally, this approach allows to add new feature to the detection very easily, which was not possible with the first approach.

However, this approach is not yet perfect and still has one limitation. The value of $maxModelWeight$ has to be chosen depending on the preferences of the user and this is not optimal.

4.7 Frequency anomaly detection

In contrast to regular Internet traffic IoT traffic is more regular [11, 53]. Therefore we include periodicity of communication as relevant part of our model. Similarly to [50,51], we use inter arrival times of communication packets in our monitor. Having the type of operation as part of our feature vector, we analyze inter-arrival per VSL operation.

This section addresses the detection of anomalies in the frequencies of flows. To do so we mine each flow to find out if there are periodicities. If so it creates a definition of each period with a confidence value. Then each new access will be compared to the learned periodicities. We use completely unsupervised learning, meaning that the goal is to detect abrupt changes in the frequency of a flow that could be the sign of a malfunction, packet injection or a DoS attack.

We begin with the requirements in section 4.7.1. Then we argue why we use inter-arrival times in section 4.7.2. We explain how we keep track of the past inter-arrival times in section 4.7.3. The periodicity mining method is explained in section 4.7.4. The integration in the communication model and the detection of anomalous frequencies is addressed in section 4.7.5.

4.7.1 Requirements

We review here the different requirements for this module.

First, it has to perform *online detection and description of the periodicities in a flow.* As the periodicities in a flow can change over time, we want the detection to adapt to these changes.

Small latencies are frequent in network communications, therefore the detection has to be *tolerant to imprecise periods.*

The periodicity mining has to be able to *detect any kind of periodicity.* For example, if three packets are send with three seconds interval every ten minutes, the flow shows a periodicity even it every packet is not send with the same period.

Moreover it should be able to use the mined periodicities for *frequency anomaly detection.*

The mining and the detection should *not consume too much resources* since there can be many flows monitored by each sphinx. Moreover it would be great if the mining could run in the background and the detection still be done in real time.

4.7.2 Inter-arrival time

The analysis of the related work (section 3.4) shows that there are two main methods used to detect periodicities within a flow, using signal analysis methods like the Fast Fourier Transform or using the inter-arrival time.

Barbosa, Sadre, and Pras [53] first tried to use the Fast Fourier Transform but conclude that it has two major drawbacks. First, the detection is not very tolerant to small latencies. Then, the signal is build using the number of packets, therefore informations about the different packets are lost. It is not possible to know if only read accesses are periodic or if all operation types are periodic. Using this analysis and their experience we decided to put the signal analysis methods on the side.

Our solution uses the inter-arrival time. On each flow, we measure the time between each packet and the packet before. We save a number of inter-arrival times in a sliding window. We mine this window to detect if there are particular inter-arrival times that happen often. This would mean that the flow is periodic because packets arrive every x seconds. Figure 4.3 shows how inter-arrival times are distributed for a periodic and for a non periodic flow. If this is the case, we create a definition of the mined frequencies and use it on newly arriving packets. In figure 4.3 two clear groups of inter-arrival times can be distinguished in the periodic flow, these are the periods that we look for in the mining process.

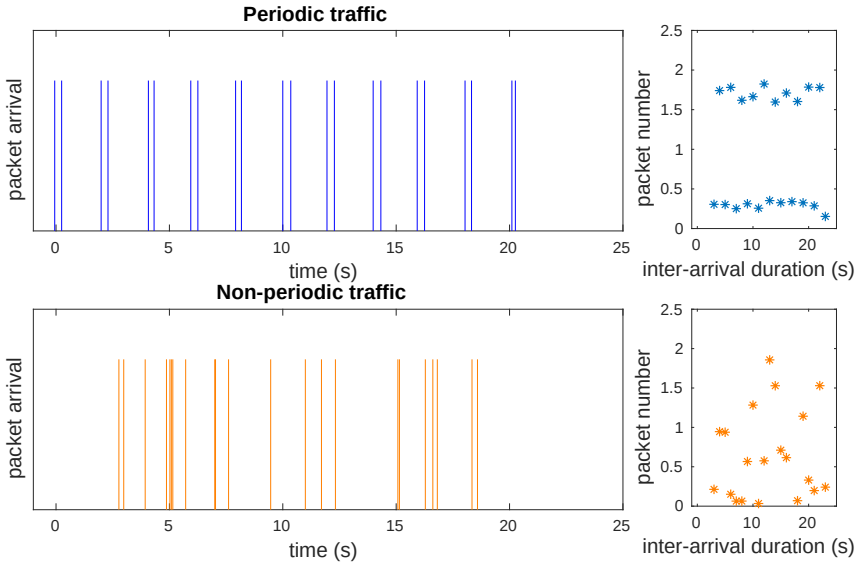


Figure 4.3: A visualization of the detection of periods using inter-arrival time.

4.7.3 Sliding window

The periodicity mining is done using the inter-arrival times between the packet of the flow. We cannot save all the inter-arrival times of all the previous packets because this could require a theoretically unlimited amount of save space. Moreover very old data might not be accurate any more. Therefore, we want to use a sliding window as described in 2.5.2.

The two major challenges linked with it are, first the choice of the size of the sliding window. And then the determination of the best data-structure for it.

4.7.3.1 Size

There are two possible criteria for the size of the sliding window. As all the timestamps in the sliding window will be used for the periodicity mining, it is a very important task to choose the size of the sliding window.

We can use the timestamps of all the packets that arrived in the last x minutes. This has the advantage of being robust against frequency changes: if a new frequency arises it has to last at least for the whole duration of the sliding window to be in the description. However, the number of packet saved is very dependent on the frequency. If we have a very high frequency we can have up to millions of packets in the sliding window which would require a lot of memory and a lot more of computation to detect pattern in it. On the other hand if the frequency is very low we might only have one packet in the window which is not very helpful to detect patterns.

