Università degli Studi di Ferrara

Corso di Laurea in Informatica

Data diodes for secure Industry 4.0 networking: A railway environment monitoring use case

Relatore
Prof. Carlo Giannelli

Laureando
Giuseppe Cervone

Correlatore
Ing. Roberto Giansante

Anno Accademico 2020/2021

Index

Riassunto

In questo lavoro di tesi si andranno a discutere i concetti di Industria 4.0, Industrial Internet of Things e Cybersecurity. Verranno utilizzate le definizioni di questi concetti per descrivere il flusso lavorativo della sede di Bologna di Alstom, compagnia per cui è stato svolto il lavoro di tirocinio esposto in questa tesi. Alstom è un'azienda multinazionale che si occupa di produzione e di monitoraggio, sia delle vie ferroviarie, che dei veicoli che viaggiano su di esse. In particolare, la sede di Bologna viene considerata un centro di eccellenza per quello che è il monitoraggio.

Si andrà quindi a discutere del pensiero e della struttura della soluzione studiata al fine di risolvere il problema posto da Alstom, dei singoli componenti che ne fanno parte e di come è possibile ricreare la suddetta soluzione in caso si volesse testarne le funzionalità e, infine, si descriverà anche degli eventuali miglioramenti identificati che potrebbero essere applicati in futuro.

La soluzione studiata, in dettaglio, si pone come obiettivo quello di tentare di replicare il funzionamento di un diodo dati a basso costo. Un diodo dati è un dispositivo di uso comune nel settore della cyber security, che permette la comunicazione unidirezionale di dati da una rete protetta verso una rete pubblica. Questo dispositivo offre quindi l'opportunità a chi lo utilizza, di ottenere informazioni sui dati presenti in una rete privata, senza la necessità di instaurare un canale di comunicazione diretto con essa. Il diodo dati in questo caso verrà simulato grazie alla comunicazione seriale tramite lo standard RS-232, comunicazione che viene resa unidirezionale tagliando il terminale responsabile per l'invio di dati verso il lato privato della comunicazione. Il cavo seriale in questione verrà connesso a due microcomputer, in dettaglio due Raspberry Pi 3. Questi ultimi saranno responsabili della sincronizzazione nella comunicazione e di rendere le informazioni comunicate facilmente da reperibili. Questo processo verrà svolto tramite l'ausilio di Zabbix, un software di monitoraggio per reti che offre anche la possibilità all'utente di simulare un ambiente con dati provenienti da altri fonti. Saranno quindi poste in esecuzione due istanze di Zabbix, la prima che sarà dislocata sulla Raspberry identificata come "privata", e che catturerà i dati dai dispositivi nella rete privata, e l'altra che andrà ad essere collocata sulla Raspberry definita "pubblica", la quale riceve i dati tramite

comunicazione unidirezionale. Questo scenario andrà a creare sulla seconda instanza di Zabbix, menzionata in precedenza, una replica di quello che succede realmente sulla rete privata e protetta.

Introduction

Industry 4.0, cybersecurity, Internet Of Things, these are all terms that 30 years ago, when the idea of “Internet” first came into our lives, would have been considered gibberish, but that now commonly used in today’s society, but how did they come to be? What are they exactly? And how can they improve our lives? These and many other questions are what I will try to tackle with my thesis work, looking at their advent, the modern perception of them, how these concepts are changing the industrial space, and how it is possible to build something by putting all these concepts together to make someone’s life a little easier.

I will do so by dissecting in the first chapter of this thesis the concept of Industry 4.0 and how it’s advent led to Internet of Things devices becoming ever more prevalent in our day to day use. This will also help us understand why the idea of cybersecurity is not to be ignored. Then in chapter two, I will look at the thought process of a multinational company, trying to understand their workflow, and how the researching of new solutions to improve their services comes to fruition. These new solutions obviously are compliant with the concepts mentioned earlier. With the information gathered about the company, and with full knowledge of the new possibilities given to us by the enabling technologies of Industrial IoT, I will dissert in the third chapter how a IIoT solution for a multinational is built. In my particular case it will be a solution with a microprocessor, with interconnectivity between devices, and with the possibility of monitoring devices under a network.

Chapter 1. Industry 4.0 and Cybersecurity

A common practice in studying history is ‘Periodization’. The concept behind it is that dividing historical periods in well defined time frames, allows us to refer more easily to a period of time and allows us to better understand the connections that current events might have to events which happened in the past [IAT14]. An example of such periodization, which uses a timeframe, events linked to that specific timeframe and the idea that time periods can be linked using the events happening in that time frame, is the periodization of Industrial Revolutions.

The Industrial Revolutions, which have been divided in four different timeframes, are a perfect example of the practice of Periodization put to use. Having four different time periods being referred to as a period of revolution in industrial processes, highlights connections in the events happening between each one of those time frames in history.

The connection between each of the Industrial Revolutions is that in each one of these time periods, introductions of new revolutionary technologies occured in the manufacturing space. The First Industrial Revolution, which started in the 18th century, is characterized by the introduction of the first machinery which substituted hand production methods for some of the tasks. The Second Industrial Revolution, usually linked to World War I, saw the widespread adoption of technological systems such as telegraphs and railroad networks amongst other introductions such as assembly lines. The Third Industrial Revolution, which began in the 70’s saw the introduction of the first automated tasks and the first computers introduced in the assembly line. The Fourth Industrial Revolution begins in the 2010s, is the revolution we are currently living, and the one we will be focusing on [IRF].

1.1. The Industry 4.0 movement

The Fourth Industrial Revolution, now commonly known as Industry 4.0 is characterized as the time period in history where implementation of information and communication technologies is taking over traditional manufacturing practices. While

the last two Industrial Revolutions both heavily involve the introduction of computers in the manufacturing space, they are differentiated by the fact that in the Fourth Industrial Revolution, the interaction between humans and technology is much more closely knit, leading to one unified network, and thus to the creation of “cyber-physical production systems” [IRF].

As mentioned earlier, the Fourth Industrial Revolution is often referred to as the Industry 4.0 movement. The terminology comes from Germany, first seen in 2011, the term ‘Industrie 4.0’ was used by Henning Kagermann, Wolf-Dieter Lukas and Wolfgang Wahlster, in a paper called “Industrie 4.0: Mit dem Internet der Dinge auf dem Weg zur 4 industriellen Revolution”, which translates to “Industry 4.0: Internet of Things on the Road to the Fourth Industrial Revolution”. In this paper, which was presented at the Hannover Fair, they presented a set of investments to the German government going from infrastructure and even touching public schools, investments which were meant to propel German manufacturing back to a state of dominance in the global economy [I4M11].

Germany definitely wasn’t the only country that realized a shift in manufacturing practices was needed to keep up with the demands of the market. For example, also in North America another project was initialized to modernize manufacturing technologies. Not necessarily called Industry 4.0, the term used was Industrial Internet, and it was used by the General Electric company in late 2012. The term was used in the foundation of the Industrial Internet Consortium. The consortium is open to any company, as it was founded to accelerate the development, adoption and widespread use of interconnected machines, devices and intelligent analytics. The consortium was later rebranded in August 2021 as Industry IoT Consortium, moving it’s main goal to delivering business value to industry, organizations, and society by accelerating adoption of trustworthy IoT technologies. It is worth mentioning that the aim of the consortium is not to sell products or services but to drive progress and to help members get the best return on their IoT investments [ICB17].

Industry 4.0 thus is a very recent concept, therefore it has different interpretations based on the needs of the manufacturing industry of each nation. There have been attempts however at trying to standardize what being an Industry 4.0 compliant

technology means. An example of one of the most cited attempts has been written by Mario Hermann, Tobias Pentek and Boris Otto in a Hawaii International Conference on System Sciences (HICSS) research paper. In this paper, they divided Industry 4.0 principles into four main categories:

- interconnection, allowing machinery, sensors and other devices to be connected with each other, and to be connected with the people monitoring them;
- information transparency, thus the ability to collect a lot more diagnostic data on each machine, so as to further analyze and optimize current processes;
- decentralized decisions which leads to increased automation of tasks. With the inclusion of more diagnostic data being captured, it is easier to optimize tasks even from outside of the production facilities;
- technical assistance, with systems assisting humans in decision-making and the ability of systems to perform tasks that are less safe for humans. Since the systems collect more diagnostic data, they could also be more aware of issues, helping users solve them or even automatically solve tasks without need for human intervention [DPI16].

These four main design principles most likely derive from looking at the development and improvement of a series of enabling technologies for Industry 4.0. The definition of enabling technology is: ”an invention or innovation that can be applied to drive radical change in the capabilities of a user or culture”. Thus the development that those technologies received in recent times helped the widespread application in the manufacturing space. These enabling technologies have been identified by the Boston Consulting Group in 2015 and have been divided as follows:

- advanced manufacturing solutions thanks to robots becoming autonomous, flexible and cooperative;
- additive manufacturing, the classic example of which is 3D printing, enabling companies to create customized products for their manufacturing line;

- augmented reality systems, which have a variety of uses in the manufacturing space. For example easily being able to place virtual machines in a real environment is a move that if done before building and placing new machinery could dramatically cut costs;
- simulations in all stages of production, useful to mirror the real world thus increasing quality, decreasing costs and making production more efficient;
- improved horizontal and vertical system integration, meaning that all departments of a business are much more closely linked thanks to data being shared;
- industrial internet of things, field devices communicate and interact with more centralized controllers, thus devices, which didn't have internet, now have it;
- the cloud, moving machine data and functionality to the cloud is achieving levels closer to local storage and computability;
- cyber security, due to the more common interconnection of devices, protecting critical industrial systems is a big focus of current manufacturing systems;
- big data analytics. Analytics based on large data sets optimizes production quality, saves energy and improves equipment service [BCG15].

