TU WIEN Informatics

Configuration Deployment for Reconfigurable Safety Systems

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Informatik

eingereicht von

Daniel Zainzinger, BSc
Matrikelnummer 11777778

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dr.techn. Wolfgang Kastner
Mitwirkung: Dipl.-Ing. Dr.techn. Thomas Frühwirth, BSc
Dipl.-Ing. (FH) Dieter Etz, MBA

Wien, 29. September 2022

Daniel Zainzinger | Wolfgang Kastner

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

TU WIEN Informatics

Configuration Deployment for Reconfigurable Safety Systems

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Computer Engineering

by

Daniel Zainzinger, BSc
Registration Number 11777778

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dr.techn. Wolfgang Kastner
Assistance: Dipl.-Ing. Dr.techn. Thomas Frühwirth, BSc
Dipl.-Ing. (FH) Dieter Etz, MBA

Vienna, 29th September, 2022

Daniel Zainzinger | Wolfgang Kastner

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Daniel Zainzinger, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 29. September 2022

Daniel Zainzinger

Danksagung

Ich bin dankbar für die Zeit, die ich an der TU Wien verbringen durfte und möchte meinen Dank an alle Menschen richten, die zur Ermöglichung dieser Arbeit beigetragen haben. Im Speziellen möchte ich Thomas Frühwirth und Dieter Etz für die großartige Unterstützung danken. Immer wenn ich bei einem Problem Unterstützung benötigte, wurde mir immer sofort weitergeholfen. Weiters möchte ich dem Center for Digital Production GmbH für die Finanzierung dieser Arbeit danken. Zum Abschluss möchte ich noch meiner Ehefrau danken, für welche die Zeit meines Studiums oft nicht einfach war. Immer wenn ich aufgrund von Abgabeterminen oder Prüfungen keine Zeit hatte, hatte sie immer vollstes Verständnis und unterstützte mich.

Acknowledgements

I am very thankful for the time I spent at TU Wien and would like to express my gratitude to all the people who contributed to the development of this thesis. I especially want to thank Thomas Frühwirth and Dieter Etz for their great support. Always when there was a problem, I got support right away, even on weekends. I am also thankful for the Center for Digital Production GmbH financing this thesis. I also want to thank my wife, for whom my university time was not always easy. Always when I had no time for her because of an exam or any other university deadline, she understood and supported me.

Kurzfassung

Flexibilität in heutigen Fertigungssystemen ist wichtiger denn je. Wirtschaftliche Entwicklungen fordern kleinere Losgrößen bis hin zur Stückgutfertigung. Dies erfordert eine schnelle Rekonfiguration der Produktionslinien. Industrie 3.0 kann diese Flexibilitätsanforderungen aufgrund der hierarchischen Architektur nicht erfüllen. Mit der Transformation zu Industrie 4.0 werden höhere Anforderungen in Bezug auf Interoperabilität und Flexibilität gestellt. Dies erfordert die Verwendung von Reconfigurable Manufacturing Systemen, welche die Anforderungen für die Stückgutfertigung erfüllen. Heutige Sicherheitssysteme von solchen Produktionslinien sind in ihrer Flexibilität jedoch beschränkt, da sicherheitskritische Komponenten statisch verdrahtet sind oder eine statisch konfigurierte Sicherheitsverbindung über Industrial Ethernet verwenden. Um diese Beschränkung zu beseitigen, ist ein Reconfigurable Safety System (RSS) erforderlich, welches die Rekonfiguration von Sicherheitssystemen mit minimaler Standzeit erlaubt.

Der Aufbau eines RSS erfordert neue Technologien, um sichere Kommunikation, Flexibilität und Interoperabilität zu vereinen. Der Ansatz, welcher in dieser Arbeit verwendet wird, teilt die Kommunikation zwischen sicherheitskritischen Komponenten in drei Ebenen. Diese sind die Sicherheits-, die Transport- und die Netzwerkebene. Für die Sicherheitsebene wird Open Platform Communications Unified Architecture (OPC UA) Safety verwendet. Dieses Protokoll verwendet ein Black Chanel Prinzip, um Daten zwischen zwei Sicherheitskomponenten zu übertragen. Die Transportebene, für welche OPC UA PubSub verwendet wird, ist für die Übertragung der Daten von der Sicherheitsebene verantwortlich. Diese Daten werden über die Netzwerkebene, für welche Time-Sensitive Networking (TSN) verwendet wird, übertragen. Da eine schnelle Rekonfiguration dieser drei Ebenen erforderlich ist, ist das Deployment eine Schlüsselkomponente von einem RSS. Diese Arbeit zeigt, wie mit aktuell verfügbaren Technologien, wie Network Configuration Protocol (NETCONF) und OPC UA Methoden verwendet werden können, um den Rekonfigurationsanforderungen eines RSS zu genügen. Zusätzlich werden Limitierungen aufgezeigt, welche im speziellen OPC UA Safety betreffen. Die Evaluierung am Ende der Arbeit zeigt, dass 80 OPC UA Publisher in 3.11 ms aktiviert werden können, wodurch dieser Ansatz für weitere Arbeiten vielversprechend ist.

Abstract

Flexibility in today's manufacturing systems is more important than ever. Economy development demands the production of smaller batch sizes or even lot-size one, which requires a fast reconfiguration of production lines. Industry 3.0 does not provide this flexibility because of its strictly hierarchical architecture. With the transition to Industry 4.0, machines need to satisfy higher requirements for interoperability and flexibility. This demands building Reconfigurable Manufacturing System (RMS), which can fulfill the requirements for production of lot-size one. Today, safety systems of such production lines are limiting the flexibility since safety-critical devices still use hard-wired communication lines or static configured safety connections based on Industrial Ethernet solutions. Therefore, there is a need for Reconfigurable Safety System (RSS), which allows to reconfigure safety functions with minimum downtime.

Building an RSS requires new technologies to combine safe communication, flexibility, and interoperability. The approach used in this work splits the communication of safety-critical devices into three layers, namely the safety, transport, and the network layer. Open Platform Communications Unified Architecture (OPC UA) Safety builds the first layer, using a black channel principle to transmit safety-related data. The transport layer transmits data from the safety layer with OPC UA PubSub while the network layer uses Time-Sensitive Networking (TSN) to allow real-time communication without losing flexibility. Because a fast reconfiguration of these three layers is required, the deployment is a key factor of an RSS. This work shows how currently available technologies like Network Configuration Protocol (NETCONF) and OPC UA Methods can be used to perform a fast reconfiguration. Additionally, also the limitations, especially for OPC UA Safety, are pointed out. The evaluation in the end of this work showed, that 80 OPC UA Publishers can be enabled in 3.11 ms, which shows that this approach is promising for further works.

Contents

Kurzfassung **xi**

Abstract **xiii**

Contents **xv**

1 Introduction **1**
1.1 Problem Statement . . . 2
1.2 Aim of the Work . . . 3
1.3 Methodology . . . 3
1.4 Structure of this Work . . . 4

2 Technical Background **5**
2.1 Open Platform Communications Unified Architecture . . . 5
2.2 Time-Sensitive Networking . . . 13
2.3 Network Configuration Protocol . . . 15
2.4 YANG . . . 18
2.5 System Configuration in IT and OT . . . 20

3 Related Work **21**
3.1 Existing Safety Protocols . . . 21
3.2 Frameworks and Concepts . . . 21
3.3 OPC UA over TSN . . . 22
3.4 Autoconfiguration of OPC UA and TSN . . . 23
3.5 TSN configuration via OPC UA . . . 25
3.6 TSN combined with SDN . . . 25

4 Configuration Approach **27**
4.1 Motivating Use Cases . . . 27
4.2 Safety Communication Stack . . . 29
4.3 Safety Data Exchange . . . 30
4.4 Required Parameters . . . 31
4.5 Centralized Safety Configuration Setup . . . 40

5 Deployment Tool **43**
5.1 Structure and Interfaces of the Deployment Tool 43
5.2 Use Case: Initial commissioning 45
5.3 Safety and Reliability Analysis 51
5.4 Deployment Procedure with Error Handling 54

6 Evaluation **57**
6.1 Used Software 57
6.2 Evaluation Environment with ns-3 59
6.3 Results 61

7 Conclusion & Outlook **77**

List of Figures **79**

List of Tables **81**

Acronyms **83**

Bibliography **87**

CHAPTER 1

Introduction

In the past few decades, smart manufacturing became an important topic for factories, which is realizing the idea to connect all machines to the Information Technology (IT) world. This makes it possible that production facilities adapt dynamically to changing production, optimize the supply chain to current needs, or monitor devices for preventive maintenance. Communication according to the widely known automation pyramid [42] is strictly hierarchical, so each layer is just connected to the layer below and above. Because of this structure, sensors, actuators and the logic units are connected in a static way. The transition from Industry 3.0 to Industry 4.0 requires increased flexibility and interoperability in order to enable communication among all connected devices. This allows building an Reconfigurable Manufacturing System (RMS) [39], which fulfills the requirements of smart manufacturing.

Reconfigurable Manufacturing Systems (RMSs) were introduced in the mid-1990s to combine the advantages of dedicated serial lines and flexible manufacturing systems [39]. The principle goal of an RMS is to to enhance the responsiveness of manufacturing systems to unforeseen changes in product demands. Nevertheless, these systems did not consider that in some use cases it is also necessary to reconfigure the safety system to react on new production requirements. A Reconfigurable Safety System (RSS) addresses that problem and allows to reconfigure a safety system with minimum down time.

An RSS is required to provide at least the same level of safety than a conventional safety system at all time, even during a reconfiguration. Additionally an RSS must be also flexible and interoperable with hardware from different vendors. This brings completely new challenges for the area of functional safety. Current Ethernet based safety protocols (c.f. Table 3.1) are using already networks which are configurable. However, these protocols are designed to configure the safety system during commissioning and do not change it afterwards. Additionally, most vendors support only one safety protocol, which makes it difficult to mix safety devices from different vendors.

Current systems often use Open Platform Communications Unified Architecture (OPC UA) (c.f. Section 2.1) to overcome the problem of the interoperability. This protocol allows to exchange data between machines and is widely supported by different vendors. For safety-related communication, there is the new OPC UA Safety standard [8] available, which is expected to find similar acceptance like OPC UA. This new standard defines a protocol which operates on top of OPC UA and allows to exchange safety critical data between machines from different vendors. Even if OPC UA allows to exchange data independent from the vendor, the configuration it is mainly done with vendor specific programs.

1.1 Problem Statement

An RSS can contain multiple safety devices from different vendors and it is required to change the configuration of these devices with minimum downtime. An RSS uses safety functions to reduce the risk in case of a hazard. Such a safety function is defined by the ISO 12100 as a function of a machine whose failure can immediately increase the risks [36]. In a conventional safety system, a safety function consists of three parts: the safety sensor, the safety logic, and the safety actuator. To get the flexibility that an RMS requires, a network is required as a fourth part. Figure 1.1 shows a safety function with an emergency stop button as a sensor and an industrial robot as an actuator. In the worst case, all four components are from a different manufacturer. This requires at least four different tools to deploy a new safety function. The usage of different tools takes quite some time, during which the manufacturing system can not operate. Additionally, the engineers responsible for a configuration change need to be trained for those different tools.