The other solution is to use the inter-arrival times of the last x packets. This solution is not as frequency dependent as the one above. However, it is affected by DoS attack flooding: if a DoS attack is lunched on a flow, as the frequency of the DoS packets is very high, after a very short amount of time all the packets in the sliding window will be packets created by the DoS attack which will lead to a false definition of the normal frequency. But this problem can be solved as we explain in 4.7.5.

We use the second solution and save the last x inter-arrival times in the sliding window, because the first solution is too frequency dependent. The last question that is to be addresses is: how many inter-arrival times should be saved in the sliding window?

We choose to use the last 200 inter-arrival times. It seems to be good trade off between not too much memory consumption or data points to perform the mining and still enough to detect relevant periodicity. It empirically proved to allow a good periodicity mining and anomaly detection.

A value of storing the last 200 inter-arrival turned out to be a good compromise between having enough history and being able to react fast.

4.7.3.2 Data-structure

Now that the size and the needs are defined we need to design the best data-structure for this task. We use a ring buffer with adjustable size.

We use a ring buffer because it is the data-structure that corresponds to our need: we save inter arrival times as they come in and discard the old ones that do not fit in the time window anymore.

If we were to use the first option: the packets of the last x minutes, we would need a ring buffer that allows changes in its size. To address this we implement a custom list as explained in the implementation section 5.4.

4.7.4 Mining periodicity

In this section we discuss the method that we use to extract the different periodicities present in the flow. We use clustering methods to detect the group that can be seen in figure 4.3.

4.7.4.1 Challenges

We want the mining to be automated and to run in a fully unsupervised fashion. It is not very hard because it is a quite basic clustering problem.

However there is one major challenge, we do not know the number of clusters in advance and many clustering algorithms require this as input.

4.7.4.2 Clustering

To be able to determine the number of clusters, we use a density-based clustering method. Our algorithm uses two type of clustering algorithms, first a density-based algorithm and then k-means.

We discretize the inter-arrival times into a grid of p intervals of similar length. Each interval contains the number of data points that fall into it.The first interval begins at the minimum inter-arrival duration and the last ends at the maximum inter-arrival duration. Once the discretization is done, we divide the number of data points in each grid by the window size and multiply them by 100. This allows us to be able to compare the densities independently of the window size.

Then, we use the density of the points within this grid to build clusters. A cluster is built on each interval that contain at least as many points as the density threshold τ.

If neighboring intervals in the grid have a density higher than τ they are identified as only one cluster. For each cluster a *cluster center* is placed within the intervals.

The difficult part of this method is known to be the determination of p and τ [21], as it determines which clusters are found. However, we can determine these parameters using what we consider as periodic and what we do not consider as periodic. We use a p of 100, as this allows to have a enough definition to detect different clusters, but normally does not regroup different clusters. We use a $\tau = 5$, this corresponds to a cluster containing $\frac{1}{6}$ of the point spread over 3 intervals. We consider such clusters to be the limit of a periodic traffic. The chosen values have empirically proven to be efficient.

Once all the clusters are identified, we run one iteration of k-means to center the *cluster center* and get the support of each cluster.

Finally, the algorithm outputs a list of clusters. Here again, we use the CluStream cluster representation.

The main advantage of this clustering method is that we can easily find out the number of clusters. If the flow is not periodic no cluster are found. Otherwise the centers of the clusters correspond to the periods.

4.7.5 Access normality of new timestamps

Once the periodicities of a flow are found, we want to use them to detect frequency anomalies.

We enhance the communication descriptors and communication model with this new feature. Resulting in an additional line in the table containing the list of the cluster representation of the mined frequencies.

This allows us assess the normality of new accesses by also computing the anomaly value of this new feature. The computation is done exactly in the same way as for the other features, as described in section 4.6.5. In this case, the newValue is the new inter arrival time. This way the frequency detection is directly included in the anomaly detection, showing how easily the communication model can be enhanced with new features. However, there is one type of attack that cannot always be detected using this method: denial of service attacks.

4.7.5.1 Denial of Service detection

The approach presented is able to detect single anomalous accesses: point anomalies, as described in section 2.4.1.

There is one attack scenario that remains hidden to the previous method: Denial of Service (DoS) attacks on allowed relationships that have one period around zero. In a DoS a high number of accesses happens with high frequency. If for such a communication relationship a periodicity with a *very low value* has been learned, the single packets of the DoS attack are seen as normal. This is a *group anomaly*, as described in section 2.4.1.

To detect such group anomalies we compare the list of periodicity clusters on the descriptor with the one of the communication model to obtain an anomaly value:

$$Group\_Anomaly\_Value = \sum_{cluster_a \in model} \frac{dist(center_a, center_b) * (weight_a + weight_b)}{3 * standard\_deviation_a}$$

With $cluster_b$ the nearest cluster to $cluster_a$ in the descriptor frequency list. In the case of a DoS attack a cluster with a very high support will form around 1 or 0. This will cause this $Group\_Anomaly\_Value$ to be very high, even if there is also such a cluster in the communication model. System inherently the first packets of an anomaly will remain undetected. However, as the goal of a DoS attack is to overload the reciever with more packets than it can handle, if only a few packets arrive at the receiver the attack will not be successful.

4.7.6 Conclusion

The presented approach allows to split the traffic of one service and split it in the different communication relationships. This allows to detect periodicities in a more fine grained way, as shown in figure 4.4. Moreover, it can detect periodicities in when monitoring all the out coming traffic would not allow it. Other approaches, like the one inspired from the signal analysis, could be investigated.