The type of products, services and machines which are compliant with the guidelines described above can be further grouped in these macro-categories: Cyber-physical systems, Internet of Things devices or platforms, Cognitive computing applications and on-demand availability of computer system resources [HDI17].

In this thesis, we will be analyzing further the Internet of Things macro-category. We will be looking at where Industrial Internet of Things as a field comes from, it's widespread implementation of its technologies in various fields of manufacturing, and some of the limitations which may arise with its inherent flaws.

1.2. Industrial Internet of Things

Industrial Internet of Things is an obvious ramification of the more well known idea of Internet of Things. As with Industry 4.0, origins of the term Internet of Things are many, the most probable of which is credited to Kevin Ashton in 1999. He used the term in a presentation for Procter and Gamble, introducing to them the idea of adding the then new technology of RFID in the supply chain. Ashton's idea of IoT is although slightly different from what IoT is perceived to be nowadays, Ashton thought that people often went unnoticed in the idea of a network of interconnected devices, but they were actually vital as they are the main provider of information. However, if computers are empowered with ways of gathering information about themselves and their environment, they can capture more information than humans, without human error, and can analyze the captured data. This combination of factors lets us delegate diagnostic work to computers thus allowing humans to concentrate on different parts of the manufacturing process [RFI09].

The evolution of Internet of Things, as it's more commonly known today, has standardized a new definition of the concept, more inclined towards the idea of embedded objects with sensors, processing ability, software and other technologies that allow for them to connect and interchange data between devices via the internet, also giving internet connectivity to devices that in the past were unable to do so [WII21]. The peculiarity behind the modern concept of IoT, is that while the original concept stemmed from the industrial space, nowadays IoT has been massively applied in the consumer electronics space so much so, that the definition of IoT itself changed. Applications of IoT concepts are now common in household appliances, this can be seen from the fact that fridges, ovens, dishwashers, coffee machines, vacuum cleaners and light bulbs nowadays are all connected to the internet, they are all integrated to voice assistants, they all have their own mobile applications for monitoring and controlling, slowly becoming the norm in modern households.

While IoT is now common in households, we want our focus to stay on the sector that is now segmented by the name Industrial Internet of Things. Historically, control and automation of manufacturing processes was first introduced with the use of Programmable Logic Controllers (PLC) solutions. PLCs, invented by Dick Morey in 1968, are basic microprocessors built with the idea of being long lasting and easily

programmable to allow multiple input/output arrangements in real time. The possibility of making the machine work differently based on their input is what makes it the first example of automation in industrial processes [MAM08].

The most obvious evolution of PLCs can be seen in Distributed Control Systems (DCSs). First introduced by Honeywell and Yokogawa in 1975, they improve on PLCs since they allow for automation and control over bigger sets of machinery, distributing control across the whole plant rather than being centralized thus removing the idea of single point of failure. The concept of having a control room with information about the plant, also evolved thanks to the adoption of DCSs, as the technology allows for high levels of plant status and production monitoring. The added set of functionalities brought the drawback that DCSs are slightly harder to program, and slightly less reactive when given an input, since further processing of information is involved. These two aforementioned solutions are not the only examples of industrial automation which came before adoption of IIoT solutions. After the advent of PLCs and DCSs, a variety of similar control systems for factories such as SCADA also gained in popularity over time. SCADA, the acronym of Supervisory Control And Data Acquisition, is a system of software and hardware components built specifically for monitoring and automation of industrial machinery. SCADA systems are often employed in mission critical contexts, meaning areas where risk of failure could lead to extremely dangerous consequences such as gas and oil distribution. All these various control systems differentiate from one to the other although still offering a set of similar features, furthermore they are mostly closed solutions to one single manufacturer. This in turn means that the invention of a more modern automation solution never fully makes the application of an older solution obsolete, even at times integrating elements of the older solution in the newer solution [EIC13].

Focusing back on Industrial IoT, its solutions can be seen as an evolution, and not as another variation on DCS solutions. This differentiation is done because of a series of enabling technologies, the implementation of which allows for clearer nomenclature. Initially the advancements in cloud computing brought massive flexibility to monitoring manufacturing processes remotely, since it is possible to diagnose issues without being on site, making the technicians jobs much easier. The

general advancement in hardware, due to the reduced cost and size of microcontrollers and microprocessors, leads generally to the technology being more applicable in manufacturing scenarios compared to proprietary devices. Often, these microcontrollers and microprocessors, can even be off the shelf products, fully programmable to one's needs and with open source support widely available. Another advancement which brought to standardization of IIoT solutions are the new and well refined standards for wireless communication, both via the web and via solutions such as Bluetooth Low Energy for short-range communications. On the web side, publish/subscribe protocols such as MQTT or request/response RESTful protocols are both valid alternatives based on the use-case scenario to exchange information between machinery.

Having talked about single advancements in IoT and thus IIoT technologies, it is now possible to identify what an IIoT solution is and the various building blocks that create an IIoT solution. The building blocks of an IIoT solution are also defined as the links in a value chain. The value chain illustrates how the different building blocks operate, and what each block contributes to the overall experience of using an IIoT solution. An illustration of the IIoT value chain can be seen in Figure 1.1, where we can see the building blocks listed in the following way:

- data capturing module. The data can be either sensor or actuator data, or data caught from a software about machinery;
- preprocessing and pre analyzing module. Thanks to edge computing, a modern distributed computing approach in which bandwidth usage is massively cut, the data is initially categorized so as to make later analysis more relevant;
- processing data module, where the caught data is actually processed, and where automation commands occur;
- data analysis module, where machine learning, artificial intelligence, and big data techniques are used to analyze the data, so as to further improve data processing and data capturing algorithms.

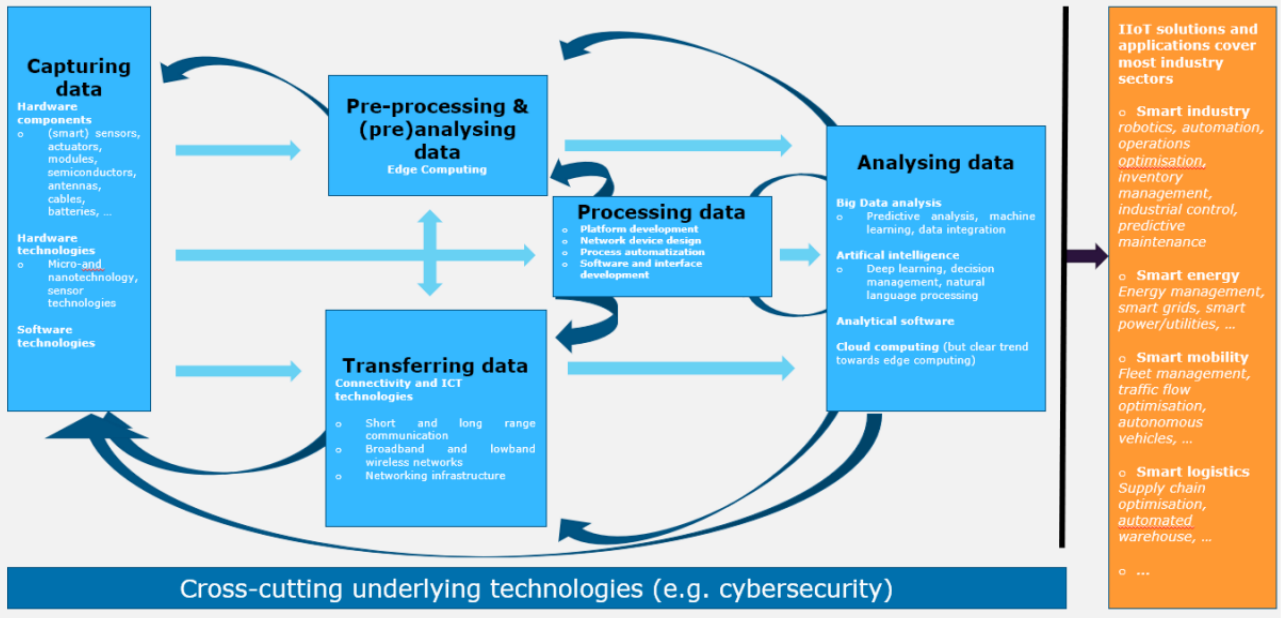


Figure 1.1. IIoT value chain description

As we can also see from Figure 1.1, these modules are also built to interact and communicate the results with each other [SVC19]. The interoperability of modules in IIoT solutions is one of it's main selling points, however this interoperability is one that also raises the most common concern with IIoT solutions, how secure is a system with openly interchangeable pieces?