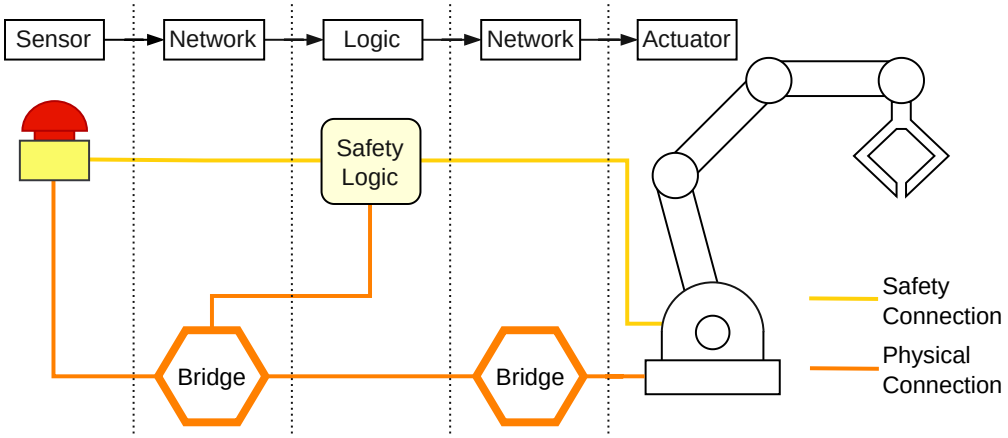


Figure 1.1: Safety Function in an RMS

The first problem is that the involvement of different tools makes the deployment too slow to fulfill the requirement of an RSS. Since the reconfiguration must be performed immediately on demand, it is also necessary to have a single point where the deployment of the new configuration can be started. The second problem is, that it is not possible with the currently available tools, because each tool needs to deploy its configuration separately.

1.2 Aim of the Work

This work should overcome these problems by deploying the configuration of safety- and network-devices of an RMS centralized and fast with the use of only one tool. This includes configuration on three different levels: Safety-, Transport- and the Network-level. An RMS which includes such a tool will be called RSS in this work. The use of only one tool brings the advantage that the engineering effort can be reduced, and it is not necessary to get familiar with multiple tools from different vendors. Besides that, it is easy to use hardware from vendors which are not already used in the production system since it will be configured with the same tool. These advantages, together with the fact that a quick deployment increases machine availability, are also an economic benefit.

1.3 Methodology

The work will be done in the following steps:

1. **Literature research:** At first, the literature research will provide a detailed picture of the current state of the art and better knowledge about the used technologies and possible approaches. The literature study will focus on deployment models, error handling during deployment, and ways to find the correct deployment sequence.
2. **Analyze different approaches:** After the literature research, different approaches will be analyzed. At first, the requirements for deploying a configuration for safety-critical devices need to be defined.

   For TSN, there are different technologies and models available, to configure the bridges and end stations. These different models and technologies will be compared with each other and the one which fits requirements the best will be chosen.

   To change OPC UA Safety together with the TSN network configuration, it is necessary to evaluate if there are dependencies that affect the order or the way of deploying the configuration.
3. **Develop a configuration procedure:** During this step, a procedure is developed that defines which inputs are required to perform the deployment and how to

choose the correct order for the deployment. Furthermore, the necessary actions for handling errors will be defined and a method will be developed to determine if the deployment process was successful.

4. **Build a prototype:** To test the findings from the previous steps, a prototype is needed. A network which connects Time-Sensitive Networking (TSN) and OPC UA servers will be emulated. On this network, the flexible deployment will be shown, and the performance can be evaluated.

5. **Evaluation:** In the end, the deployment tool will be evaluated in a simulated environment with the following criteria:

   a) Evaluate the detection of errors in different deployment phases.

   b) Evaluate duration for the reconfiguration depending on the number of nodes in the network.

   c) Analyze the the duration of activating a configuration.

1.4 Structure of this Work

This work is structured according to the waterfall model [20] with the following steps:

1. **Analyze:** Chapter 2 presents the used technologies: TSN, OPC UA PubSub and OPC UA Safety. Chapter 3 analyzes different approaches and different technologies that can be used to reconfigure the safety system of an RSS.

2. **Design:** Chapter 4 presents several use cases that an RSS should support and how to use the technologies from the *Analyze* step to handle those use cases. Additionally, possible fault states will be analyzed and defined, how to handle them.

3. **Implementation:** Chapter 5 shows how the used technologies are implemented and structured in the deployment tool. Additionally, it discusses the preconditions that the components of the RSS need to fulfill, so that the deployment tool can deploy the safety configuration.

4. **Evaluation:** The created deployment tool from the *Implementation* step will be evaluated in Chapter 6. At first, the correct error handling according to the *Design* step will be tested. In the second part of the evaluation, the performance of the deployment tool will be evaluated in a simulated RSS.

CHAPTER 2

Technical Background

In order to build an Reconfigurable Safety System (RSS), a vendor-independent technology is required to exchange safety-critical data. Additionally, the used technologies must also be configurable with an open standard in order to have a basis for a central deployment tool. OPC UA in combination with TSN fulfills these requirements [26] and will therefore be described in more detail.

First, the main concepts of OPC UA, namely its information modeling capabilities and its transport mechanisms, are briefly discussed. As OPC UA covers only the upper layers of the OSI model, TSN is presented as a suitable technology enabling real-time communication for OPC UA. A major challenge is the configuration of both technologies simultaneously on multiple devices. While the OPC UA standard already includes methods to configure an OPC UA server, the TSN standards only suggest protocols for the configuration. A common way to configure TSN devices is Network Configuration Protocol (NETCONF) in combination with YANG models. Therefore, at the end of this chapter, these two technologies are discussed.

2.1 Open Platform Communications Unified Architecture

Open Platform Communications Unified Architecture (OPC UA) is a specification that allows platform-independent and service-oriented machine-to-machine communication. It is used on different hardware like Programmable Logic Controllers (PLCs), personal computers, microcontrollers, or cloud-based servers [49]. OPC UA is available for a variety of operating systems like Windows, Linux, and mobile devices. Further there are open-source implementations available like the open62541 OPC UA stack [4].

Figure 2.1 depicts the foundation, i.e., the basic structure of the OPC UA architecture. It is built on two pillars, the transport and the modeling mechanisms. The

transport mechanisms define how data are transferred between OPC UA servers and clients and how data are mapped for different standards like Web Service Extensible Markup Language (XML) and HyperText Transfer Protocol (HTTP). To not be limited to specific protocols, it also allows adding new protocols in the future [45]. Besides the transport mechanisms, OPC UA also defines the information modeling and services [27]. The information modeling defines rules and objects on how to expose data with OPC UA and also defines additional concepts like describing the use of state machines. After starting the OPC UA server, the data from the information model are copied in the address space, where it can be accessed over the OPC UA protocol. Above the two pillars, there are the services that are the interface between server and client and define how the data from the information model can be accessed. This allows the client to access just the required data without understanding the whole model [45]. On top of this basic structure, there is the Base OPC UA Information Model, which can be extended by developers and vendors.

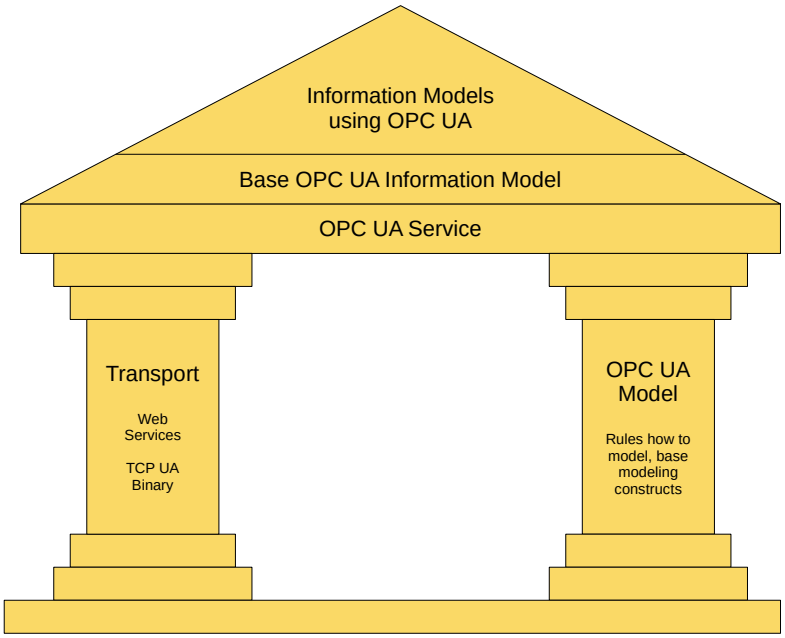


Figure 2.1: The foundation of OPC UA [45]

2.1.1 OPC UA PubSub

Since Part 14 (PubSub) was included in the OPC UA standard, a publish-subscribe architecture is also possible. This allows having a one-to-many communication where the message layout can be changed at runtime. The standard defines two different publish-subscribe communication models. The first is a broker-based model, where Subscribers and Publishers use a broker-based protocol like Message Queuing Telemetry Transport (MQTT) or Advanced Message Queuing Protocol (AMQP) for communication. The second option is a broker-less model using a UDP to transport Unified Architecture

Datagram Protocol (UADP) network messages with unicast or multicast Internet Protocol (IP) addresses. For multicast, the Subscriber registers itself to a multicast group to subscribe to a topic. The network will then distribute the messages to all Subscribers registered to the multicast group. Because of the multicast infrastructure, the broker-less method is generally limited to local networks [52]. Part 14 also defines a way to send messages directly on the data link layer of the OSI model, which sends messages without UDP or IP header [7].

A Publisher and the Client/Server model can also run on a single OPC UA server, as shown in Figure 2.2. Both get their data from the same address space, where all the data at runtime are stored. For the Publisher, the *PublishedDataSet* selects the data from the address space, and the *DataSetWriter* sends this data to a middleware like a multicast network or a broker. The middleware distributes the data to the according subscribers. For the Client/Server model, the Client establishes a connection to the Server, which transmits to the Client the requested data.