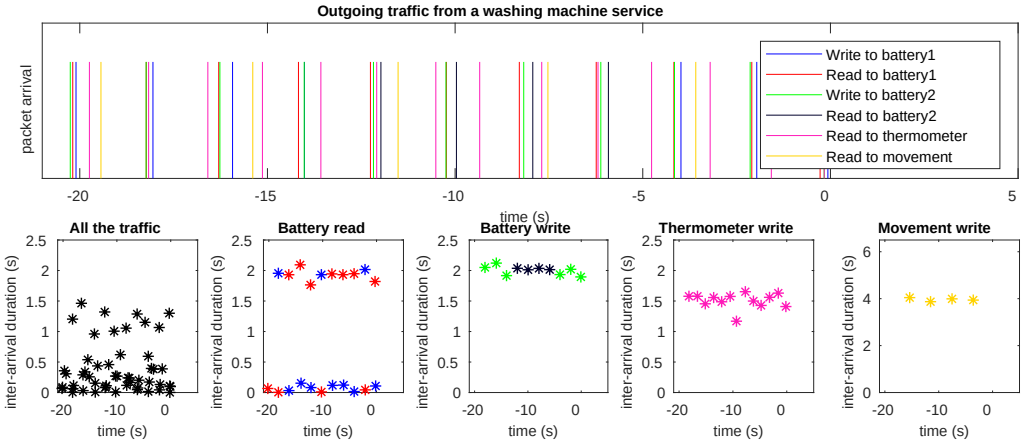


Figure 4.4: Decomposition of the outgoing accesses of a node into the communication relationships.

Chapter 5

Implementation

In this chapter we review some particular points of the implementation.

5.1 Connection to the VSL

In this section we explain where the detection mechanism is linked to the VSL and why.

The implementation of the routing of accesses in the VSL uses a class on the KA called RequestRouter in the package ka-wiring. All the accesses passing through a KA transit through this class. We intercept the accesses here.

We create a class AccessLogger that is called each time an access transits through the KA. Their is only one active instance of the AccessLogger on each KA. This class is used to preprocess the accesses and create a definition of the access that is send to the Sphinx in order to detect if the access is anomalous or not. The AccessLogger waits for the output of the detection to allow the routing of the packet in the RequestRouter.

This architecture also allows us to use the exact same program of the Sphinx to test the detection on datasets, as the description of the access given by the AccessLogger is the one that is saved in datasets.

5.2 Connection between Sphinxes and Echidna

The connection between the Sphinx and the Echidna is made via the VSL.

The Sphinxes publish in the form of VSL context nodes all the information that Echidna might need. This includes, a vertex list and an edge list that correspond to the internal flow-list of the Sphinx, as well as a descriptor list, that contains the descriptors of all the service types monitored by the Sphinx.

Echidna also publishes in form of VSL context nodes all the information that the Sphinx can need. In our final approach, Echidna publishes the list of all the communication models present in the site.

Echidna looks for services of the type "sphinx" every two seconds. When it finds one, with which it is not yet connected, it reads its context nodes and subscribe to changes. Then the Sphinx subscribes to changes of the context nodes of the Echidna. This way the other side is always aware of changes, and the VSL handles the communication.

The advantage of this approach is that it can be easily extended by adding additional context nodes if needed.

5.3 Hash-list

To implement the hash-list, we took the implementation of the hash table from the java.util package.

We added an ArrayList containing the keys of all the entries currently saved in the hash table. When an entry is added to the hash table, its key is added to the ArrayList; and, when an entry is removed from the hash-table, its ke is removed from the list.

We enhanced the hash-table with an iterator allowing to iterate trough all entries of the table using the ArrayList.

5.4 Sliding window

The sliding window is a ring buffer with changeable size.

We took the implementation of the ArrayList from the java.util package and enhanced it.

We added the variables $maxWindowSize$, corresponding to the size of the sliding window, and $currentHead$ corresponding to the place of the newest element in the array containing all the elements. Using them, we rewrote the method used to add new data points. New data points are to be added at the position after the $currentHead$ and if $currentHead$ corresponds to the last position in the array, it is saved at the first position of the array.

When changing the size of the buffer, there are two possibilities, either the window size is being reduced and then only the most recent data points are to be kept; or the window is being lengthened and the new empty space have to be the first to be used for new data points. This is implemented in the "setWindowSize(int newSize)" method.

Chapter 6

Evaluation

We evaluate here the solution we propose. We begin by describing the datasets that are used for the evaluation in section 6.1. Then we measure the required resources 6.2. After this, we evaluate the periodicity mining and the anomaly detection in sections 6.3 and 6.4.

6.1 Datasets

For the evaluation we captured our own datasets since to the best of our knowledge there is no public IoT network trace. We monitored connections between 8 different types of services: Light controllers, movement sensor, thermostat, solar battery, washing machine, door lock and user smart phone. We captured the traffic of four different simulated IoT sites over 24 hours.

6.1.1 Datasets description

We will discuss here why we want to create datasets and which ones.

As explained in section 2.7.1, we want to have labeled datasets that can be streamed in order to run and assess the detection. The datasets will represent a set of connections as they would be monitored by the detection system once it is deployed.

We create four datasets: two to optimize the parameters and then access the frequency anomaly detection alone as well as two to optimize the parameters and then evaluate the final approach.

The dataset purely for the frequency anomaly detection are the capture of only one flow on which there are two periodic cycles. We add two type of anomalies: random packets

that do not conform the periodicities, these could be caused by injection attacks; and DoS attacks.

The datasets for the final approach are more complex and contain many different type of attacks. These datasets were created by simulating four IoT-sites during 24h. We describe more in detail the setup in section 6.1.2. We added a wide range of attack scenarios that are shown in table 6.1. These attacks are inspired from the attack scenarios presented in [9].