1.3. Cybersecurity and Industrial IoT

In cybersecurity there's a saying: "Fast, secure, cheap, pick two". This saying is related to the self-explanatory idea that it is often difficult for these three features to coexist in computer science based solutions. This triangle of options is often true because of a series of factors, the most common of which is that the standards on which the internet relies on, were created in a time where cybersecurity wasn't a vital part of design, but only an afterthought. Since Industrial IoT solutions can also be considered computer science solutions, the concept stands true also in this scenario. This raises the obvious issue of which of the three is better to cut, and the answer is that it depends. For example, in the case of communications where no sensible data

is shared, it is often considered ok for the communication to be less secure, on the other hand sensible data can in some cases be shared without hurry, so speed is sacrificed. Convenience in communication is often lost in the case of devices from different manufacturers communicating between different blocks, while it is true that Industrial IoT allows for flexibility between layers as protocols are standard, the added flexibility is not always free, thus sacrificing inexpensiveness.

A different cybersecurity issue altogether, not strictly related to the Industrial IoT space but still worth mentioning, is the issue of well known manufacturers that have never built computer science based solutions. It is an issue that is much more common in the IoT space for households in house appliances, where said manufacturers are usually not educated on the concept of cybersecurity itself, often completely omitting its principles when creating hardware or software for IoT.

While the security concerns might deter companies from investing in Industrial IoT solutions, there are hardware solutions easily applicable to IIoT solutions that are often used, and that haven't been mentioned yet. The one we will be talking about is a specific hardware solution that can help with making data transfer secure, it does so by making data transfer physically unilateral, securing certain pieces in a closed network, with those pieces outputting data to the end user. This hardware is commonly called a data diode, and as mentioned earlier, it is used as an unilateral single point of communication to send data from an isolated network towards a public network.

Data diodes, as we can see from the diagram in Figure 1.2, are able to communicate unilaterally thanks to physical limitations in the construction of the circuit itself, that allows for current to only flow one way. This in turn leads to those pieces of the network being completely erasing any points of entry in the network.

There are similar software-side solutions to create unidirectional communication in protected networks, the most common of which is the use of firewalls, the main difference however between firewalls and data diodes is the level at which they operate. Taking into account the ISO/OSI stack, the conceptual standard for packet communications, firewalls most commonly operate around the network and transport layers, where TCP/IP operates. This means that attacks on the lower layers of the

protocol (physical and data link) there are still vulnerabilities which could be attacked, layers which are protected by data diodes.

Given their security oriented nature, the main application of data diodes is in mission critical situations, where a part of the appliances has to be secured, but information about those appliances is still needed. Example use cases can be oil production plants, which are offshore, the status of which has to be monitored, and it is obviously not safe to connect directly to those plants as being on an open network could lead to attacks. Similarly air traffic and railway traffic monitoring appliances could be placed all over paths of interest.

Industrial grade data diodes however have a big drawback, which is their cost. A single diode device starts from $6000 because of the complex circuits made for unidirectionality to work. Furthermore in industrial applications often a diode needs to be installed for each single piece of machinery, the cost of which can get very expensive. Luckily there are cheaper alternatives to industrial grade data diodes which do a similar job for most use case scenarios. For example a cheaper diode could be achieved using serial communication via an RS-232 cable.

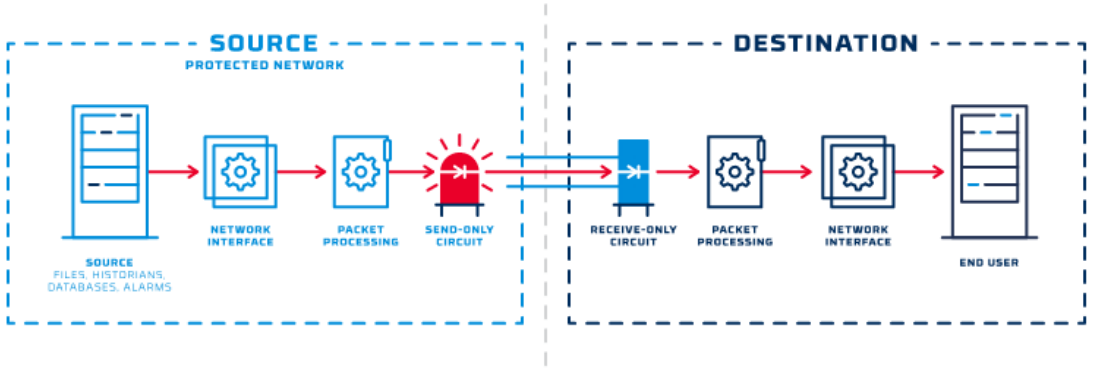


Figure 1.2. How data diodes function

This works because the RS-232 standard for communication uses a pin to send data, and the other to receive data, unilateral communication thus could be achieved by cutting the receiving pin from the side that is sending the data, and similarly the transmitting pin could be cut from the side that receives the data. While this solution obviously does not function as smoothly as an industrially built diode, it will work for most applications.

Another issue with data diodes is that malfunctions in the private network could be difficult to signal to the end user. If the diode itself malfunctions for example, data will be lost and the machine itself has no way to diagnose the problem or to send a warning message. This problem could be solved with bidirectional communication, for example having the unprotected side ping the data diode periodically so as to check its status. However diodes are an unilateral communication device, so this procedure is not immediate. There is a way to make bidirectional communication possible, within a diode environment it is usually done by placing two data diodes, one from the protected network to the unprotected network (as seen in Figure 1.2), and another diode placed sending data the opposite way. In this bidirectional model data transfers can only be started from the protected side however, so as to take software precautions before starting communication [OWL18].

Chapter 2. Disserting ALSTOM

I had the pleasure of conducting my internship activity in collaboration with the Italian Branch of ALSTOM Group, more specifically with their office in Bologna. ALSTOM Group is a French multinational manufacturing company operating in the rail transport markets. Market leader in both train building and in railway signaling systems, they are responsible for the production and maintenance of high speed trains deployed all over the European railway network, local trams and metro trains deployed in some of Europe's biggest cities and responsible for the monitoring systems behind those trains and tracks. Their entrance in the italian market can be traced back to when, in the early 2000s, they bought 51% of Fiat Ferroviaria shares, giving them access to the patent for their Pendolino train and a leading position in the Italian railway transport market.

2.1. How ALSTOM operates

ALSTOM, with now many offices and production plants all over the Italian territory, now has two main high competence locations. One of them is the Savigliano (CN) production plant, where Avelia Pendolino high speed trains and Coradia Stream regional trains are built amongst others. The other location is found in Bologna, and it is the leading location in Italy as far as railway signaling and monitoring applications are concerned. The missions of the railway signaling and monitoring branch are not only to make sure that no accidents happen on the train lines, but also to offer maintenance in case some pieces become faulty, and also to foretell the need for maintenance of specific components in case of time degradation. This need comes from the push from all major stakeholders of the company to offer not only high quality and fast transport, but also high efficiency support and sustainability within the materials involved. It is thus very important that railway signaling and monitoring systems be as efficient as possible, so as to minimize wasted materials, and downtime when a component needs to be replaced. This is one of ALSTOM's moves towards a predictive model rather than a reactive model for their support structure, which in itself is revolutionary.

ALSTOM Service Signaling current monitoring solution starts from the Control Room. The Control Room is a mixture of highly advanced hardware and software components and technologies. Amongst its many features, It is able to offer an in-depth view of all devices, looking for anomalies and degradation of components, and also making sure that rail traffic stays smooth when repairs are occuring. The Control Room works together with ALSTOM servers, as diagnostic data captured from sensors both found along the railways and on the train themselves are sent under a private network to the servers, where they are elaborated, organized and analyzed, improving the quality of information that the Control Room gives to ALSTOM's technicians. The Control Room also grants access to the documentation and manuals of all components of which it's giving information about, making debugging of issues much more immediate. It is also responsible for keeping track of the maintenance history for all components, done by ALSTOM tech support technicians.

The current solution for tech support is divided in a 3-tier structure. The first tier is remote diagnosing of problems using the data captured by the Control Room. ALSTOM technicians have the chance to connect remotely to their server racks placed either on their trains, or along railway lines to see which component of the system is malfunctioning. Obviously this connection is always done under a private and encrypted network, and only if the owner of said piece agrees to it. After connection to the server racks, if malfunctions are spotted and the client has the broken piece in inventory, the client himself is easily able to substitute components using its part number. The second tier of diagnosis is done via help desk both remotely or on site. This type of technical support is done when the replacement might be more complicated, or when the client being helped doesn't have replacement components in stock. The third tier help desk is somewhat connected to the development and improvement of the ALSTOM components, as it works on bug fixing and repurposing of components. This essentially means that if the malfunction of a component is software-side, the bugs are identified and fixed, furthermore this tier of service is also responsible for the repurposing of components, which might have been replaced initially by the client, but might still be reusable. The support structure, as discussed above, is a well oiled system that has proved to be successful for many years, but as a leading company in their sector ALSTOM is

always looking to evolve, furthermore, their engineers are always looking to implement the most up to date technologies in their monitoring solutions. ALSTOM Bologna as of recently hadn't dived into the world of Industrial IoT devices, and seeing how their aim is to constantly improve reliability, availability, efficiency and security of both their railway monitoring systems and of their technical support to their customers, an interest sparked in their engineers to find out how integrating Industrial IoT based solutions could be beneficial to their technical support model.