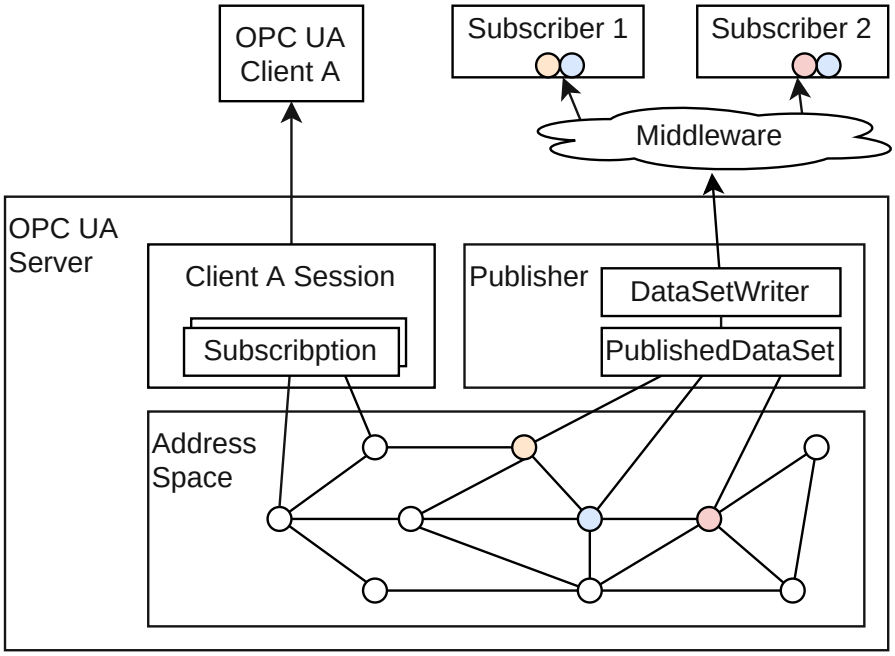


Figure 2.2: OPC UA address space modified from [6]

Structure of OPC UA PubSub Components

OPC UA PubSub defines multiple parameters for the single components, which are represented in the OPC UA address space. The parameters of those components can be modified in the address space, and new components can be added with OPC UA methods. The component and their parameters define how published data are encoded to a *NetworkMessages*. The component configuration needs to be known on the Subscriber

side to be able to decode the received *NetworkMessages*. As shown in Figure 2.3, the components of an OPC UA PubSub are organized as a tree. The top node is the *PublishSubscriber* component, which is available in the OPC UA address space if the server supports the PubSub functionality. All following described components are derived from this node and can be added with methods available in the OPC UA Information Model [7]:

- **DataSetMessage:** Can contain multiple dataset fields and additional header information like *DataSetWriterID*, Sequence number, Timestamp, Version and Status. A *DataSet* field contains the actual value of the *DataSet*.
- **TransportProtocol:** Includes the header from the used protocol like Ethernet, IP and UDP. The *TransportProtocol* is used to transmit a *NetworkMessage*.
- **NetworkMessage:** Is used as a container for *DataSetMessages* and includes the *PublisherID*, Security data and published fields.
- **PubSubConnection:** Contains the URL of the communication partner, the transport profile URI and the *PublisherID* to identify the messages on the Subscriber.
- **WriterGroup:** Can contain multiple *DataSetWriters*, a *WriterGroupID*, and the interval of publishing the *NetworkMessages*. Data in one WriterGroup are sent as a single *NetworkMessage*.
- **DataSetWriter:** Links to one dataset in the *PublishedDataSet* to define which data should be published.
- **PublishedDataSet:** Defines the content of a *DataSetMessage*, which can contain multiple *DataSetMessage* fields.

  It also defines the metadata of the DataSets, which is required for the interoperability between Publisher and Subscriber [7]. The *PublishedDataSet* on the publisher side also contains the information source for each field, which is implemented as a link to a node in the address space.
- **ReaderGroup:** Can contain multiple readers and a few common settings. In contrast to the *DataSetWriter*, it does not contain any IDs for filtering *NetworkMessages*.
- **DataSetReader:** Contains the parameters of the *PublisherID*, *DataSetWriterID* and the *WriterGroupID* for filtering the *NetworkMessages*.
- **SubscribedDataSet:** Contains the information required to decode a *DataSetMessage* and to write it in the address space.

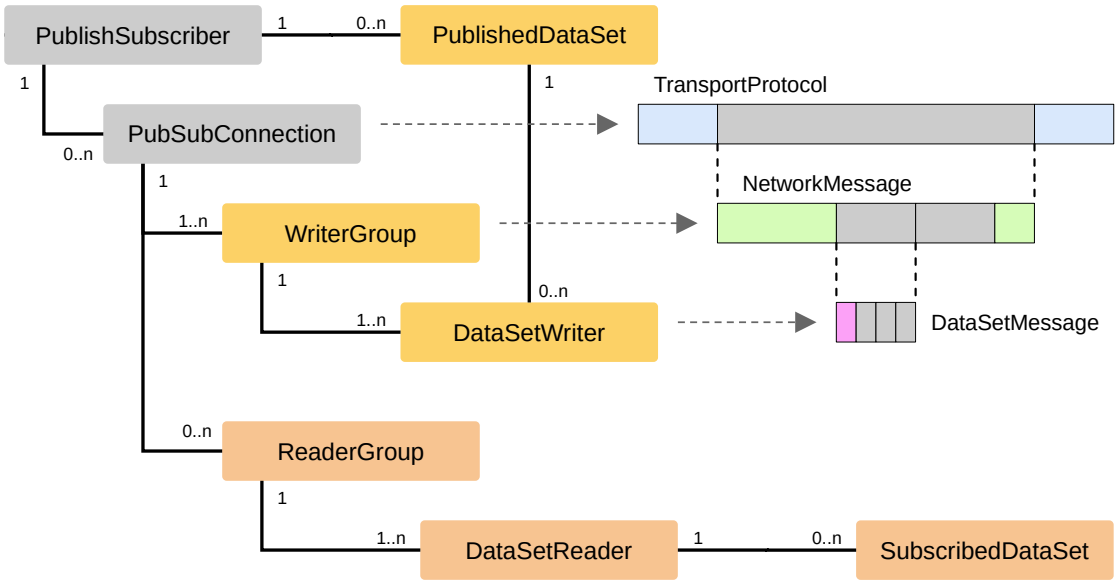


Figure 2.3: PubSub overview OPC UA [7]

PubSub States

OPC UA PubSub allows enabling and disabling components of a *PubSubConnection* with methods in the OPC UA Information Model. While a disabled Publisher stops publishing messages into the network, a disabled Subscriber will stop to overwrite values in the address space. If a component is disabled, it stays in the Disabled state, and an enabled and operational component stays in the Operational state. As shown in Figure 2.4, OPC UA Part 14 [7] defines in addition to the Operational, Disabled, and Error state, also a PreOperational and a Paused state. The last one is required because also sub-components of a *PubSubConnection*, like a *DataSetWriter*, can be enabled and disabled. If a sub-component is enabled, but the parent is disabled, the sub-component moves in the Pause state.

The second state, which is OPC UA PubSub specific, is the PreOperational state. A component stays in this state if enabled, but the necessary steps to enter the Operational state are currently in process.

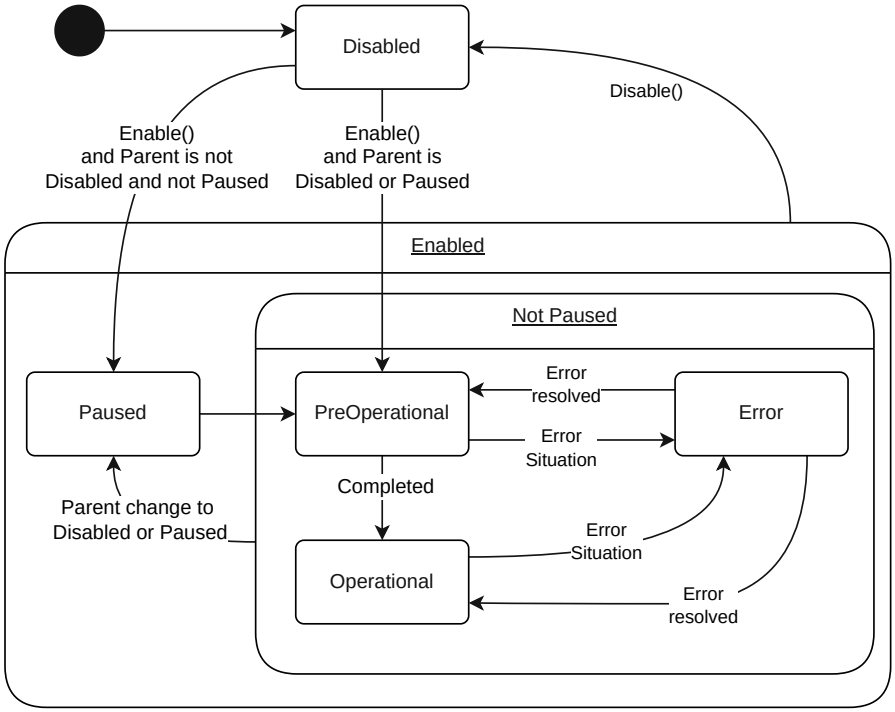


Figure 2.4: OPC UA PubSub state machine modified from [7]

2.1.2 OPC UA Safety

OPC UA Safety, which is defined in Part 15 of the specification, allows devices to use OPC UA to exchange safety-related data. OPC UA Safety provides safe communication mechanisms in an application-independent way. The safety application that utilizes these mechanisms over the Safety Application Program Interface (SAPI) is responsible for providing and evaluating data transmitted over OPC UA Safety. It also defines the structure and length of the sent data. The added Safety Communication Layer (SCL) uses unique MonitoringNumbers (MNRs) to identify SPDUs, timeouts, and cyclic redundancy code to detect communication errors on the underlying transport mechanisms. This standard offers a Probability of dangerous Failure per Hour (PFH) and a Probability of dangerous Failure on Demand (PFD) low enough to build safety-critical applications with a Safety Integrity Level (SIL) of up to SIL 4, while SIL is a measurement to classify the functional safety of a system [35].

An OPC UA Safety connection consists of two parts, the SafetyProvider, and the SafetyConsumer. The architecture of the communication stack that is used to exchange data between SafetyProvider and SafetyConsumer is shown in Figure 2.5. The SafetyProvider is the data source of a unidirectional safety link, and the SafetyConsumer is the sink. The SafetyConsumer is also responsible for error detection on received messages, and together with the SafetyProvider, they build the SCL. Even if the data flow is

unidirectional, the communication on a lower level uses a request/response pattern. This allows the SafetyConsumer to check the timeliness of the messages without a synchronized clock.

Below the SCL, the OPC UA layer is located, which can either use Client/Server or PubSub. For the Client/Server model, the SafetyProvider uses the server, and the SafetyConsumer uses the Client, which calls the methods provided by the SafetyProvider. If the SCL uses a PubSub model, the SafetyConsumer provides a RequestSafety Protocol Data Unit (SPDU) to a Publisher which sends it to the middleware. On the SafetyConsumer side, a Subscriber is subscribed to that messages and provides the received RequestSPDU to the SafetyProvider. The SafetyProvider generates then a ResponseSPDU and provide it to a Publisher which sends it to the middleware. A Subscriber on the SafetyConsumer side receives the ResponseSPDU and provides it to the SafetyConsumer. In order to handle packet loss to increase reliability, a mechanism is required to send packages multiple times. In the case of the OPC UA Safety Protocol, the Standard OPC UA layer is responsible for that by using a higher frequency to transfer the SPDUs. The SCL then reacts only to the first SPDU with the same MNR.