Type	Number of packets	Feature used	Description
Network scan	1559	type & number	A μS accesses a range of other μSs.
Spying	532	type & operation	A μS reads values a few other μSs.
Malicious control	889	type & operation	A μS tries to take control over another.
Malicious operation	805	operation type	A μS performs another operation as it should.
Denial of Service	5780	frequency	A μS performs accesses with a very high frequency.
Data types probing	342	data type	A μS writes anomalous data types.
Wrong set up	120	location	A μS accesses a μS in the wrong room.
Total	10027		Corresponds to 3% of the dataset.

Table 6.1: Attack scenarios in the dataset to test the final approach, inspired by [9].

As we stand for the reproducibility of our research we published the two datasets online at `www.kaggle.com/francoisxa/ds2ostraffictraces`. There, we also published the exact description of the different attacks, their start time, and their duration.

6.1.2 Simulating a smart space

To create the dataset we captured the traffic of four simulated sites. Table 6.2 shows which devices were deployed in which sites. In each room there was one light controller, one movement sensor, and one thermometer.

To simulate the dynamic behavior of a smart space, un update of all the light sensors is monitored at around 11 am. Moreover during the capture some devices were added.

Site	Rooms	Washing Machine	Battery	Thermostat	Smart door	Smart phone
House	6	1	3	1	1	1
2 rooms flat	2	1	1	1	1	1
3 rooms flat	3	1	2	1	1	1
Office	10	0	0	1	2	1
Total	21	3	6	4	5	4

Table 6.2: The distribution of the services over the four sites.

6.2 System requirements

In this section we evaluate the speed, the scalability and the resource consumption of our final approach.

Nota bene: our approach is implemented in Java; therefore we use the size of the variables in Java. Ints require 4 bytes, booleans require 1 byte and longs as well as doubles require 8 bytes.

6.2.1 Within one Sphinx

The anomaly value of each access is computed before it is allowed to be routed to its destination; therefore the short duration of this computation is critical. We designed the hash-list and the other mechanism to have an overall computation complexity of $\mathcal{O}(1)$ and to scale well with the number of services.

Figure 6.1 shows the latency added by detection for different number of services. As we can see, the latency is quite low. The high number of outliers in the figure are explained by the very high number of measurements. We can see that the detection time does not grow with the number of services.

6.2.2 Traffic overhead

Our approach requires to send the descriptors to Echidna and the communication model back to the Sphinxes. We measure this traffic overhead and compare it with the overhead that there would be if all the detection were to take place on Echidna.

Sending all the traffic log to Echidna would require on average 169,3 bytes per access. Whereas our approach only requires to send one descriptor to Echidna every minute and receive a communication model. On our dataset the average descriptor size is 96 bytes. The descriptor is of smaller size, but its other advantage is that the bandwidth needed does not increase if the monitored traffic increases. This solution allows a completely scalable detection.

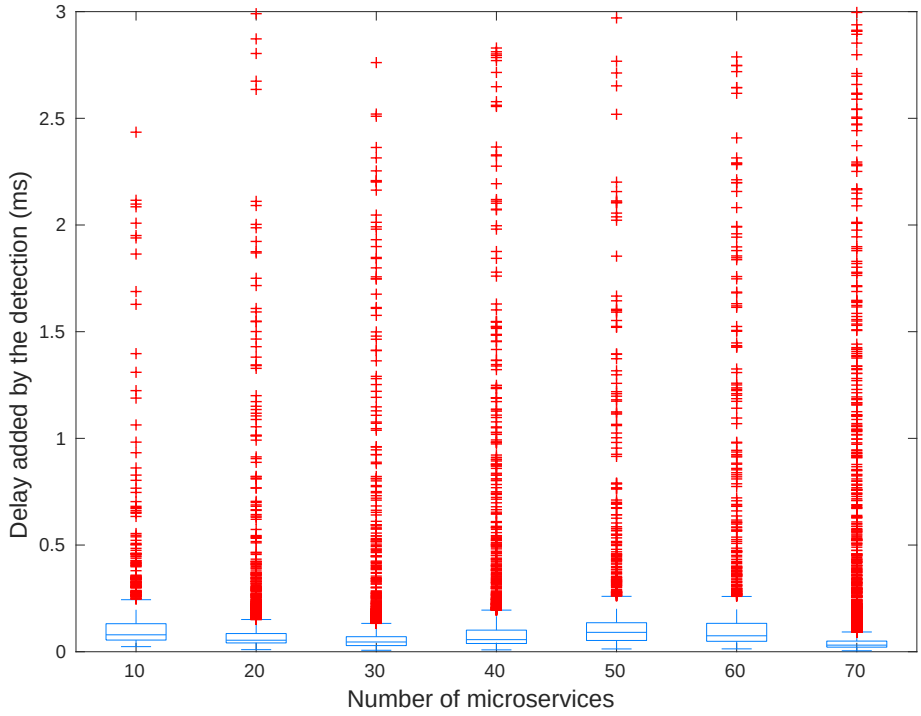


Figure 6.1: The latency added by the detection mechanism.

6.2.3 Memory requirement

The cluster representation we use here, allows to minimize the memory consumption as well as the computation complexity of the clustering.

For d-dimensional data points the modified BIRCH clusters we use, only require $(2*d+3)$ saved values independently of the number of points in the cluster. Whereas a list keeping track of all the k points in the cluster would require $k * (d + 1)$ saved values: one value for per dimension plus the timestamp for each data point.

To illustrate the difference, we compare the memory needed for a cluster of 1000 data points composed of two doubles. A normal cluster representation keeping track of all the points would require 24000 bytes, whereas the modified BIRCH clusters we use require 52 bytes.