While ALSTOM's current system is highly innovative and technological, their solutions have always leaned towards the engineering spectrum rather than the computer science spectrum, furthermore, using purpose-built components makes building a secure and well integrated infrastructure much more immediate. As mentioned in the earlier chapters, Industrial IoT solutions have many benefits, but as we also mentioned earlier introducing Industrial IoT based solutions in a company that as of today hasn't had to deal with them, means having to deal with the inevitable cyber security concerns that are likely to arise at some point in the future [ALS22].

2.2. Internship explained

In the last chapters a series of singular concepts have been analyzed, which are now slowly going to come together to introduce my internship project. We initially discussed the security concerns when applying IIoT solutions, mainly when the devices are placed in the workflow or made by a company without previous cybersecurity skills. We then also discussed that ALSTOM is already aware that secure communication is vital in the industrial workflow, since their remote connections to their purpose built devices are already programmed to be secure. Furthermore we also spoke about ALSTOM's interest in Industrial IoT solutions to improve their signaling capabilities. Putting all these pieces together it is obvious that if ALSTOM wants to integrate IIoT solutions they won't only need a way to integrate it in the system, but they are actively looking at cybersecurity being a vital part of the solution itself.

To find out what needs to be built in my internship project, we can look at a diagram of ALSTOM's current structure shown in Figure 2.1. Currently, the communication with their custom built remote devices, happens over an encrypted connection, with information being saved in the ALSTOM servers and displayed in the Control Room. Since their interest is to introduce Industrial IoT off the shelf devices in their on site racks, a similar encrypted communication system has to be integrated for them to communicate data to the servers, this is where another subject which we spoke earlier of comes in, data diodes.

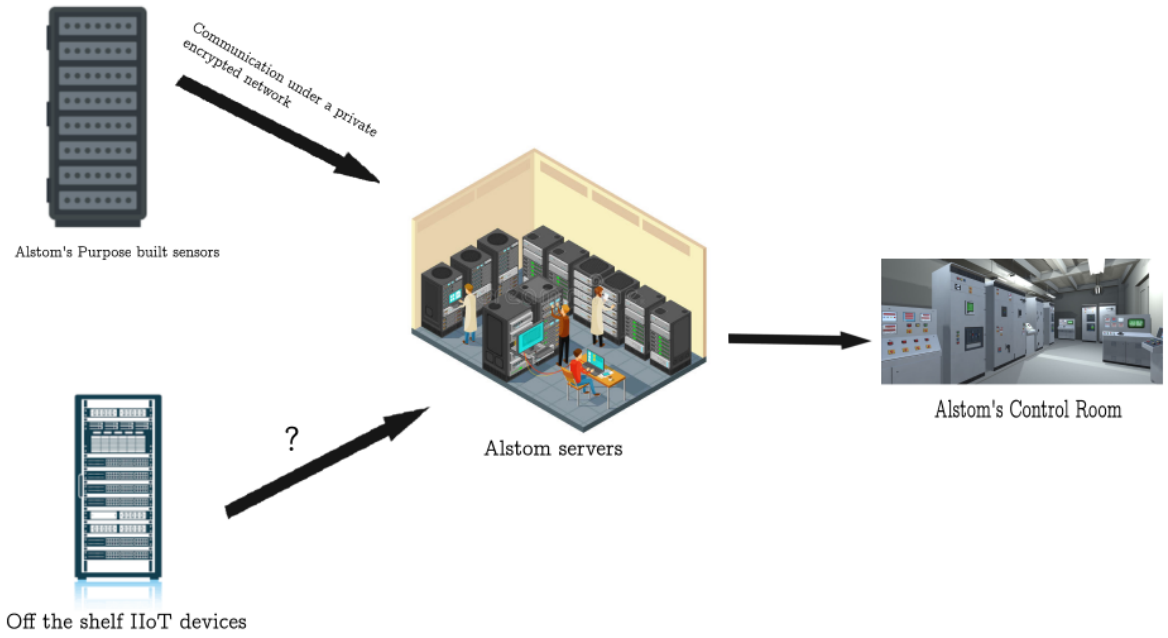


Figure 2.1. Current network structure structure

As we had mentioned in the previous chapter, data diodes are a hardware solution well known in the cybersecurity spectrum, useful when needing information from a secure network but not wanting a point of entry to that network, furthermore the off the shelf Industrial IoT monitoring devices that ALSTOM could possibly include in their system, will have issues interacting on the same encrypted network the private devices talk on. This means that including a data diode in ALSTOM's infrastructure should make introducing off the shelf devices much more immediate and will solve the issue of not being able to use a private network. However data diodes have a drawback which had been mentioned earlier, the cost of accessibility. Considering the high amount of device racks which will have to be equipped with a data diode,

their expensive cost is a limit in implementing high amount of them, as it would make the solution very expensive, so finding a diode alternative with a very similar functionality to an industrial grade data diode will be the aim of this thesis project. Furthermore most industrial grade diodes are closed-source, and trying to build an open source solution at this point gives many more options when deciding other parts of the project, and additional functionalities.

2.2.1. Serial Cable

The starting point for this project will be another concept which has been mentioned earlier, the idea that a data diode can be simulated by using a RS-232 compliant serial cable with the RX pin cut. This cable is obviously not the only element that is needed to create a data diode clone, as we will need software to transmit the data, the cable needs a sender and a receiver for the data, and eventually we will need data to transmit. Before diving into the other elements, there are a few reasons to use more specifically a serial RS-232 compliant cable rather than other cabling option. Firstly on this type of cable cutting the receiving pin is a hardware modification, compared to software modifications which also lead to unidirectionality of the communication, a hardware modification is much more secure as it is not crackable. Furthermore there are many types of RS-232 compliant serial cables, as it is a well known and flexible standard, thus there will be more freedom when choosing the specific cable type. In this case the cable chosen for the project will be the TTL-232R-3V3 USB-to-serial cable with a specific configuration. The choice went to this cable as it’s compliant with the standards to use, it’s inexpensive, it has a well written documentation and the transmit/receive pins on one of the ends being singled out allows for custom configuration without having to break it’s structure.

2.2.2. Raspberry Pi

Having chosen the type of cable, we have to choose which devices we will plug the cable into. For this particular use case scenario, a very powerful Industrial IoT microprocessor device will be used, which is the Raspberry Pi class of devices.

Raspberry Pi devices are a good option for this type of application as they are relatively inexpensive, there are many models which to choose from based on the software requirements and lots of software could be handled by its many CPU options. Very importantly for this project, the Raspberry Pi class of machines tends to have a wide range of ports which can be used, and most Pi devices also have access to pinout ports which support many standards such as GPIO. Raspberry Pi machines also fit with the open source philosophy, as it usually runs an open source operating system and has lots of support in the open source community. For our specific use case scenario, our Raspberry Pi needs to have enough RAM to handle big data chunks, it needs to have a fast enough CPU to handle the utilities we want to run, it has to be possibly be used without cooling, possibly inexpensively. Since the Raspberry Pi 4 has to be cooled to be used at its best, and the Raspberry Pi 2 CPU has maybe a little too low clock speed, we will use a Raspberry Pi 3 with the following configuration:

Raspberry Pi 3 Model B Ver1.2:

- Quad Core 1.2GHz Broadcom BCM2837 64bit CPU;
- 1GB RAM;
- BCM43438 wireless LAN and Bluetooth Low Energy (BLE) on board;
- 100 Base Ethernet;
- 40-pin extended pinout with GPIO support;
- 4 USB 2.0 ports;
- 4 Pole stereo output and composite video port;
- Full size HDMI;
- CSI camera port for connecting a Raspberry Pi camera;
- DSI display port for connecting a Raspberry Pi touchscreen display;
- Micro SD port for loading your operating system and storing data;
- Upgraded switched Micro USB power source up to 2.5A;
- 16GB microSD card;

The choice of a Raspberry Pi, almost a full fledged computer, offers tons of liberty when having to choose how to transmit data and how to gather the data. Transmitting data, thanks to the use of Raspberry Pis, can be done using an open

source data transfer script written in Python with the right amount of prevention in case of failures. The synchronization of transfers will be looked at later [RPI22].