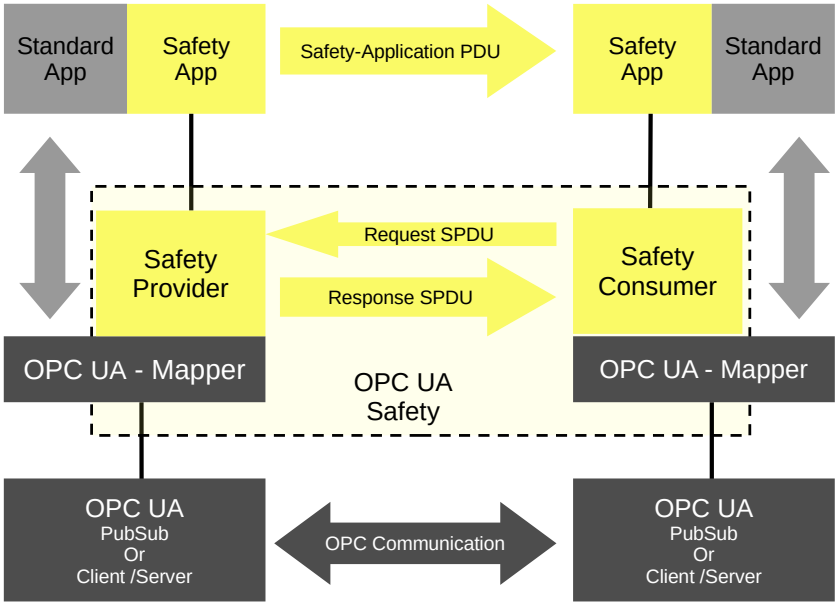


Figure 2.5: OPC UA Safety stack modified from [8]

Figure 2.6 shows the internal state machine of an OPC UA SafetyConsumer.

At first, the SafetyConsumer checks the parameters, such as SafetyProvider, and SafetyConsumer IDs, if they are not 0. If the SafetyConsumer is enabled and all configuration parameters are OK, it prepares a new RequestSPDU, which is transferred by the Standard OPC UA layer. After the SafetyConsumer receives a ResponseSPDU, the state machine leaves state S14. At state S15, the Cyclic Redundancy Check (CRC) is verified, followed

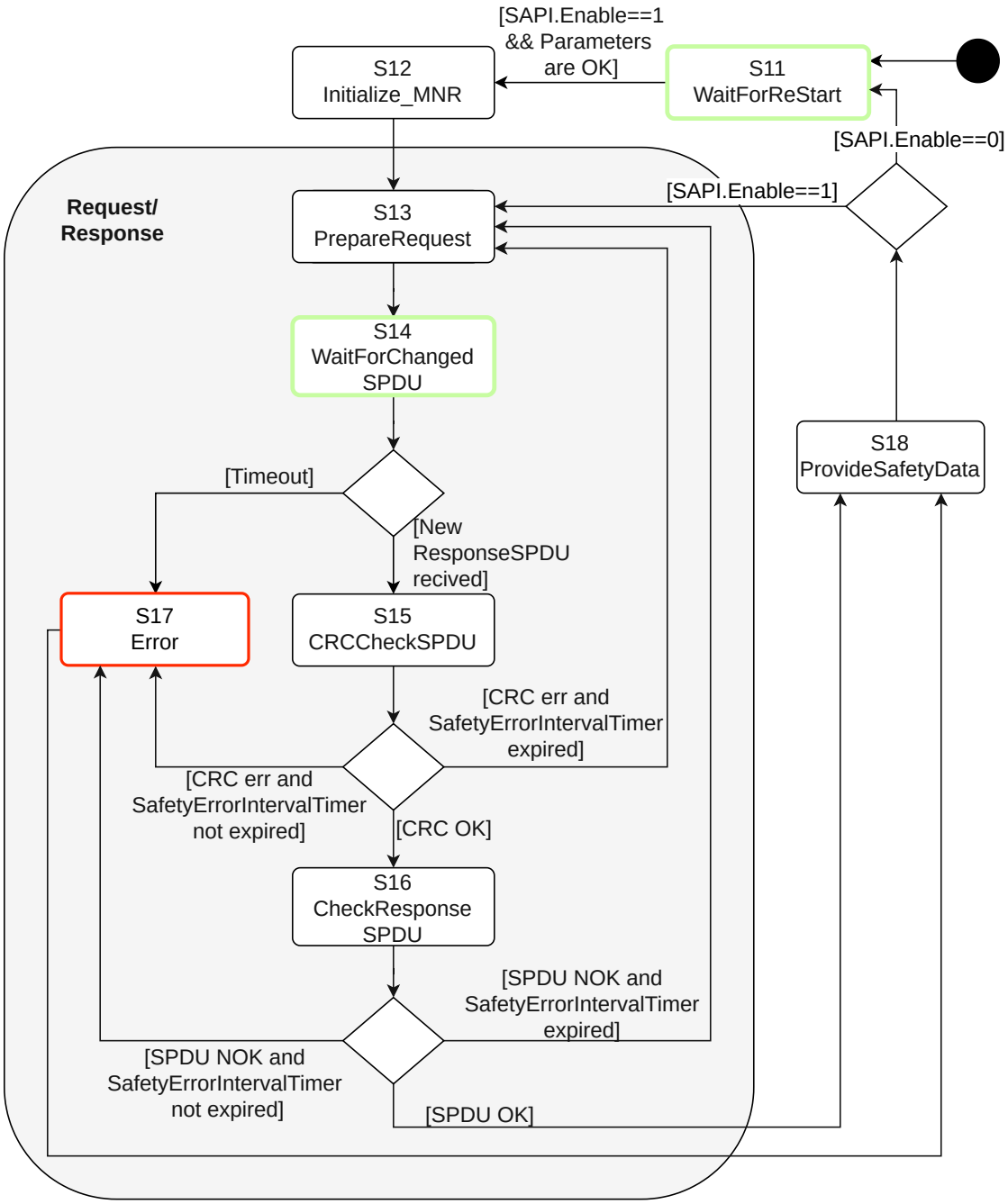


Figure 2.6: OPC UA safetyConsumer state machine modified from [8]

by a check of the SafetyConsumerID, the SPDU ID, and the MNR at state S16. If the checks at states S15 and S16 fail for the first time in a configured time interval, the request is repeated. Otherwise, the state machine goes into an error state. If all checks are passed, the received Safety Data is transferred to the safety application via the SAPI. Since the SafetyProvider only replays to requests from the SafetyConsumer, the state machine consists only of two states, namely the *WaitForRequest* and *PrepareSPDU*. Therefore, the state machine will not be shown in a separate figure [8].

As discussed in this section, OPC UA Safety uses the Standard OPC UA layer to transfer SPDUs between SafetyConsumer and SafetyProvider. Because OPC UA only covers the upper layers of the OSI model, a technology for the lower layers like TSN is required.

2.2 Time-Sensitive Networking

TSN is a set of standards, where each standard defines a specific functionality [25]. Table 2.1 gives an overview of some TSN standards, defined by the IEEE TSN task group [9]. Since TSN extends the IEEE 802 Ethernet standard, it is possible to send regular data like emails or browsing data via the same network as real-time data [38], like communication with an emergency stop button. For that, TSN sends the regular traffic as unscheduled traffic with best-effort and time-critical data as scheduled traffic.

To schedule the traffic, TSN uses Time Division Multiple Access (TDMA), which is defined in IEEE 802.1Qbv [11]. TDMA splits a communication channel into multiple time slots. These time slots can be used by different services or, in the case of TSN, for different queues. As shown in Figure 2.7, TSN IEEE 802.1Qbv uses eight queues per Ethernet port. The queue for each Ethernet frame will be chosen according to the Priority Code Point (PCP) value and the mapping in the device configuration. The PCP value is part of the IEEE 802.1Q header and allows to assign a certain priority to Ethernet frames.

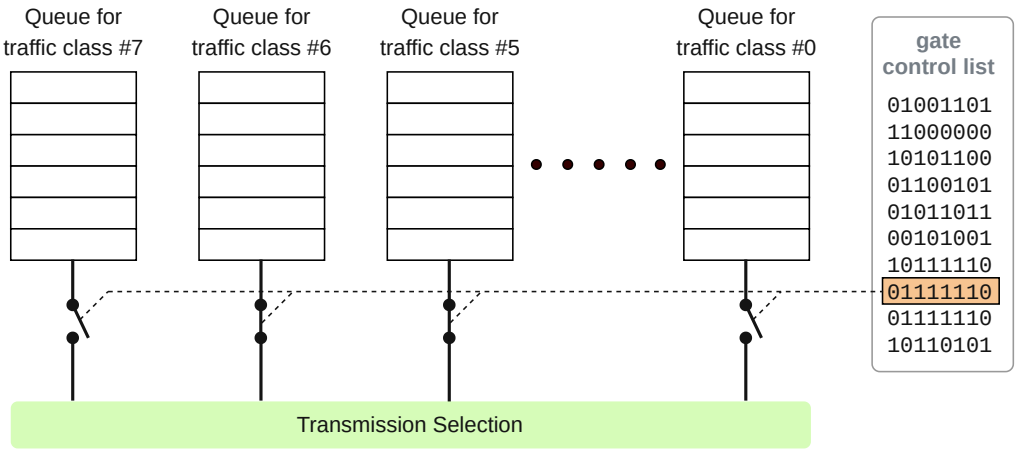


Figure 2.7: Transmission selection with gate control list modified from [11]

Each queue has a *Transmission Gate* where a Transmission Selection Algorithm selects frames from queues with an open gate for transmission. The gate defines if the frame from the queue can be selected for transmission. A *gate control list*, which is controlled by configuration, selects which gates are open or closed at a certain point in time. Because the *gate control list* is a finite list, which is processed in a loop, each queue has an open gate at a fixed interval. By setting the *gate control lists* on multiple TSN devices in a network accordingly, a stream between two end stations with guaranteed bandwidth can be configured.