6.2.4 Computational complexity

The modified BIRCH allow our approach to be efficient also regarding storage CPU usage. We compare the computational complexity of adding points in k-means clustering and in our approach.

Adding a new point to the dataset of size n and dimensionality d requires to iterate

though the k existing clusters and then merge it to the corresponding cluster. The complexity of the first operation is the same for k-means and our algorithm: $\mathcal{O}(k)$. However updating the cluster position in k-means requires $\mathcal{O}(n * d)$. Whereas it only requires $\mathcal{O}(d)$ with BIRCH clusters. [21]

6.2.5 Conclusion

IoT often operates on computing nodes with limited resources, therefore detection systems should require as little resources as possible even if it has to run in real-time.

The outcome of the previous sections shows that the effort we put into the limitation of the resource usage allows our approach to be adapted to resource limited IoT-devices.

6.3 Periodicity mining

We first evaluate the detection of frequency anomaly detection alone.

In our dataset we obtain a detection rate of 93% and false positive rate of 0.3% for the detection of single anomalous packets in the flow.

With the DoS attacks in the dataset we achieve a detection rate of 95%, an accuracy of 97% and a false positive rate of 1%. This is due to the few packets that we do not achieve to detect at the beginning of the DoS attack.

6.4 Anomaly detection

We evaluate in this section the anomaly detection of the final approach.

On the test dataset we achieve a detection accuracy of 99% and a false positive rate of 0.2%. We obtain an overall detection rate of 96.3%. It has to be mentioned that we start without learned communication models. Therefore some anomalous packets in the beginning go undetected. The measured detection rate suffers a bit from this test situation. However in real world scenarios, the communication models can directly be taken from the store, they will have been learned on other sites.

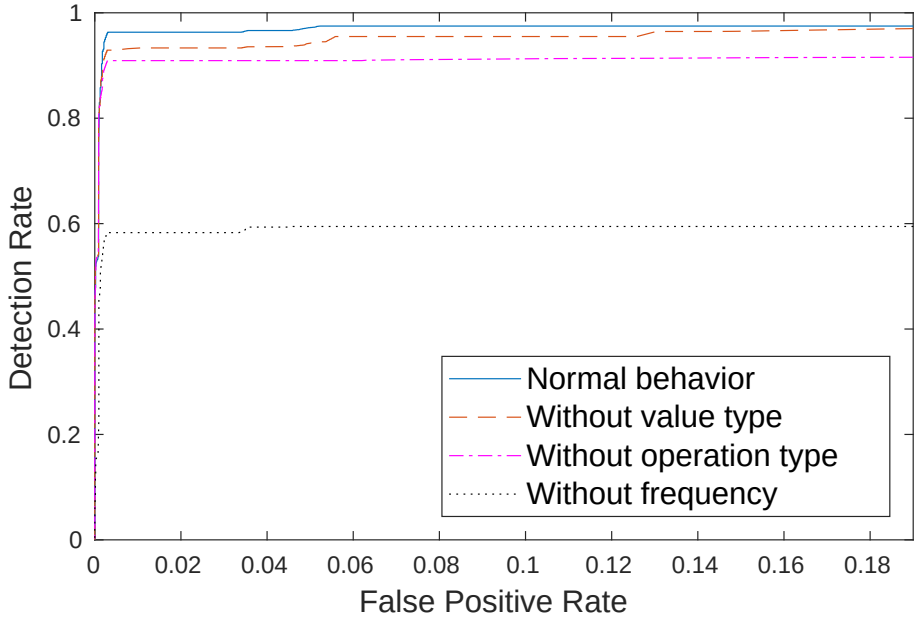


Figure 6.2: Comparison of the ROC curves with different features.

Figure 6.2 shows the performance of the detection by comparing the detection rate and the false positive rate at different thresholds. The figure also shows the influence of the different features on the detection. The performance of the detection is lower when some features are ignored, however the system is still able to detect anomalies. This shows the positive effect of our design of the communication model, it can be used with different features depending on the features available in the IoT environment.

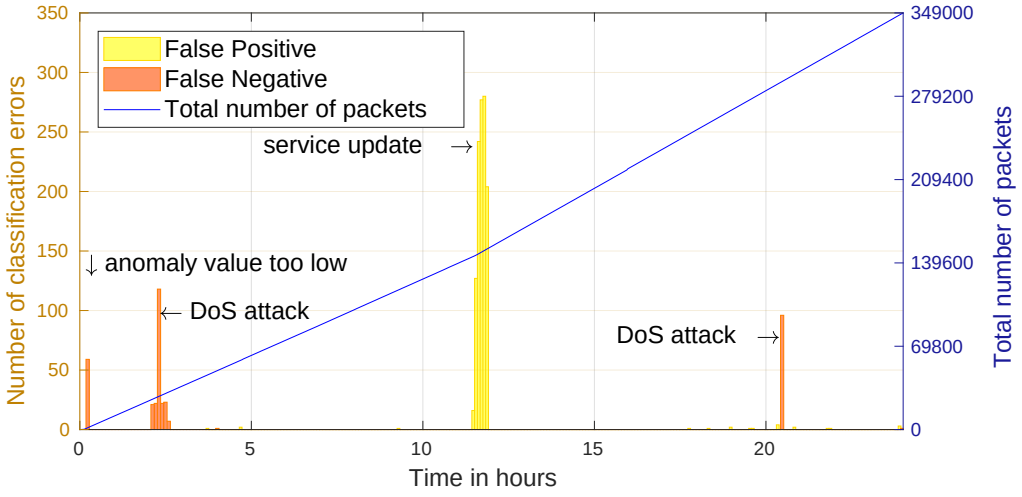


Figure 6.3: Classification errors over the day of capture.