2.2.3. Data Gathering

As far as data gathering goes, it is necessary to find a software that is able to get parameters from the other off the shelf IoT devices, then format them in a file and send them to the other Raspberry Pi. The sending part has already been covered, but for the data gathering part another common Industrial IoT solution could be used, which is a network monitoring software. These types of software offer tons of features such as gathering data from other devices on the network, the possibility of creating graphs with relevant data, they often offer the chance to set up warnings in case there are values out of the norm, warnings which can be sent to a variety of messaging services. Network monitoring softwares usually can be interacted with using APIs and there's a lot of open source options as far as monitoring softwares are concerned. Furthermore some network monitoring tools provide the ability to create virtual environments with user created data, meaning that it could be possible to create an environment similar to that of the Control Room on one of our Raspberry Pi's. The most well known network monitoring software nowadays is PRTG Network manager, but it is only made to run on Windows server, and it is a closed source solution which won't run on an ARM based processor like the one on the Raspberry Pi. The three alternatives which are commonly used in industrial settings are Cacti, Nagios and Zabbix. These three are all open source, they all work on ARM processors and have very similar architecture, so picking a specific one will be done looking at which one of these softwares gives us the most functionality we might need. Cacti is a stable solution that has been in development since 2001, this leads sadly to it having less modern features compared to some of the other network managers in favor of stability, and this lack of features leads to Cacti being used less in the user space. Even if some of these features won't be used for the project, picking a program that has a large user base will make debugging errors easier. Cacti has a very straightforward way of importing data into it, but it has very little documentation on the idea of recreating a device which is not actually on the current

network, so it was decided to look at other options. Nagios is the second option that was analyzed, it is stable, widely used, modern and has a big user space, sadly it has a very limited free version, and it is very comparable to Zabbix, which instead is free. Zabbix is the option that was chosen for all the reasons mentioned above including some other factors such as the fact that their Raspberry Pi install guide is also very detailed, it includes the utility zabbix_sender which can be used to import data [ZBX22].

2.2.4. Python and Cron

As mentioned in chapter 2.2.2, a Python script will be used for data transfer. Python is a high-level general-purpose programming language created by Guido van Rossum in 1991. Designed to be easily readable thanks to its focus on good looking and readable code, done by indentation and the use of objects. It supports both structured, object-oriented and functional programming and has a large standard library and a large variety of user built libraries, furthermore the large variety of user built libraries also make it so Python is the programming language of choice in many sectors outside of computer science. Python was specifically chosen for this project as it features libraries to use the Raspberry Pi GPIO ports, libraries for serial communication and a library called pyzabbix which interfaces with the Zabbix API which will be used in the future [PYT22].

Cron is another utility which has a smaller job but will also be used in this project. It is a command line job scheduler, and in this project it will be used for launching the python script responsible for receiving data on one of the Raspberry Pi machines, and responsible for launching at a specified time interval, the script responsible for sending data on the other Raspberry Pi machine.

So in conclusion this internship project will consist in creating a data diode clone using Raspberry Pi machines, RS-232 unidirectional communication, and using a network monitor instance to gather information from the private network, while using another instance of the same network monitor to display that same information on the other Raspberry Pi’s. In the next chapter we will discuss specific hardware and

software choices, and also look at how the network diagram will evolve thanks to these inclusions.

Chapter 3. The solution explained

Having defined in general terms the aim of my solution, the kind of devices and appliances we want to work with and why we picked those more specifically, it is now time to proceed to the in depth explanation on how to set up each piece, and how they work together to reach our aim.

3.1. Raspberry Pi setup

On both Raspberry Pi machines, we installed Raspberry Pi Os Lite in its latest version at the time of creation (2021-05-28 version date). The choice of operating system was done because it is considered the stablest operating system for Raspberry Pi machines without a desktop environment. It is possible to mitigate completely the need for a monitor to complete the setup, by enabling WiFi and SSH capabilities at this stage. Furthermore, if one wants to do so, after having created the image, you can include two files in the boot folder on the Raspberry Pi SD card. The first file, is a *wpa_supplicant.conf* file, used to enable and set up WiFi. An example of this type of file can be found in the code snippet shown in Listing 3.1:

```
country=<Insert 2 letter ISO 3166 -1 country code here>
update_config=1
network={
ssid ="<Name of your wireless LAN>"
psk="<Password for your wireless LAN>" }
```

Listing 3.1 *wpa_supplicant.conf* example

For SSH support, it's enough to include a file named *ssh* in the boot folder of the SD card, without extensions and without anything written inside. Upon first boot up, the default Raspberry Pi login credentials are the username "pi" and the password "raspberry". As mentioned earlier, Raspberry Pi OS Lite is an OS without a desktop environment, however, getting to system settings can be done using the command *raspi-config* which opens a system configuration application. Using the command

sudo raspi-config, you can access the utility, once in the main menu change these specific settings:

- In the system options:
    - Enable WiFi if needed for connection purposes and if you haven't done this before. Upon first booting, the country code is also set.
    - Change username and password if needed.
- In the interface options:
    - Enable SSH if you haven't done this before. Using SSH is suggested for development on these units, as we decided to use an operating system without a desktop environment/window manager configuration, and thus it's best to use your preferred terminal emulator.
    - Enable the setting "*Serial interface"*, remembering to disable login shell but enable serial interface.
- In the localisation options:
    - Change the timezone to the local one.

Outside the *raspi-config* utility the following modifications also have been done:

- Disable bluetooth:
    - It's suggested disabling bluetooth to make the final product more secure. Bluetooth can be disabled by adding the line `dtoverlay=disable-bt in /boot/config.txt` using your preferred command line text editor.
- Additional software:
    - Run `sudo apt-get update && sudo apt-get upgrade`. These commands make sure that the packages installed on both machines are up to date.
    - Run `sudo apt-get install python3` to install Python in it's latest version.
    - Run `sudo apt-get install python3-pip` to install Python's package manager.

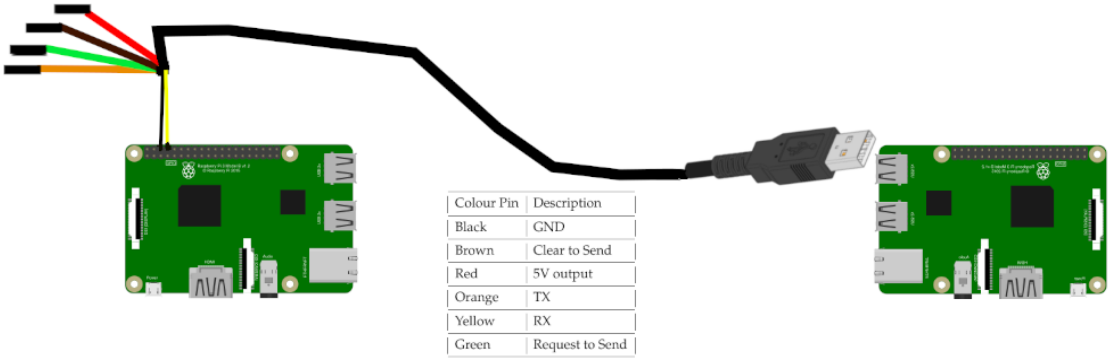
- Run *pip3 install pyserial* to install the serial library used to have the machines communicate.
- Run *pip3 install pyzabbix* to install the Python library used to interface with the Zabbix API. This packet only has to be installed on the Raspberry working on the private network.
- Run *sudo apt-get install zabbix-sender* to install Zabbix sender, the utility we use to interface with Zabbix trapper items. This packet only has to be installed on the Raspberry Pi machine localized on the public network.
- Run *sudo apt-get install git* to install the git program. Useful to clone a repository of any kind of software into a directory.

After having followed the guide above, the two Raspberry Pi machines will be ready to have the rest of the pieces installed and set up. However it is to be remembered, after completing the setup, that some system settings might need to be adjusted based on the needs of the final user. An example might be moving the private Raspberry Pi to the network it's trying to monitor.

3.2. Serial Cable connection

As touched on in the earlier chapters, our data diode will be simulated by using a RS-232 compliant serial cable. This cable will be the center piece as obviously without it, communication doesn't occur. The cable used for this project is a *"TTL-232R-3V3"* cable, which is USB-to-serial, with +3.3V TTL levels UART signals. The cable has 6-pins on one end, and USB on the other. With the cable plugged in, as shown in Figure 3.1, running the command *ls /dev/tty** on both Raspberry Pi machines, the system will check that the ports used to send and receive data are available. The port that's sending data will be */dev/ttyS0*, while the port receiving data should be */dev/ttyUSB0*. In older Pi models, instead of using */dev/ttyS0*, the port that was used instead to access the pins was the port */dev/ttyAMA0*. This change in naming caused incompatibility in some programs, thus Raspberry Pi Os will always map ports *S0* and *AMA0* to the alias */dev/serial0*, so it is suggested to

use the alias in programming. As mentioned earlier, in Figure 3.1 there is an image showing how the cable is meant to be plugged in, while in the Appendix 1 there is the full extract from the documentation of the cable, explaining the functionality of all the cable pins in detail.



Colour Pin	Description
Black	GND
Brown	Clear to Send
Red	5V output
Orange	TX
Yellow	RX
Green	Request to Send

Figure 3.1: Serial cable pinout diagram.

The cable pinout is peculiar as most pins are not used. The pins that get used are the ground pin and the pin that sends data over to the other Raspberry Pi machine. Yellow pin is plugged in GPIO14, ground can be plugged in any of the Raspberry Pi ground pins. This cable layout resembles that of a data diode, because as mentioned earlier, simulating a diode with this cable standard, can be done by simply cutting the pin which would send data from the public network to the private one.