Use	IEEE Std.	Description
Time Sync.	802.1AS	Time Synchronization
Bounded Low Latency	802.1Qav	Credit Based Shaper
	802.1Qbv	Time Scheduled Traffic
	802.1Qbu	Frame Preemption (also 802.3br)
	802.1Qch	Cyclic Queuing and Forwarding
	802.1Qcr	Asynchronous Traffic Shaping
Ultra Reliability	802.1CB	Seamless Redundancy, Stream Identification
	802.1Qci	Filtering and Policing
	802.1Qca	Path Control and Reservation
Resource Management	802.1Qcc	Stream Reservation Protocol Enhancements
	802.1Qcp	YANG Model for Bridging
	802.1Qcw	YANG Model for Qbv, Qbu, Qci
	802.1CBcv	YANG Model for CB

Table 2.1: TSN standards overview [47]

Another central part of the TSN specification is the time synchronization, which is used to synchronize the time between network devices and the end stations to reduce jitter and latency, as defined in IEEE 802.1AS [10]. The IEEE 802.1Qcc [12] (Stream Reservation Protocol) defines the management interface and administration of TSN networks. For configuration, it is necessary to obtain information about the end stations (Talkers and Listeners), like their resources, capabilities, and characteristics of associated streams.

To relate frames with the according stream, information like IP, MAC Address and Virtual Local Area Network (VLAN) tags are used [55]. The standard describes three different models for configuring TSN networks:

- **Fully distributed model**: The end stations (Talkers and Listeners) send their information to the network, and the network configures itself entirely. This means each bridge gets the requirements from the end stations, but they do not have information about the entire network.
- **Centralized network/distributed user model**: In contrast to the *fully distributed model*, it uses a Centralized Network Configuration (CNC), which knows all user requirements and capabilities of all bridges in the network. The CNC receives

all information from the bridges in the network and does not communicate with the end stations. The CNC computes the scheduling and configuration, which it then sends to the TSN bridges.

- **Fully centralized model**: In the two previous models, the configuration of the end stations was not addressed. Nevertheless, in some cases, it is also necessary to configure the TSN interface of end stations to fulfill complex timing requirements [33]. As shown in Figure 2.8, the fully centralized model includes a Centralized User Configuration (CUC) to communicate over an Operation Technology (OT) protocol with the end stations, to retrieve end station capabilities and requirements, and discover them. The CUC communicates over an Application Programming Interface (API) with the CNC to exchange the collected information [2]. The CNC will calculate the configuration and send it to the TSN bridges and the TSN end stations over a protocol like NETCONF, which is discussed in the following section.

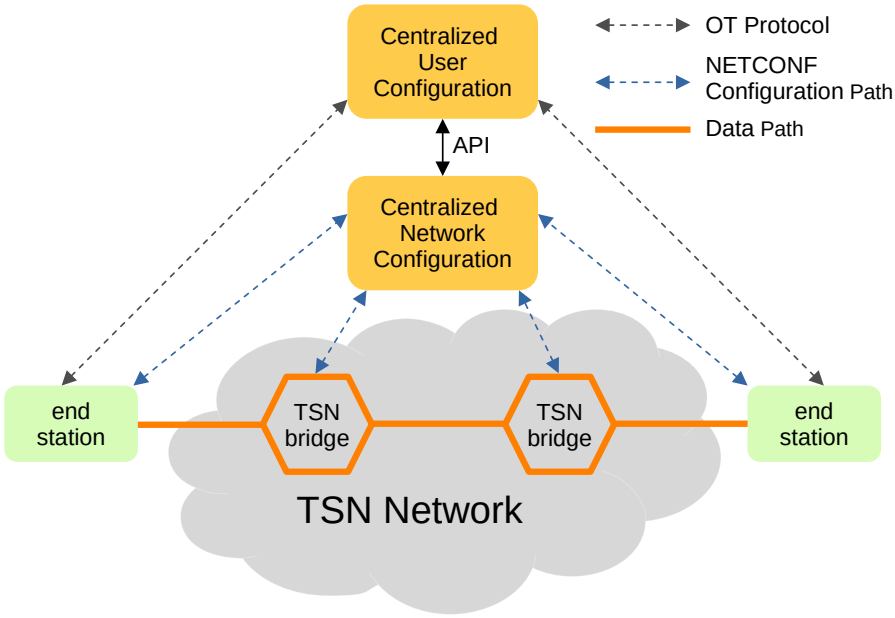


Figure 2.8: Fully centralized model from IEEE 802.1Qcc [12]

2.3 Network Configuration Protocol

For the configuration of TSN devices from different vendors, the NETCONF protocol [24] is available, which allows installing, manipulating, and deleting configurations on the network devices. NETCONF uses Remote Procedure Call (RPC)-based mechanisms to communicate between client and server, where the server is typically a network device, and the client a script or application [15].

Figure 2.9 shows a four-layer model of the NETCONF protocol:

- **Secure Transport:** NETCONF can use different transport protocols, in this work, only Secure Shell (SSH) is used.
- **Messages:** To send data, an RPC model is used, where the client sends a request with *<rpc>* and the server answers with a *<rpc-reply>*. The message layer uses an XML syntax. To have a relation between request and reply, an equal *message-id* is used for both messages. The NETCONF server can inform the client about an event via a *<notification>*.
- **Operations:** To modify data via NETCONF, a set of operations is defined, which are described in Section 2.3.1.
- **Content:** The content of the operations is defined in the YANG model (see Section 2.4), which is used on the NETCONF server.

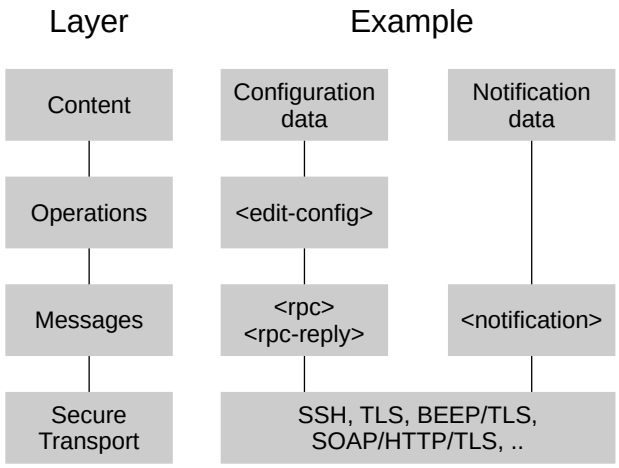


Figure 2.9: NETCONF layers [24]

2.3.1 NETCONF Datastores

Initially, NETCONF defined three datastores, which got extended by two read-only datastores in RFC 8342 [16]. The two additional datastores, *intended* and *operational* are not used and therefore not further discussed.

The *candidate* datastore, which is optional, can be modified without effecting the network device. After the modification, the configuration in *candidate* can be committed to *running*, which holds the current configuration. *Running* can also be directly modified, but it must always contain a valid configuration. The *startup* datastore is non-volatile storage and is optional. If a *startup* datastore is used, the NETCONF server copies the configuration from the *startup* to the *running* datastore when the NETCONF server boots. In case the *startup* datastore is not used, the configuration in the *running* datastore is used when the NETCONF server boots [16].

2.3.2 NETCONF Operations

As also showen in Figure 2.10, the client can invoke the following operations on the NETCONF server [24]:

- **get-config:** Request the configuration of a datastore which needs to be defined as a parameter. Additionally, a filter can be used to request only a sub-configuration.
- **edit-config:** Load all or part of the configuration to the specified datastore. Further one of the following options for editing the configuration on the network device can be chosen: *merge*(default), *replace*, *create*, *delete* or *remove*.
- **copy-config:** Copy a configuration from one datastore to another. The configuration will be replaced or created if it does not exist.
- **delete-config:** Delete a configuration from a datastore. The running datastore can be not deleted.
- **lock:** Lock a datastore so that other clients can not modify the configuration in a datastore.
- **unlock:** Unlock a previous locked datastore.
- **get:** Request the running configuration and the device state information.
- **close-session:** Request a graceful termination of the NETCONF session.
- **kill-session:** Force the termination of the NETCONF session.

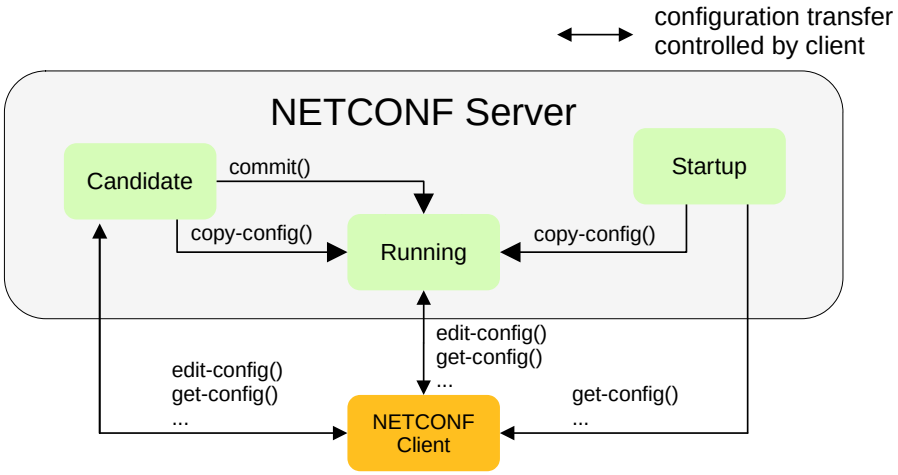


Figure 2.10: NETCONF datastore modified from [16]

2.4 YANG

NETCONF defines the interface to modify a configuration on a device. It does not define how the configuration is structured, therefore YANG is used to define a model of a configuration. YANG is a data modeling language used to model network device configuration and state data with a NETCONF interface. This data model defines the schema of the stored data, similar to a class in an object-oriented programming language. YANG models organize the data in a tree, where each node has either a value or a set of child nodes [15]. An example of a simple YANG model can be found in Listing 2.1. A network device can have multiple YANG models. After a client has connected to a NETCONF server, the server informs the client about its used YANG models in a hello message.

YANG is structured in modules, where each module has its own namespace. The namespace needs to be defined with a unique string at the beginning of a module. To simplify the structure of a module, it can include multiple submodules. In order to keep track of the modification history, each module should have a revision statement for the initial revision, and each modification should be described with the modification date and a detailed description. To restrict the values that can be stored, a new datatype can be defined with the *typedef* statement. An example of that would be a type IP address, where four values between 0 and 255 separated with a point must be provided in order for YANG accepts a change of an IP address. In case a set of nodes is used frequently, a group can be defined with the *grouping* statement. An example would be a timestamp consisting of a second and a nanosecond value. This timestamp can be included in the model with the *usage* statement instead of having two leafs.

The actual data are stored in *leaf* nodes, which must contain a *type* statement to define the data type of the stored value and cannot contain any child nodes. To structure the data, there are containers, *leaf-list* and *lists* available. A *container* groups related nodes, has only child nodes, no values [15], and it is similar to a folder in a file system. With a *leaf-list* multiple values with the same data type can be stored. In contrast to the *leaf-list*, the *list* statement provides the possibility to store a list of datasets with different datatypes. A *list* has a key value, which is one element of the dataset and needs to be unique in the *list* to distinguish the different datasets. A *list* is similar to a list of structs in the C++ programming language.