Figure 6.3 shows the classification errors over the day of capture in our dataset. During

the capture of the dataset the light controller services were updated. Directly after the update, the new accesses are labeled as anomalous which generate a peak of false positives that can be seen at 11 hours. However the model was able to learn this new normal behavior, which allowed a correct classification again. This shows that the definition of the normal behavior can be changed to include changes. This is the prove that our approach handle concept drift.

The concern could be that the model would learn too easily and add anomalous behavior to the definition of the normal behavior. However, we did not observe this. Moreover, the anomalous behavior would need to the same on all the instances of all sites in order to be added to the normal behavior.

The first burst of false negatives is due to the model that were not relevant enough at the beginning of the day of capture. This would be different with learnt models. To prove this, this attack is present again at the end of the dataset. There, it is correctly labeled as anomalous. Thereby we show that the model need to gather enough data in oder to detect anomalies. However, once the model has gathered enough data, the anomalies are detected.

The second and the third small bursts of false negatives correspond to few accesses that are not detected correctly at the beginning of DoS attacks. However, these accesses represent at most 1% of the attack. As discussed in section 4.7.5.1, this is due to the fact that the communication relationships have one cluster around zero and need to detect the group anomaly. However, there is one more DoS attack in the dataset, and it is perfectly detected. This is because there is no such low cluster center.

In comparison [37] obtain an average accuracy of 98%, a detection rate of 96% and a false positive rate of 6%. We obtain better accuracy and false positive rate. However, it is hard to compare the performance of two detection that were not tested on the same dataset. Sadly, they did not publish their dataset. So we could not test our algorithm on it.

6.5 Demo

We present a demo of our approach at the IEEE/IFIP Network Operations and Management Symposium conference in Taipei on the 24/04/2018. [7]

The demo is composed of a smart door lock, a washing machine and two batteries, as it can be seen in figure 6.4. We show how our approach allows to monitor the connections in the smart space and to block malicious accesses from the washing machine to the door lock.

We show two scenarios to illustrate the functioning of the system. In the first scenario, a malicious update is done on the washing machine and it tries to open the door. When

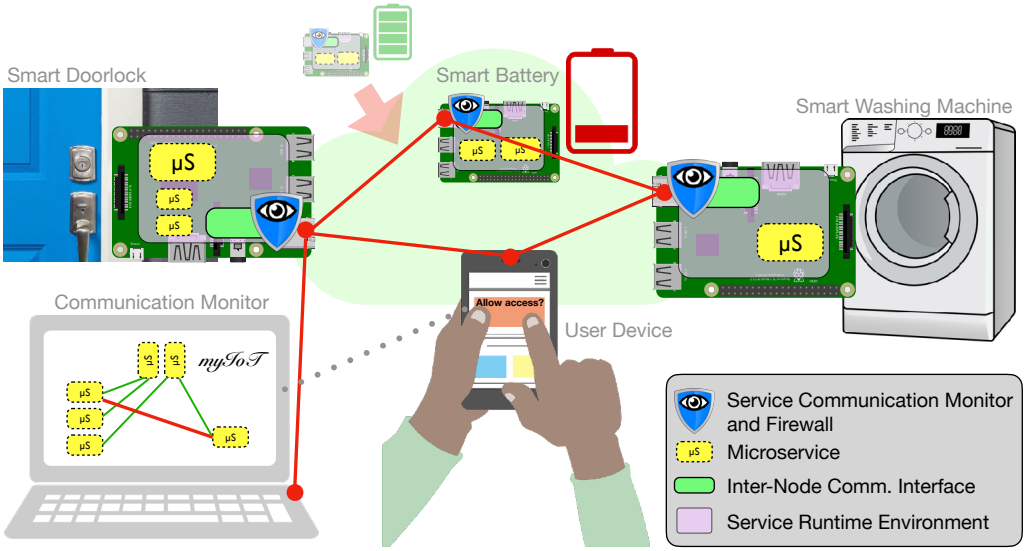


Figure 6.4: The setup of the demo. [7]

the system is not working, the door is opened by the washing machine. However, when the system is working, the access is blocked. Moreover, the user is notified and the corresponding connection is shown in red on the visualization of the connections in the smart space.

In the second scenario, a battery is added to the network, allowing the washing machine to run constantly. In this scenario, we show how the addition of new services is handled by the system.

This demo presents the detection system, and shows how the user can be kept in the loop.

Chapter 7

Conclusion

The goal of this work is to answer the question:

How can anomalous connections between the services of a smart-space be detected the most accurately in the shortest time?

We answered this question by, first analyzing the particularities of IoT traffic, then using this particularities to design an adapted detection algorithm. We also analyzed the characteristics required for a detection system in a smart space.

Using this analysis, we propose an approach that allows to detect anomalies with a very low latency. It is quite accurate and can adapt to changes in the normal behavior of the smart space, which is, as we have seen, a very relevant aspect of IoT environments. Moreover it is scalable, which allows its deployment in any size of smart spaces. Finally, this solution allows to keep the user in the loop by providing a human understandable model of the connections.

7.1 Future work

However, there might be better solutions or approaches to detect anomalous connections between the services of a smart-space. We give some ideas of future work that could lead to better solutions.

First, the two additional modules that we mentioned in the analysis: value anomaly detection and access patterns anomaly detection could be designed and implemented. We did not have time to look towards these directions in this work, however, the current approach allows easily the addition of such modules.

Then, even if the user interaction for the final approach is designed, it is yet to be tested with real users. This might lead to a better design.

Finally, we could think of an anomaly mining process on Echidna that would detect frequent patterns in the anomalies occurring, like if there is a KA in which many anomalies take place.