3.3. Zabbix initialization

As discussed in the previous chapters, Zabbix is the network monitoring tool we ended up choosing for the project, keeping in mind that at the time of creation, Zabbix version 5.4 was used, running on an Apache web server and MySQL database. While there is an installation guide available on the Zabbix official website specifically for our Raspberry Pi OS version, there is also, in Appendix 2, the full installation guide followed for the creation of this project.

Having followed the guide on both machines, access to the frontends should be available by connecting to the URL *http://RaspberryPiIPadress/zabbix* where you will be greeted by the finalization of the install process. After the welcome screen, where you can pick the installation language, the next screen is a pre-requisites screen, where if installation was done correctly Zabbix ideally shouldn't find any problems. At this point Zabbix gives you the opportunity to configure the DB connection, here it is important to insert the password used earlier to authenticate in MySQL. The other parameters are supposed to look like the ones shown in Figure 3.2.

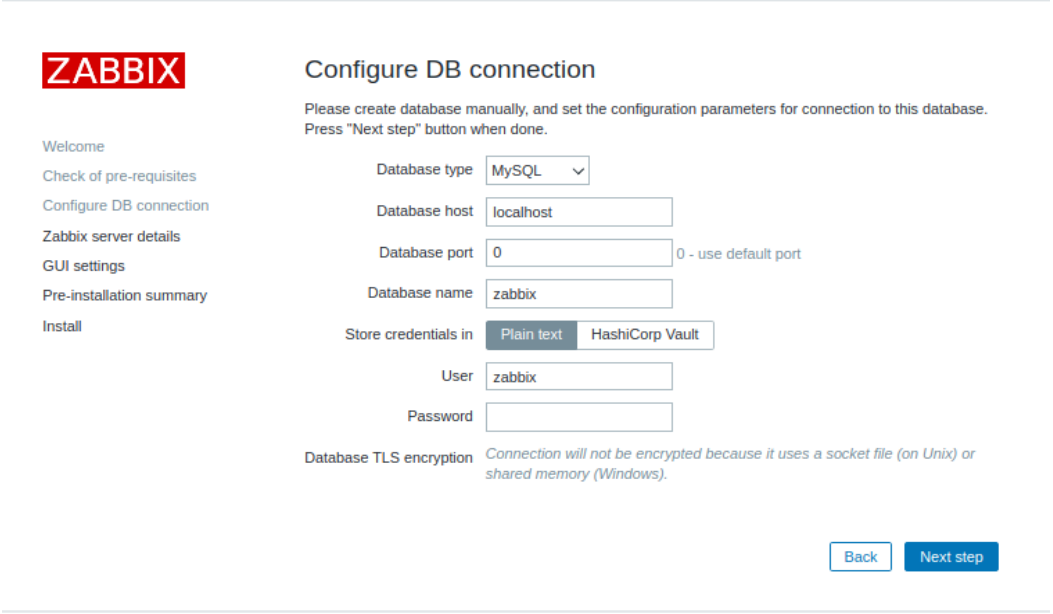


Figure 3.2. DB connection parameters in Zabbix configuration screen.

The next screen shows a series of server settings, here you can give a name to the Zabbix installation, but this step is not mandatory. Finally, in the last part of the installation, pick the correct timezone. The Zabbix frontend will now be correctly set-up, there is the opportunity to have Zabbix scan for devices in the network, or include devices manually. Furthermore being Zabbix a highly customizable system, tons of customization options can be applied now.

Before looking at customization options however, for the solution to work correctly, a couple of configuration adjustments still have to be applied to both machines via the frontend.

Raspberry Pi private network adjustments:

- Change the name of the host from "*Zabbix server"* to "*Zabbix server pvt"*.

Raspberry Pi external network adjustments:

- Add a new host named *Zabbix server pvt* without filling the interface parameter.

It is possible to add more hosts to the configuration, each host being equivalent to a machine you want to monitor. For a machine to be monitored, a Zabbix agent instance has to be installed and configured on it. On the official website it is possible to find agent install files for most of the available platforms, remembering that agents can have specific types based on the monitoring needs. Once the hosts are configured in the private Zabbix instance, it is then possible to add items, equivalent to the parameters the client wants to track for a single machine. For a seamless integration, every host and item that is added in the private network, that we want to also see on the public network, has to be configured on the Zabbix instance running in the public network, without the interface parameter and the exact same name. In the same way, every item configured on the private network has to be configured on the public network, but with the Zabbix trapper type.

Zabbix handles atypical behavior via a system of triggers and actions. The idea behind this system is the following: the user writes an expression regarding one or more parameters being monitored, if the expression is true at any moment, the trigger is activated and the user is notified in the dashboard. An action can be configured as a reaction to one or more triggers being activated at the same moment and an action can include a variety of possibilities, such as automated commands or notifications via email.

The possibilities regarding triggers and the actions connected to the activation of those triggers is endless, as Zabbix allows for a ton of customization options. In the Appendix 3 there is available a guide to get a general idea of a basic configuration including the configuration of a trigger, linking that trigger to an action and the configuration of a mail server, so as to receive a notification of the trigger being activated. For the configuration parameters, the Zabbix documentation has a high

amount of information in great detail, thus it is highly suggested to follow that for further information.

Having followed the guide in the Appendix 3, a basic trigger with a basic action should be configured. Zabbix allows for many more customization options, and even for the possibility to automate the problem solving aspect of the warnings. All the information to do so can be found on the official Zabbix documentation, which might have slightly different parameters to look at based on the version number of the program one is using.

3.4. Transfer script and cron

We will now look at the scripts that were written to make each piece come together. The Python scripts are the ones that will be responsible for API calls and file transferring, while the cron jobs will be responsible for automation of tasks.

3.4.1. Python scripts

In the spirit of keeping the project open source, the transfer scripts have been uploaded on GitHub, and in case one wants to replicate the project, it is suggested to import the repository in the home directory of both Raspberry Pi machines, and following the directions in these next chapters, as it will lead to a working system [GIT21]. In case one doesn't want to import the GitHub repositories, the content of both transfer scripts can be found in Appendix 4. In case one imports the repository via GitHub, it is still suggested to make sure that inside the repo there are both `/home/pi/DocInternship/send/main.py` and `/home/pi/DocInternship/rcv/main.py` files. In case one doesn't want to import the repository via GitHub, make sure to stay consistent with the file paths as the thesis is written with those file paths in mind, and the project was tested with those same file paths.

The code which runs on the Raspberry Pi sending data will work under a cron timer of five minutes which we will talk about later, and will send data with a 5 minute

interval. The code snippet in Listing 3.2, is responsible for making sure that the data taken by the API starts from the time from file *time.txt*. This script also makes sure that if the file with the time specification doesn't exist, it gets created.

The receiving script will open at startup, listening for data on the serial port, receiving all the data every time it's being sent, dividing it in files of 250 lines for zabbix_sender, and then pushing the data to the trapper items. The division of the file in 250 lines chunks, is sadly a limitation of the utility *zabbix_sender,* luckily it hasn't invalidated how the final product works.

Both Python scripts also print debug statements on a log file, which can be viewed and checked for errors by using the command *tail -f /home/pi/DocInternship/send/logger.log* or the command *tail -f /home/pi/DocInternship/receive/logger.log*. Obviously the command changes whether one wants to monitor the sending or the receiving Raspberry Pi.

```
def setDate():
    if os.path.isfile('/home/pi/DocInternship/send/time.txt'):
        f = open('/home/pi/DocInternship/send/time.txt', 'r')
        timeFrom = int(f.readline())
        timeTill = timeFrom + 300
        f.close()
        f = open('/home/pi/DocInternship/send/time.txt', 'w')
        f.write(str(timeTill))
    else:
        timeTill = int(time.time())
        timeFrom = timeTill - 300
        f = open('/home/pi/DocInternship/send/time.txt', 'w')
        f.write(str(timeTill))
        f.close()
    return timeFrom, timeTill
```

Listing 3.2: Code snippet to create or update the file time.txt

3.4.2. Cron jobs

Automation of the send and receive scripts is done via cron jobs, the setup is done in the following way. On the private Raspberry Pi, use command *crontab -e* to open up the cron job editor, if asked to choose an editor, pick the editor you prefer. Include at the end of the file the lines of code visible in Listing 3.3:

```
*/5 * * * *  /usr/bin/python3 /home/pi/DocInternship/send/main.py
@reboot  sudo rm /home/pi/DocInternship/send/time.txt
```

Listing 3.3: Cron jobs for the “Private” Raspberry Pi

Save, close and afterwards make sure that the commands have been written correctly using *crontab -l*. The first command will make the private Raspberry Pi open up the data sending file script every 5 minutes. The second command has the job of removing the file *time.txt* on boot. The reasoning behind the use of this command is the following: with the device being turned off, no data is being captured by Zabbix. The implication of this data missing, is that file transfers starting from that moment will include no data for a long time, thus we agreed that it makes more sense to delete the file, and have our code create a new file with the current timestamp.