YANG provides the possibility to define RPCs with input and output parameters, which can be called via NETCONF. For devices with optional features, YANG gives the possibility to define specific parts of the model as optional *features*. The device controls if this feature is available or not.

```
module simpleSwitch {
    namespace "urn:tuwien:interface";
    prefix tu;

    revision 2022-08-22 {
        description
          "initial revision";
    }

    typedef if-number {
        description
          "range of interface numbers that are accepted";
        type uint8 {
            range "0..48";
        }
    }

    grouping timestamp {
        leaf seconds-field {
            type uint64 {
                range "0..281474976710655";
            }
        }
        leaf nanoseconds-field {
            type uint32;
        }
    }

    container systemtime {
        uses tu:timestamp;
    }

    container interfaces {
        list interfaces {
            key number;
            leaf number {
                type if-number;
            }
            leaf enable {
                type boolean;
                default false;
            }
        }
    }
}
```

Listing 2.1: Simple YANG model of a switch

2.5 System Configuration in IT and OT

OPC UA and TSN are designed to be interoperable, so two devices from different vendors can easily communicate with each other. To make changes in the safety system, a configuration must be deployed to TSN bridges, TSN end stations, safety sensors, safety actuators, and the safety logic. An engineering tool from the hardware vendor must be used to modify configurations and deploy them. In the worst case, for each TSN and each OPC UA device, a different tool is necessary, which are mostly working just under Windows.
For example, to modify and deploy the configuration of an OPC UA server, it is necessary to use the *TIA Portal* for Siemens devices, *Automation Studio* for B&R devices or the *PNOZmulti Configurator* for Pilz devices. With NETCONF, manufacturers try to use a common standard for the configuration of TSN devices. For that, the YANG models are standardized, which allows sending a configuration XML via NETCONF to a TSN device, independent of the vendor.

In the IT world, though, deployment of configurations is already more advanced than in the OT world. An example is Ansible, which can be used for software deployment and configuration management. The user creates a YAML file, which is called Playbook. With the Playbook, a task can be defined which is then performed on the remote machines. Ansible will then establish an SSH connection to the remote hosts and run a python script simultaneously, to perform the actions according to the Playbook [34]. A second example is the Simple Network Management Protocol (SNMP) introduced in 1988, which is the predecessor of NETCONF. It was designed to simplify the management of bigger networks and allow remote management of devices. However, the main focus of SNMP is to monitor network components. Nevertheless, there are still massive problems with interoperability in the IT world. Especially network devices often require customized Command-Line Interfaces (CLIs) and tools, instead of using open protocols like NETCONF [46]. In this work, a tool will be shown, that covers the configuration on all three layers of an RSS, namely the TSN, OPC UA PubSub, and the OPC UA Safety layer.

CHAPTER 3

Related Work

There are many works available in terms of the configuration of flexible systems, which propose approaches on how to configure TSN, combine TSN with other technologies like OPC UA and NETCONF, or use frameworks to get an RSS. So far, there is no work publicly available where OPC UA Safety (Part 15) is used together with TSN since there is no stack publicly available that implements OPC UA Safety. This chapter gives an overview of different approaches, concepts, and protocols.

3.1 Existing Safety Protocols

For a real-time network infrastructure based on Ethernet, there are multiple protocols available, which often come with their own safety protocol. PROFINET [59], which is an Industrial Ethernet (IE) solution for industrial automation comes with PROFIsafe or EtherCAT with Safety-over-EtherCAT [53]. OpenSAFETY is an open-source standard for transferring safety-critical data over a deterministic network like TSN [9][3]. Gent et al. [28] prove the concept that the combination of OpenSAFETY with TSN is suitable to transport time-critical safety data and non-critical traffic over one Ethernet channel. In this master thesis, the new OPC UA Safety standard together with OPC UA PubSub and TSN is used. Table 3.1 gives an overview of a few IE protocols and their corresponding safety protocols, which are specified in IEC 81784-3.

3.2 Frameworks and Concepts

The selection of a configuration for an RMS is one of the key issues to using RMS efficiently. Ashraf and Hasan [14] introduce a framework based on a sorting algorithm using reconfigurability, operational capability, reconfiguration cost, and reliability of the involved machines and tools. With the use of this framework and a mathematical model

IE	Safety Protocol	IEC NORM	ORG
EtherNet/IP	CIP Safety	61784-3-2	ODVA
PROFINET	PROFIsafe	61784-3-3	PNO
EtherCAT	Safety-over-EtherCAT	61784-3-12	ETG
Powerlink	openSAFETY	61784-3-13	EPSG
RAPIEnet	RAPIEnet Safety	61784-3-17	RAPIEnet Association
SafetyNET p	SafetyNET p	61784-3-18	SNI

Table 3.1: Ethernet and safety protocols modified from [25]

of the optimization problem, an optimal selection of reconfigurable machine tools for an RMS can be found.

Another challenge arises from the need for flexibility for safety-critical systems. A self-configuring safety network is presented by Etz et al. [25]. The authors presented the requirements and a concept of self-configuring safety networks in that work. The proposed idea includes three technologies, where each of them fulfills a set of the introduced requirements. For the network infrastructure, they use TSN since it has the ability to send real-time and non-realtime data over the network. As communication layer, OPC UA is used, and as a safety layer openSafety. The reconfiguration of a safety network consists of the three listed technologies can be split into four phases. At first, new or removed Safety Nodes need to be discovered via OPC UA, followed by a validation phase, where the safety configuration of a new device needs to be validated. After that, the configuration is checked for plausibility issues, like device-matching. If these three phases are successful, the processing phase can be entered, where the Safety Nodes start to transmit safety-related process data, which concludes the reconfiguration process.

In an additional paper, Etz et al. [27] extended the framework to increase flexibility and the support for the safety engineer as shown in Figure 3.1. This framework contains, in contrast to [25], a Knowledge-Based System (KBS), which gives the possibility to generate safety configurations automatically. The generated configuration can be checked and validated with the help of the KBS. The framework also considers an automatic deployment, which is the last step before the RMS runs with its new configuration. The automatic deployment is not further discussed in that work. Nevertheless it is a critical topic to reconfigure the three different systems, namely TSN, OPC UA, and OPC UA Safety. Therefore, this thesis continues with this part of combining OPC UA with TSN to reconfigure a safety system.

3.3 OPC UA over TSN

To transfer time-critical data over OPC UA, a network is needed that can transport data with a guaranteed delay. Since TSN is an open standard that is designed for that purpose, it is often used to combine these two technologies. Therefore, the OPC

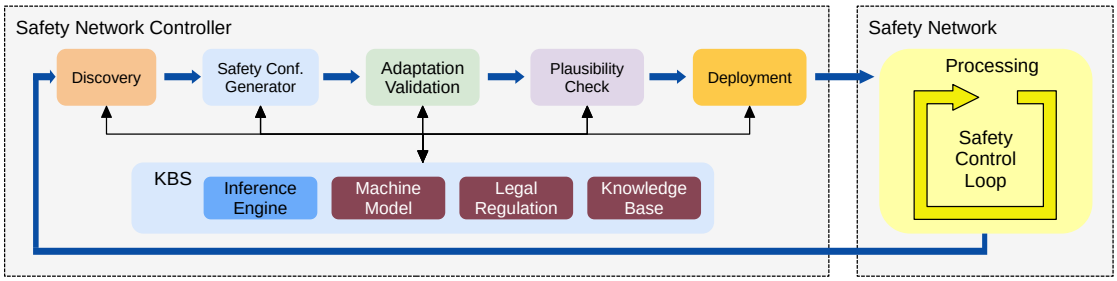


Figure 3.1: Self-organizing safety system model modified from [27]

Foundation is working on the new Open Platform Communications Field eXchange (OPCFX) standard, where the combination of TSN and OPC UA is defined [5]. An overview of combining OPC UA with TSN give Bruckner et al. [19] in their work. At first, they give a detailed background for OPC UA and the different TSN standards. Then, the authors show how these two technologies can be combined to fulfill the requirements for industrial automation. At the end of their work, they discuss the known issues of the two technologies and what still needs to be done so that the combination of these two technologies can become a competitive successor to fieldbuses.

Yuting et al. [43] show the implementation of OPC UA on a TSN backbone and evaluate its performance. As shown in Figure 3.2, the architecture consists of three layers: a Factory Cloud Layer for the management, an edge Layer with an OPC UA server for aggregating the data from the field, and a Field Layer containing the manufacturing facilities with one OPC UA server each. The communication inside manufacturing facilities can happen with different protocols, like EtherCAT or Powerlink. The communication between the three layers, though, is realized with an OPC UA Client/Server model over a TSN network. With this layer structure, the management system accesses the factory level over a single point, namely the OPC UA aggregating server, which reduces the complexity of connections. For their performance evaluation, they used only their Field Layer with TSN, where a Talker and a Listener communicate over two TSN switches. To get some reference values, they compared the end-to-end latency of a TSN communication with the latency of “normal“ Ethernet communication. The evaluation shows that the “normal“ Ethernet connection is faster than the TSN connection, if there is no load on the network. This is because the TSN traffic has always fixed intervals independently from the network load. As soon the network gets floated with User Datagram Protocol (UDP) traffic, the latency of the “normal“ Ethernet connection increases rapidly. The traffic which uses TSN streams, though, is not affected at all by the load of the network.