Bibliography

[1] S. Liebald, "Caching Content in Smart Spaces," Master's thesis, Technical University of Munich, Germany, 2016.

[2] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, p. 15, 2009.

[3] T. R. Hoens, R. Polikar, and N. V. Chawla, "Learning from streaming data with concept drift and imbalance: an overview," *Progress in Artificial Intelligence*, vol. 1, no. 1, pp. 89–101, 2012.

[4] V. Jakkula, D. J. Cook *et al.*, "Anomaly detection using temporal data mining in a smart home environment," *Methods of information in medicine*, vol. 47, no. 1, pp. 70–75, 2008.

[5] B. AsSadhan and J. M. Moura, "An efficient method to detect periodic behavior in botnet traffic by analyzing control plane traffic," *Journal of advanced research*, vol. 5, no. 4, pp. 435–448, 2014.

[6] R. R. R. Barbosa, R. Sadre, and A. Pras, "Towards periodicity based anomaly detection in scada networks," in *Emerging Technologies & Factory Automation (ETFA), 2012 IEEE 17th Conference on*. IEEE, 2012, pp. 1–4.

[7] M.-O. Pahl, F.-X. Aubet, and S. Liebald, "Graph-based iot microservice security," p. 2, 02 2018.

[8] T. H. Cormen, *Introduction to algorithms*. MIT press, 2009.

[9] E. Fernandes, J. Jung, and A. Prakash, "Security Analysis of Emerging Smart Home Applications." *IEEE Symposium on Security and Privacy*, 2016.

[10] R. Neisse, G. Steri, I. N. Fovino, and G. Baldini, "Seckit: a model-based security toolkit for the internet of things," *Computers & Security*, vol. 54, pp. 60–76, 2015.

[11] N. DeMarinis and R. Fonseca, "Toward Usable Network Traffic Policies for IoT Devices in Consumer Networks." *IoT S&P@CCS*, 2017.

[12] C. Shearer, "The crisp-dm model: the new blueprint for data mining," *Journal of data warehousing*, vol. 5, no. 4, pp. 13–22, 2000.

[13] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 2, pp. 1153–1176, 2016.

[14] P. Marc-Oliver and D. Lorenzo, "Automatic security management for distributed microservices on unattended iot nodes," *TBD*.

[15] M.-O. Pahl, "Distributed Smart Space Orchestration," Ph.D. dissertation, Technische Universität München, München, Jun. 2014.

[16] A. Cavoukian, "*Privacy by Design*: Leadership, Methods, and Results." *European Data Protection*, pp. 175–202, 2013.

[17] M.-O. Pahl, G. Carle, and G. Klinker, "Distributed smart space orchestration," in *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP*. IEEE, 2016, pp. 979–984.

[18] C.-M. Cheng, H. Kung, and K.-S. Tan, "Use of spectral analysis in defense against dos attacks," in *Global Telecommunications Conference, 2002. GLOBECOM'02. IEEE*, vol. 3. IEEE, 2002, pp. 2143–2148.

[19] J. Ye, G. Stevenson, and S. Dobson, "Fault detection for binary sensors in smart home environments," in *Pervasive Computing and Communications (PerCom), 2015 IEEE International Conference on*. IEEE, 2015, pp. 20–28.

[20] K. Scarfone and P. Mell, "Guide to intrusion detection and prevention systems (idps)," *NIST special publication*, vol. 800, no. 2007, p. 94, 2007.

[21] C. C. Aggarwal, *Data mining: the textbook*. Springer, 2015.

[22] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, and E. Vázquez, "Anomaly-based network intrusion detection: Techniques, systems and challenges," *computers & security*, vol. 28, no. 1-2, pp. 18–28, 2009.

[23] S. A. Boyer, *SCADA: supervisory control and data acquisition*. International Society of Automation, 2009.

[24] R. R. R. Barbosa, "Anomaly detection in scada systems-a network based approach," Ph.D. dissertation, Centre for Telematics and Information Technology, University of Twente, 2014.

[25] R. Kohavi and F. Provost, "Glossary of terms," *Machine Learning*, vol. 30, no. 2-3, pp. 271–274, 1998.

[26] C. C. Aggarwal, S. Y. Philip, J. Han, and J. Wang, "-a framework for clustering evolving data streams," in *Proceedings 2003 VLDB Conference*. Elsevier, 2003, pp. 81–92.

[27] T. Zhang, R. Ramakrishnan, and M. Livny, "Birch: an efficient data clustering method for very large databases," in *ACM Sigmod Record*, vol. 25, no. 2. ACM, 1996, pp. 103–114.

[28] M. Mantere, M. Sailio, and S. Noponen, "Network traffic features for anomaly detection in specific industrial control system network," *Future Internet*, vol. 5, no. 4, pp. 460–473, 2013.

[29] F. Amiri, M. R. Yousefi, C. Lucas, A. Shakery, and N. Yazdani, "Mutual information-based feature selection for intrusion detection systems," *Journal of Network and Computer Applications*, vol. 34, no. 4, pp. 1184–1199, 2011.

[30] J. Rajahalme, A. Conta, B. Carpenter, and S. Deering, "Ipv6 flow label specification," Internet Requests for Comments, RFC Editor, RFC 3697, March 2004.

[31] "Netflow," https://pliki.ip-sa.pl/wiki/Wiki.jsp?page=NetFlow.

[32] B. Li, J. Springer, G. Bebis, and M. H. Gunes, "A survey of network flow applications," *Journal of Network and Computer Applications*, vol. 36, no. 2, pp. 567–581, 2013.

[33] R. Neisse, I. N. Fovino, G. Baldini, V. Stavroulaki, P. Vlacheas, and R. Giaffreda, "A model-based security toolkit for the internet of things," in *Availability, Reliability and Security (ARES), 2014 Ninth International Conference on.* IEEE, 2014, pp. 78–87.