On the Raspberry Pi on the open network, use command *crontab -e* to open up the cron job editor, if asked to choose an editor pick the editor you prefer. Include at the end of the file the command as found in Listing 3.4:

```
@reboot  /usr/bin/python3 /home/pi/DocInternship/rcv/main.py
```

Listing 3.4: Cron job for the “Public” Raspberry Pi

Save and close, making sure that the command has been written correctly using *crontab -l*. This command opens the receive script every time this Raspberry Pi boots up, and it loads the script in receiving mode, waiting for data from the Private Raspberry Pi. Both the send and the receive processes will be running as orphan

processes. It is possible to monitor their cron timings, making sure that nothing is broken by using the command *tail -f /var/log/syslog*.

3.5. How does the system work?

Having done the initial setup the project is now fully functional. However, how does it function? Turning off the devices and turning them back on will be enough for the system to start running. On the “private” Raspberry Pi, monitoring of the network will begin at startup, while on the “public” Raspberry Pi, the *send.py* script will be ready to receive the file with the data. Every five minutes the data will be updated, and possible triggers will be set up if they were configured. With the current setup transfer happens at a 115200 baud rate. This means that following the initial tests done only on file transfers which we can see in Table 3.1 below, a ≈3mb file every 5 minutes is the closest to maximum file size that the solution is able to handle. Taking into consideration that for each monitored parameter there will be around 5 lines of text, thus around 200 bytes per parameter, the amount of data that can be transferred should be plenty for most use case scenarios.

File size (mb)	Time (s)	SHA256
1	88.8	Yes
3.5	308	Yes
5	448	Yes

Table 3.1. File transfer times.

While the project is currently in a working state, due to time constraints there are a series of plausible improvements which we identified, and the implementation of which could be applied to the solution in the future. Currently the Python script responsible for receiving the file and sending it to *zabbix_sender* is done in single thread mode. For smaller file sizes this will always be fine, but for bigger files, separating file receiving and the instance of *zabbix_sender* in two different threads is

the more correct approach so as to not have overlap. Furthermore some of the parameters in the Python scripts, such as IP addresses, could be configured by a config script, which could be run before running senders and receivers.

Finding ways of optimizing further the communication protocol is always a plausible area of improvement, in a similar way. Moreover, the specific cable we used has included a set of CTS/RTR (Clear To Send/Request To Send) pins. These pins could make synchronization easier, as they allow for the two machines to communicate with each other whether they are ready to send or receive data, while not losing diode functionality, as no other data could be sent between these two pins. Another way to further try to improve the solution is also by trying more cable options with differently written file transfer scripts.

As far as improvements to make the Zabbix setup more immediate, currently importing hosts and items is to be done by hand, while a custom script could also be built so that the configuration phase is more immediate. Obviously a new configuration script will have to make sure items still have to be configured as Zabbix trapper. Initial setup of the private machine in the public Zabbix instance is also done by hand, and is a process which could be automated in the future. Furthermore Zabbix includes tons of freedom and functionalities, for example the possibility of accessing the dashboard via smartphone application. Investigating further all the possible customization options programmed in Zabbix could even lead to different Zabbix setups based on whether Alstom is trying to monitor on-board appliances or appliances placed along the railway line.

Conclusion

In conclusion, in this thesis I presented what I believe will be an integral part of the future not only for railway monitoring, but also airways and even more simply car travel. Furthermore I believe that the skeleton of what we built during this internship will be often replicated in the future by other competing companies in similar sectors.

Looking at the project more closely, even if it is in a working state, there are always features that could be improved both to make the user setup experience easier, to make the end user experience more pleasant, and to be more compliant with software engineering rules. Furthermore I consider myself extremely lucky, as in these three months of internship at Alstom, I got the opportunity to work with a company leader in its market, which gave me the opportunity to learn all the inner workings of a multinational. Moreover, this internship project involved tons of interesting technologies, which kept me always ready to study something new. The idea that tons of single parts came together to build this project is what I think makes the world of Internet of Things so beautiful.

On a more personal level, I also believe that this internship taught me how to accept constructive feedback. A skill that I think I improved on as time went on. In conclusion I can only hope that I gave Alstom a hundredth of what Alstom gave to me, and I thank them for it.

Bibliography

[ALS22] Alstom Official Website.
https://www.alstom.com/it/alstom-italia

[BCG15] Boston Consulting Group, " Industry 4.0: The future of productivity and Growth in Manufacturing industries", 2015.
https://image-src.bcg.com/Images/Industry_40_Future_of_Productivity_April_2015_tcm9-61694.pdf

[DPI16] M. Hermann, T. Pentek and B. Otto, "Design Principles for Industrie 4.0 Scenarios," *2016 49th Hawaii International Conference on System Sciences (HICSS)*, 2016, pp. 3928-3937, doi: 10.1109/HICSS.2016.488.
https://ieeexplore.ieee.org/document/7427673?arnumber=7427673

[EIC13] PACE staff writers, "Evolution of Industrial control systems", 2014, *Pace 60-year Anniversary Series: Control Systems.*
https://pacetoday.com.au/evolution-of-industrial-control-systems/

[GIT21] G. Cervone, "Internship project Documentation", 2021.
https://github.com/giuseppecervone971/DocInternship

[HDI17] G. Erboz , *"How to Define Industry 4.0: The Main Pillars Of Industry 4.0"*, Managerial trends in the development of enterprises in globalization era, 2017.
https://www.researchgate.net/publication/326557388_How_To_Define_Industry_40_Main_Pillars_Of_Industry_40

[IAT14] A. Rabinowitz, *"It's about time: historical periodization and Linked Ancient World Data"*, ISAW PAPERS, 2014.
http://dlib.nyu.edu/awdl/isaw/isaw-papers/7/rabinowitz/

[ICB17] A. Rojko, *"Industry 4.0 Concept: Background and Overview"*, International Journal of Interactive Mobile Technologies (iJIM), 2017.
https://online-journals.org/index.php/i-jim/article/view/7072

[IRF] *"Industrial Revolution - From Industry 1.0 to Industry 4.0"*, Desoutter Industrial Tools.

https://www.desouttertools.com/industry-4-0/news/503/industrial-revolution-from-industry-1-0-to-industry-4-0

[I4M11] H. Kagermann, W. Lucas, W. Wahlster, “Industrie 4.0: Mit dem Internet der Dinge auf dem Weg sur 4. industriellen Revolution ”, 2011. https://www.ingenieur.de/technik/fachbereiche/produktion/industrie-40-mit-internet-dinge-weg-4-industriellen-revolution/

[WII21] Alexander S. Gillis, *“What is Internet of Things (IoT)?”*. Internet of Things Agenda, August 2021. https://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT

[MAM08] Allison Dunn, *“The father of invention: Dick Morley looks back on the 40th anniversary of the PLC”*, Automation Mag, 12 September 2008. https://web.archive.org/web/20190609030841/https://www.automationmag.com/features/the-father-of-invention-dick-morley-looks-back-on-the-40th-anniversary-of-the-plc.html

[OWL18] Dan Crum, *“What is a data diode and how do data diodes work?”,* Owl Cyber Defense, 25 June 2018. https://owlcyberdefense.com/blog/what-is-data-diode-technology-how-does-it-work/?utm_source=website&utm_medium=learnaboutdatadiodes&utm_campaign=learnaboutpages

[PYT22] Python official website. https://www.python.org/

[RFI09] Kevin Ashton, *“That ‘Internet of Things’ thing”*, RFID Journal, 22 June 2009. https://www.rfidjournal.com/that-internet-of-things-thing

[RPI22] Raspberry Pi official website. https://www.raspberrypi.com/

[SVC19] EU Strategic Forum on Important Projects of Common European Interest: Strategic Value Chain Report – Industrial Internet of Things (IIoT). https://ec.europa.eu/docsroom/documents/37824

[ZBX22] Zabbix official website.
https://www.zabbix.com/

Appendix

Appendix 1: Serial cable documentation extract.

4.2 TTL-232R-5V and TTL-232R-3V3 Cable Signal Descriptions

Header Pin Number	Name	Type	Colour	Description
1	GND	GND	Black	Device ground supply pin.
2	CTS#	Input	Brown	Clear to Send Control input / Handshake signal.
3	VCC	Output	Red	+5V output,
4	TXD	Output	Orange	Transmit Asynchronous Data output.
5	RXD	Input	Yellow	Receive Asynchronous Data input.
6	RTS#	Output	Green	Request To Send Control Output / Handshake signal.