3.4 Autoconfiguration of OPC UA and TSN

Gutierrez-Guerrero and Holgado-Terriza [32] proposes a mechanism for auto-configuration of OPC UA systems. The goal of this mechanism is to get a Plug-and-Produce (PnP) system to avoid the need for a manual configuration of a newly added PLC. In their

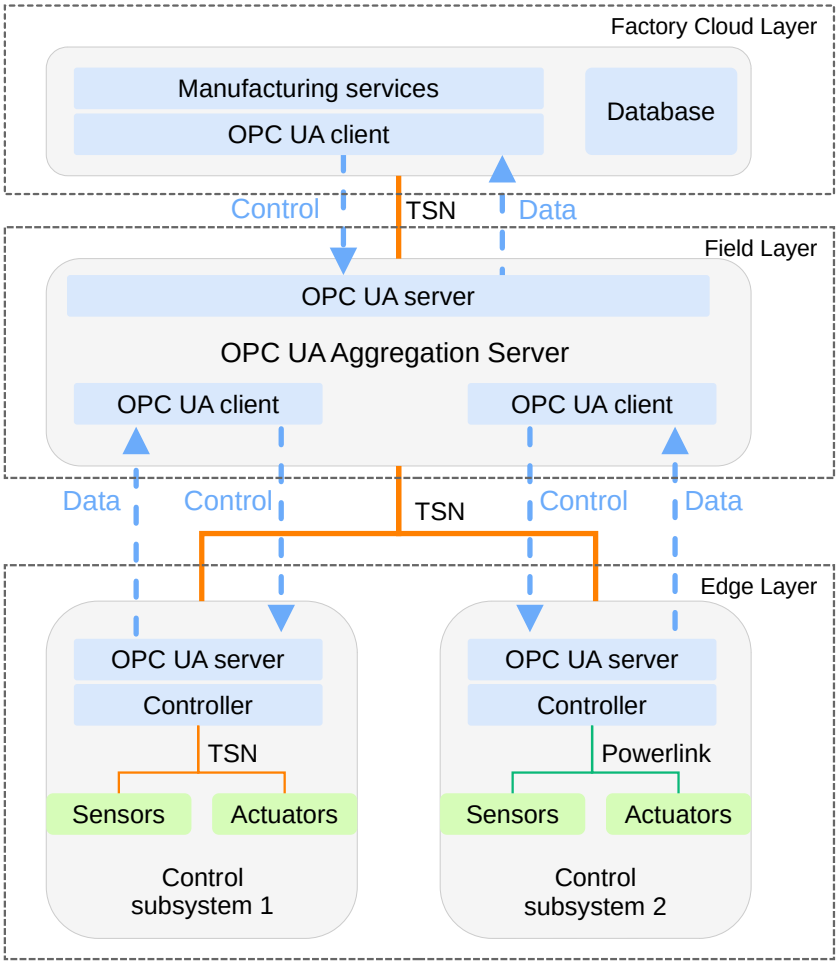


Figure 3.2: OPC UA TSN communication architecture [43]

approach, only one OPC UA server is used, which contains the data from all PLCs in the network. To identify new devices in the network, a central service regularly sends multicast messages to the network. A newly added device replies with its Unicast IP. This IP is used to query the meta-information of the PLC, which includes information about its process variables provided via Modbus. With the gained meta-information, a new Information Model is created on the OPC UA server. The OPC UA server synchronizes its variables via Modbus with the PLCs, and clients can then interact only with the nodes provided by the OPC UA server.

In contrast to the central solution of the previous paragraph, Liu and Bellot [44] show a similar approach to the approach in this work, but without a configuration of the network. All the devices are equipped with an OPC UA Publisher or Subscriber, which can communicate via an MQTT broker with each other. The introduced configuration tool

provides a Graphical User Interface (GUI) to modify or add new OPC UA connections between Publisher and Subscriber and change the address space on both sides accordingly.

Pahlevan et al. [51] use a framework [50] to simulate a TSN in combination with the Rapid Spanning Tree Protocol (RSTP). Switches are simulated with queuing and scheduling mechanisms and end stations with configured time-triggered messages. The streams were defined by different parameters like source port, transmission/reception window, and VLAN ID. The authors use a fully centralized configuration model and NETCONF for the remote network management. The proposed central unit, which acts as CNC and CUC, detects topology changes in the network and configures TSN nodes if needed. If the RSTP changes the tree in the network, the central unit detects the update, calculates new streams, and deploys the new configuration via NETCONF. For evaluation, they use a fault model which simulates different faults on Ethernet links between two switches.

3.5 TSN configuration via OPC UA

Since OPC UA is already widely used in the automation area, there are some works showing the usage of it also for TSN configuration changes. Zhou and Shou [61] show a scheme where the TSN Talker and the Listener send service requests to the CUC over the OPC UA protocol. The CUC forwards the request to the CNC, which calculates the routing and scheduling for the TSN network. The CNC then sends the configuration to the TSN bridges via NETCONF, and additionally, it sends the configuration to the CUC, which sends it to the Talker and Listener. The end stations can then configure themselves with the received information. Tian and Shou [57] also use OPC UA for the communication between CUC and the end stations, but with a focus on giving a general overview of the used technologies.

Kobzan et al. [37] also use OPC UA for the communication with the end stations, similar to [61] and [57]. The focus there, though, is on the dynamic detection of new TSN devices, followed by the configuration of the devices and the end stations. For that, a central controller constantly tries to ping the IP addresses of the components to detect new devices. If the central controller detects a new device, it gathers information about it from the OPC UA server, which is running on all devices. This information is used to create a configuration for the TSN nodes, which are deployed by the Software Defined Networking (SDN)-controller via NETCONF.

3.6 TSN combined with SDN

SDN is a network management paradigm for managing complex IT networks. Figure 3.3 shows a simplified SDN structure. The bridges in the data plane communicate with SDN over the SouthBound Interface (SBI). The SDN controller in the control plane defines the traffic controls rules and performs administration control of the network devices. The user application, which runs on the application plane, requests network services over the NorthBound from the SDN controller [55].

Siva et al. [55] use SDN to deploy the configuration to the TSN network and also give a comprehensive overview of both technologies. The target of that work is to evaluate SDN and TSN in the context of Industry 4.0, focusing on different requirements like real-time performance, overhead, and feasibility. In contrast to [37], Siva et al. use OpenFlow instead of NETCONF as SBI, which is currently the *de facto* standard SBI protocol [55]. A comparison between NETCONF and OpenFlow as SBI can be found at [40]. However, OpenFlow and SDN are out of this work's scope and will be therefore not further discussed.

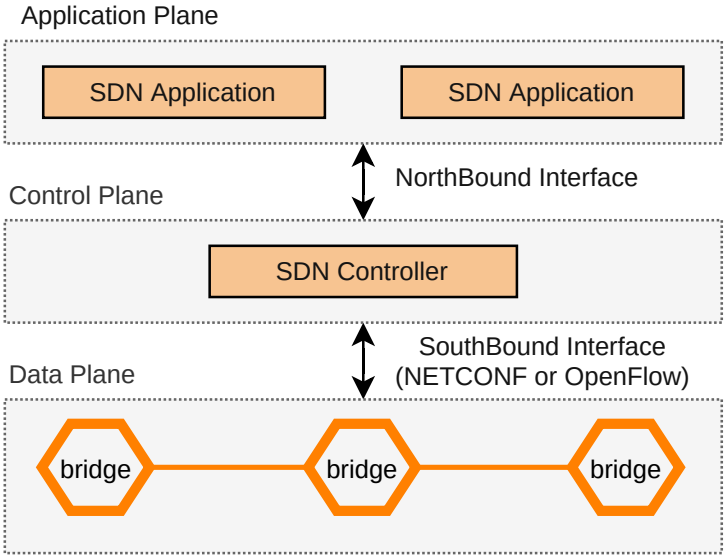


Figure 3.3: SDN overview

CHAPTER 4

Configuration Approach

In order to change the configuration of an RSS, as it is defined in [25] and shown in Figure 3.1, a tool is needed that can make configuration changes on three layers of a safety system, the TSN, the OPC UA PubSub, and the OPC UA Safety Layer. This chapter specifies the use cases that this tool should cover and shows how the three layers communicate in an RSS. At the end of this chapter, the required parameter are discussed and the architecture of an RSS with a central deployment tool is shown.

4.1 Motivating Use Cases

As discussed in Section 2.5, reconfiguring a safety system with current tools requires considerable engineering effort. Additionally, it is impossible to modify the safety system during runtime. This deployment tool should reduce the effort to reconfigure a safety system and increase flexibility. Figure 4.1 shows an overview of use cases and requirements to deploy a configuration on an RSS. We defined three use cases that cover the key features of a deployment tool in an RSS, the *initial commissioning*, *add new safety device*, and *remove safety device*. Even if the configuration needs to be changed entirely during runtime, it can be split in those three cases. Therefore, they are briefly discussed in the following sections.

4.1.1 Initial commissioning

The initial commissioning is the first use case, which is covered by current configuration tools. All safety-critical components need to be configured during the commissioning of a new system. This includes the network, which transports the safety data, the safety sensors, the safety logic, and the safety actuators. This process should happen only with one click without knowing the manufacturer of the used devices or knowledge of the used technologies. As shown in Figure 4.1, *Initial commissioning* is used by a *Safety*

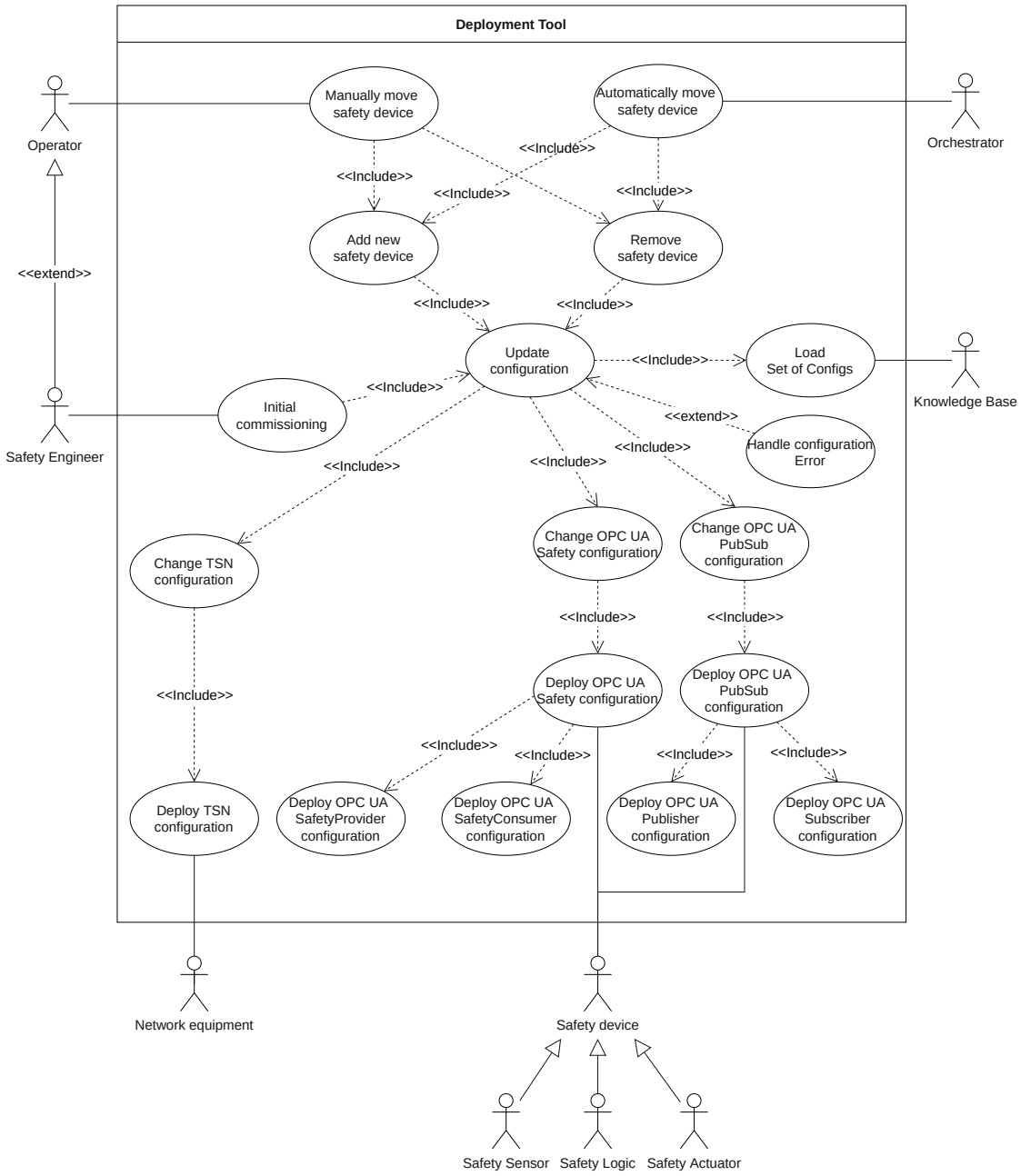


Figure 4.1: Use case diagram

Engineer and uses the function of the *Update configuration* use case. However, this is a simple special case because no old device configurations must be considered. For that, the configuration is loaded with *Load set of configs* from the KBS (cf. Figure 3.1). The *Update configuration* uses functions of *Change TSN*, *OPC UA Safety*, and *OPC UA PubSub* use cases. The *Change TSN configuration* use case covers the management

of the configuration activation and deploys new configurations with the *Deploy TSN configuration* use case.