[34] C. Sarkar, A. U. N. SN, R. V. Prasad, A. Rahim, R. Neisse, and G. Baldini, "Diat: A scalable distributed architecture for iot," *IEEE Internet of Things journal*, vol. 2, no. 3, pp. 230–239, 2015.

[35] Y. Meidan, M. Bohadana, A. Shabtai, J. D. Guarnizo, M. Ochoa, N. O. Tippenhauer, and Y. Elovici, "Profiliot: a machine learning approach for iot device identification based on network traffic analysis," in *Proceedings of the Symposium on Applied Computing*. ACM, 2017, pp. 506–509.

[36] Y. Meidan, M. Bohadana, A. Shabtai, M. Ochoa, N. O. Tippenhauer, J. D. Guarnizo, and Y. Elovici, "Detection of unauthorized iot devices using machine learning techniques," *arXiv preprint arXiv:1709.04647*, 2017.

[37] I. Hafeez, A. Y. Ding, M. Antikainen, and S. Tarkoma, "Toward secure edge networks: Taming device-to-device (d2d) communication in iot," *arXiv preprint arXiv:1712.05958*, 2017.

[38] D. J. Cook, M. Youngblood, E. O. Heierman, K. Gopalratnam, S. Rao, A. Litvin, and F. Khawaja, "Mavhome: An agent-based smart home," in *Pervasive Computing and Communications, 2003.(PerCom 2003). Proceedings of the First IEEE International Conference on.* IEEE, 2003, pp. 521–524.

[39] C. Kidd, R. Orr, G. Abowd, C. Atkeson, I. Essa, B. MacIntyre, E. Mynatt, T. Starner, and W. Newstetter, "The aware home: A living laboratory for ubiquitous comput-

ing research," *Cooperative buildings. Integrating information, organizations, and architecture*, pp. 191–198, 1999.

[40] S. Helal, W. Mann, H. El-Zabadani, J. King, Y. Kaddoura, and E. Jansen, "The gator tech smart house: A programmable pervasive space," *Computer*, vol. 38, no. 3, pp. 50–60, 2005.

[41] S. T. M. Bourobou and Y. Yoo, "User activity recognition in smart homes using pattern clustering applied to temporal ann algorithm," *Sensors*, vol. 15, no. 5, pp. 11 953–11 971, 2015.

[42] B. Das, D. J. Cook, N. C. Krishnan, and M. Schmitter-Edgecombe, "One-class classification-based real-time activity error detection in smart homes," *IEEE journal of selected topics in signal processing*, vol. 10, no. 5, pp. 914–923, 2016.

[43] M. Sheikhan and Z. Jadidi, "Flow-based anomaly detection in high-speed links using modified gsa-optimized neural network," *Neural Computing and Applications*, vol. 24, no. 3-4, pp. 599–611, 2014.

[44] P. Winter, E. Hermann, and M. Zeilinger, "Inductive intrusion detection in flow-based network data using one-class support vector machines," in *New Technologies, Mobility and Security (NTMS), 2011 4th IFIP International Conference on*. IEEE, 2011, pp. 1–5.

[45] R. R. R. Barbosa, R. Sadre, and A. Pras, "Flow whitelisting in scada networks," *International journal of critical infrastructure protection*, vol. 6, no. 3, pp. 150–158, 2013.

[46] M. Leef and R. Addanki, "Traffic classification for flow whitelisting."

[47] A. Lemay, J. Rochon, and J. M. Fernandez, "A practical flow white list approach for scada systems," in *Proceedings of the 4th International Symposium for ICS & SCADA Cyber Security Research 2016*. BCS Learning & Development Ltd., 2016, pp. 1–4.

[48] D. Kang, B. Kim, J. Na, and K. Jhang, "Whitelists based multiple filtering techniques in scada sensor networks," *Journal of Applied Mathematics*, vol. 2014, 2014.

[49] P. Barford, J. Kline, D. Plonka, and A. Ron, "A signal analysis of network traffic anomalies," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*. ACM, 2002, pp. 71–82.

[50] N. Hubballi and D. Goyal, "Flowsummary: Summarizing network flows for communication periodicity detection," in *International Conference on Pattern Recognition and Machine Intelligence*. Springer, 2013, pp. 695–700.

[51] L. Bilge, D. Balzarotti, W. Robertson, E. Kirda, and C. Kruegel, "Disclosure: detecting botnet command and control servers through large-scale netflow analysis," in

Proceedings of the 28th Annual Computer Security Applications Conference. ACM, 2012, pp. 129–138.

[52] N. Goldenberg and A. Wool, "Accurate modeling of modbus/tcp for intrusion detection in scada systems," *International Journal of Critical Infrastructure Protection*, vol. 6, no. 2, pp. 63–75, 2013.

[53] R. R. R. Barbosa, R. Sadre, and A. Pras, "Exploiting traffic periodicity in industrial control networks," *International journal of critical infrastructure protection*, vol. 13, pp. 52–62, 2016.

[54] L. Edmunds, *The Sphinx in the Oedipus legend*. Hain, 1981, no. 127.

[55] M. A. Kolosovskiy, "Data structure for representing a graph: combination of linked list and hash table," *arXiv preprint arXiv:0908.3089*, 2009.

[56] T. Hésiode, "Les travaux et les jours, le bouclier, texte établi et traduit par p," *Mazon, Paris, Les Belles Lettres*, p. 58, 1928.

[57] F.-X. Aubet, M.-O. Pahl, S. Liebald, and M. R. Norouzian, "Graph-based anomaly detection for iot microservices," p. 2, 03 2018.