Table 4.1 TTL-232R-5V and TTL-232R-3V3 Cable Signal Descriptions

Appendix 2: Zabbix installation guide

Install Zabbix packages by using the following commands:

```
wget
https://repo.zabbix.com/zabbix/5.4/raspbian/pool/main/z/zabbix-release/zabbix-release_5.4-1+debian10_all.deb
sudo dpkg -i zabbix-release_5.4-1+debian10_all.deb
sudo apt update
sudo apt install zabbix-server-mysql zabbix-frontend-php
zabbix-apache-conf zabbix-sql-scripts zabbix-agent
```

Initialize now the database using the following commands:

```
sudo apt-get install mariadb-server
sudo mysql_secure_installation
```

During the secure installation set a new root password and reload privilege tables. then proceed with these commands:

```
mysql -uroot -p
#"Insert your password here"
mysql> create database zabbix character set utf8 collate utf8_bin;
mysql> create user zabbix@localhost identified by 'password';
mysql> grant all privileges on zabbix.* to zabbix@localhost;
mysql> quit;
```

Using the preferred password for the Zabbix user. Proceed then with this command, which creates the SQL tables needed to function correctly.

```
zcat /usr/share/doc/zabbix-sql-scripts/mysql/create.sql.gz | mysql
-uzabbix -p zabbix
```

And then configure the database for Zabbix server by adding the password to */etc/zabbix/zabbix_server.conf* with the parameter *DBPassword=password*. To gain access to the GUI, use the following commands:

```
systemctl restart zabbix-server zabbix-agent apache2
systemctl enable zabbix-server zabbix-agent apache2
```

Appendix 3: Triggers configuration guide

1. Go in the *Configuration* menu and select *hosts*;
2. In the *hosts* section, click on *Triggers* on the host for which we want to configure a trigger for, then in the window where all the triggers for the items are shown, click on the button *Create trigger*;
3. In this section there are a series of mandatory parameters to fill:
    - Name: The name of the trigger, mandatory parameter;
    - Severity: Severity of the trigger. For our use case we will use WARNING;
    - Expression: An explanation of the value to monitor, and the parameter for which Zabbix has to set a trigger. In our case the expression will be: *last(Host/Item)>value*;
    - OK event generation: The action that must happen for the trigger to go back to the OK state. Here it is possible to set more expressions to further specify the trigger values. In our case we will leave it as *Expression*;
    - The other parameters we won't be using in this case, however the parameters are very well explained in the Zabbix documentation. Leave them as default in this case;
4. Having set up the trigger, set up a corresponding action by moving, via the *Configuration* menu, to *Actions*, option *Trigger actions*.
5. On this page, all of our triggers can be viewed with their status. A trigger creates an event, and the action is done based on an event.
6. To create a new action, click on *Create action*. This will open a configuration window with two tabs, *Action* and *Operations*.
7. In the *Action* tab, fill in the *Name* parameter with what the specific action is connected to. In the *Condition* section, select the type of condition with which the action should be activated. This section is also really customizable, but for the sake of the demonstration use *Trigger* as type, with operator *Equals*, choosing the trigger we built earlier.
8. In the operations tab there are a lot of customizable options too. For this guide simply add a basic operation, which sends to a specified user an email. This

step gives us the liberty of resolving the problem automatically without sending an email.

9. The last part of this guide involves filling the parameters for the mail server. To do so open the *Administration* tab, and find the option *Media types*.
10. Find the Email parameter and click on it, then based on your Email client fill in the specific SMTP parameters. These are the parameters from which the email will be sent from.
11. Now in the User settings section, go to the *Profile* subsection. On the top there will be a *Media*, click on it, and add the media where you will be receiving the Email from the Zabbix server.

Appendix 4: File transfer scripts

File name /home/pi/DocInternship/rcv/main.py:

```
import serial
import time
import hashlib
import subprocess
import logging

def sender():
    f = open("/home/pi/DocInternship/rcv/data.txt", "r")
    line = f.readline()
    while line:
        x = 0
        f2 = open("/home/pi/DocInternship/rcv/tmp.txt", "w")
        while x in range(250):
            f2.write(line)
            line = f.readline()
            x+=1
        f2.close()
        subprocess.run(["zabbix_sender", "-z", "192.168.1.157",
"-i",
                "/home/pi/DocInternship/rcv/tmp.txt", "-T",
"-vv"])


def calculateHash():
    f = open("/home/pi/DocInternship/rcv/data.txt", "rb")
    sha256_hash = hashlib.sha256()
    line = f.read(1024)
    while line:
        sha256_hash.update(line)
        line = f.read(1024)
    sha256_hash = sha256_hash.digest()

    return sha256_hash


def recvHash(ser):
    while True:
        if ser.inWaiting() < 32:
```

```
            time.sleep(1)
        else:
            break
    hash2 = ser.read(32)
    return hash2


def recvFile(ser):
    eof = b'EOF'
    f = open("/home/pi/DocInternship/rcv/data.txt", "wb")

    while True:
        recvdatalen = ser.inWaiting()
        if recvdatalen>0:
            line = ser.read(recvdatalen)
            if eof in line:
                f.write(line[:-3])
                break
            else:
                f.write(line)
        time.sleep(0.001)
    logging.info("File transfer completed...")


def createSerial():
    try:
        ser = serial.Serial(
            port = "/dev/ttyUSB0",
            baudrate = 115200,
            parity = serial.PARITY_NONE,
            stopbits = serial.STOPBITS_ONE,
            bytesize = serial.EIGHTBITS,
            timeout = 1
        )
        logging.info("Serial port created...")
        return ser
    except Exception as e:
        logging.warning(e)


def main():
```

```
logging.basicConfig(filename='/home/pi/DocInternship/rcv/logger.lo
g',
                    format='%(asctime)s %(message)s',
level=logging.INFO)

    ser = createSerial()

    while True:
        recvFile(ser)

        hash2 = recvHash(ser)

        hash1 = calculateHash()

        if hash1==hash2:
            logging.info("File Transfer Success... Importing data
in Zabbix...")
            sender()
            logging.info("File imported...")
        else:
            logging.warning("File Transfer Failed")



if __name__ == '__main__':
    main()
```

File name /home/pi/DocInternship/send/main.py:

```
from pyzabbix import ZabbixAPI
import sys
import datetime
import time
import logging
import argparse
import os
import serial
import hashlib
```

```
def calculateHash():
    f = open("/home/pi/DocInternship/send/data.txt", "rb")
    sha256_hash = hashlib.sha256()
    line = f.read(1024)
    while line:
        sha256_hash.update(line)
        line = f.read(1024)
    sha256_hash = sha256_hash.digest()

    return sha256_hash

def sendFile(ser):
    eof = b'EOF'
    f = open("/home/pi/DocInternship/send/data.txt","rb")
    logging.info("Starting transfer...")

    line = f.read(512)
    while line:
        ser.write(line)
        line = f.read(512)
    f.close()
    ser.write(eof)

    logging.info('Transfer completed...')

def createSerial():
    try:
        ser = serial.Serial(
            port = "/dev/serial0",
            baudrate = 115200,
            parity = serial.PARITY_NONE,
            stopbits = serial.STOPBITS_ONE,
            bytesize = serial.EIGHTBITS,
            timeout=1
        )
        return ser
    except Exception as e:
        logging.warning(e)


def setDate():
```

```
    if os.path.isfile('/home/pi/DocInternship/send/time.txt'):
        f = open('/home/pi/DocInternship/send/time.txt', 'r')
        timeFrom = int(f.readline())
        timeTill = timeFrom + 300
        f.close()
        f = open('/home/pi/DocInternship/send/time.txt', 'w')
        f.write(str(timeTill))
    else:
        timeTill = int(time.time())
        timeFrom = timeTill - 300
        f = open('/home/pi/DocInternship/send/time.txt', 'w')
        f.write(str(timeTill))
        f.close()
    return timeFrom, timeTill

def historyToFile(zapi, hosts, items):
    y = 0
    timeFrom, timeTill = setDate()

    f = open('/home/pi/DocInternship/send/data.txt', 'w')

    for item in items:
        hostname = hosts[y]["name"]
        hostid = hosts[y]["hostid"]
        for x in range(len(item)):
            itemkey = item[x]['key_']
            itemid = item[x]['itemid']
            itemtype= item[x]['value_type']
            historys = zapi.history.get(hostids = hostid,
                    itemids = itemid, time_from = timeFrom,
                    time_till = timeTill, history = itemtype,
                    output="extend")
            for history in historys:
                f.write('"%s" %s %s %s\n' % (hostname, itemkey,
                            history["clock"], history["value"]))
        y+=1
    logging.info('Exported history...')
    f.close()


def getItems(zapi, hosts):
    items = []
```

```
    for host in hosts:
        hostid = host['hostid']
        items.append(zapi.item.get(hostids = hostid,
            output=["key_", "hostid", "hostname","value_type"]))
    if len(items) == 0:
        logging.critical('No items.. Quitting program')
        sys.exit()
    else:
        logging.info('Items found...')
        return items


def getHosts(zapi):
    hosts = zapi.host.get(output=['name'])
    if len(hosts) == 0:
        logging.critical('No hosts... Quitting program')
        sys.exit()
    else:
        logging.info('Hosts found...')
        return hosts


def login(zapi, username, password):
    try:
        zapi.login(username, password)
        logging.info("Login Success...")
    except:
        logging.critical("Zabbix server not reachable... Quitting
program")
        sys.exit()

def main():

logging.basicConfig(filename='/home/pi/DocInternship/send/logger.l
og',
                    format='%(asctime)s %(message)s',
level=logging.INFO)

    zapi = ZabbixAPI('http://192.168.1.198/zabbix')

    login(zapi, 'Admin', 'zabbix')
```

```
    hosts = getHosts(zapi)

    items = getItems(zapi, hosts)

    historyToFile(zapi, hosts, items)

    ser = createSerial()

    sendFile(ser)

    hash1 = calculateHash()
    time.sleep(5)
    ser.write(hash1)
    logging.info('Hash sent... Closing program.')


if __name__ == '__main__':
    main()
```