Change OPC UA Safety configuration and *Change OPC UAPubSub configuration* remove old configurations, enable and disable OPC UA PubSub functionalities, and deploy new configurations with the function of *Deploy OPC UA Safety configuration* and *Deploy OPC UA PubSub configuration*. The *Deploy OPC UA Safety configuration* distinguishes between a deployment on a SafetyProvider and a SafetyConsumer. Similar to that, *Deploy OPC UA PubSub configuration* distinguishes between deploying a configuration on a Publisher and a Subscriber. Nevertheless, errors that occur at *Update configuration* are handled with *Handle configuration Error*.

4.1.2 Add new Safety Device

In an RSS, it can be necessary that a new safety device must be added during runtime. An example would be multiple machines sharing one mobile user interface with an emergency button. When the operator connects the user interface to the machine, the emergency stop button must be integrated into the safety system. With current technologies, a safety engineer must stop the whole manufacturing system to integrate the emergency stop button. As shown in Figure 4.1, this use case is used if the operator uses the *Manually move safety device* functionality. The operator must be able to do that in a simple way without stopping the manufacturing system. Additionally, *Automatically move safety device*, which is used by an *Orchestrator*, uses the function of *Add new safety device*. That again, uses the functionality of *Update configuration*, like *Remove safety device*.

4.1.3 Remove Safety Device

Similar to Section 4.1.2, removing a safety device from an RSS is sometimes required. An example would be a robot with an overlapping area with humans. Depending on the product, the robot operates in areas where humans usually work. If the robot is in a mode that cannot reach the common area, the safety device, which stops the robot if a human enters the common area, must be removed from the safety system. This logical removal has to be done automatically on the configuration change, without stopping the manufacturing system.

4.2 Safety Communication Stack

As discussed in Chapter 3.1, most of the IE protocols come with their own safety protocol. These protocols are mainly proprietary, and different manufacturers use different protocols. OPC UA is already widely used by many manufacturers in the industry for interoperable communication. Therefore, we expect that the new OPC UA Safety standard will also find similar acceptance like the OPC UA standard. For that reason, we decided to use OPC UA in combination with OPC UA Safety. On the OPC UA layer, there is the possibility of a Client/Server and a Publish/Subscribe model, as discussed in

Section 2.1.2. In this work, we use the Publish/Subscribe model because it works with UDP packages. The UDP traffic brings the possibility of calculating the amount of critical traffic beforehand because there are no handshakes and retry mechanisms involved. Since TSN has the advantage of mixing non-critical and critical traffic and it is independent of any hardware manufacturer, TSN will be used in this work.

Figure 4.2 shows the stack consisting of the three layers. The safety sensor on the left side transmits its data to a safety logic, and the safety logic performs some logical operation and transmits its result to the safety actuator. Those three safety components use the black channel principle to transport their data. This means that the components on the safety layer can use a nondeterministic channel to transport the data by using mechanisms to detect a missing or delayed message. OPC UA Safety does not transport any data by itself, this is done by the OPC UA PubSub layer, which is located below the OPC UA Safety Stack. OPC UA PubSub uses the SPDUs and sends them at a fixed interval to the communication partner over the TSN network. The TSN network has for each Publish Subscriber a stream reserved. This guarantees that the safety data arrives on time at its destination, and the RSS can work reliably.

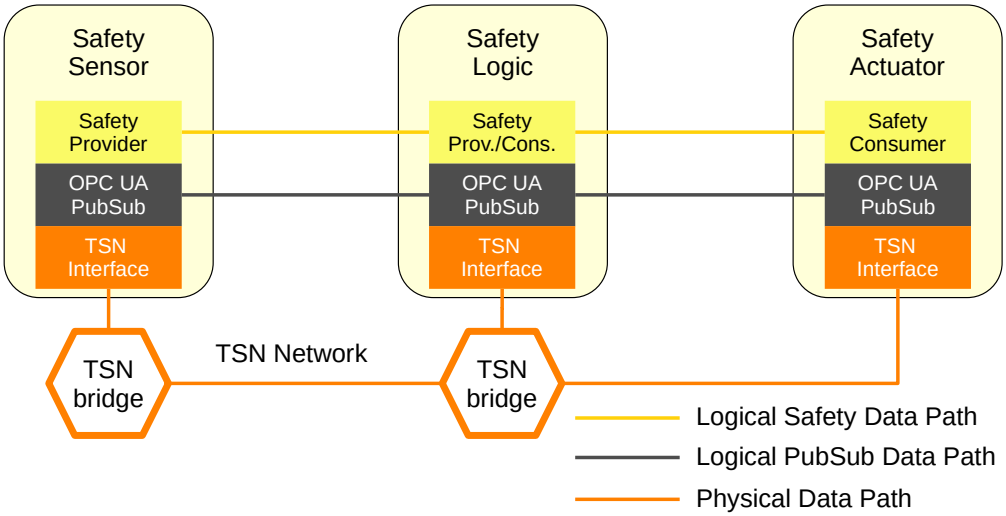


Figure 4.2: Three layer safety stack

4.3 Safety Data Exchange

To get detailed knowledge about the function of the presented safety stack in Section 4.2, the messages that are exchanged at runtime are discussed in this section. As shown in Figure 4.3, the communication consists of three layers. At first, a SafetyConsumer creates a RequestSPDU with a SafetyConsumerID and an MNR to identify the SPDU. The OPC UA PubSub layer takes the SPDU from the OPC UA address space and sends it in constant intervals as a UDP package. When the packet enters the TSN network, the first TSN device adds a VLAN tag with the according PCP. The TSN network can identify the

package with the VLAN tag and schedule it accordingly. On the SafetyProvider side, the Subscriber receives the UDP packet and updates the RequestSPDU on the address space. The SafetyProvider takes the information from the RequestSPDU and puts it together with the corresponding safety data from the safety application in a ResponseSPDU. The Publisher then sends the ResponseSPDU to the SafetyConsumer, as it was done with the RequestSPDU. The SafetyConsumer forwards the safety data to the safety application via the SAPI.

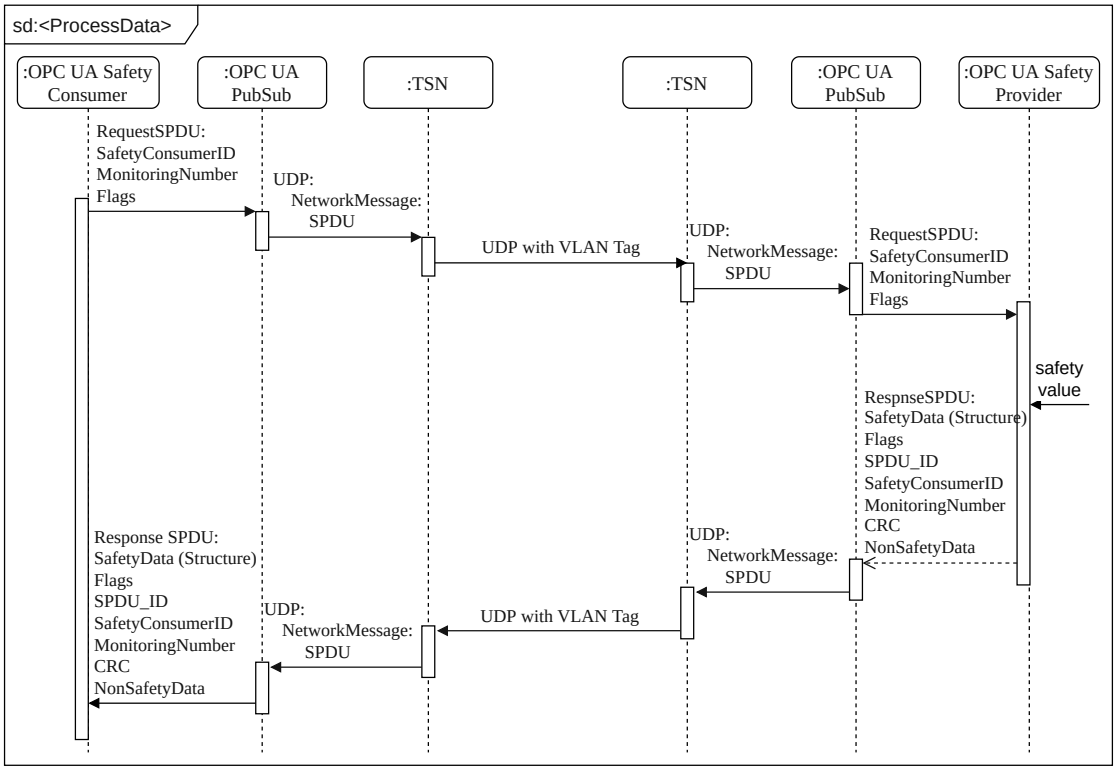


Figure 4.3: Process data of safety devices

4.4 Required Parameters

The deployment tool needs to configure a safety connection on three different levels. The TSN, OPC UA PubSub, and the OPC UA Safety layer. With the information from Section 4.3, we can create a list of parameters, which will be required to configure such a safety connection.

4.4.1 TSN

A TSN node must be configured on three levels. One for the time synchronization so that the cycles for the traffic scheduling run synchronously. The next one sets some basic

settings on the bridge, and the last one is the setup of the interfaces, which also includes the configuration for the traffic scheduling. The parameters listed in this subsection are only the ones required by the deployment tool, and not all parameters that are defined in the YANG models. Besides the parameters described in this subsection, also the IP address, username, and password are required to access the TSN device via the NETCONF interface.

802.1AS

All TSN network devices need a time synchronization to schedule the data traffic. For this work, the *ietf-gptp@2018-03-28.yang* YANG model is used. The deployment tool needs the parameters shown in Table 4.1 to configure some basic Generalized Precision Time Protocol (gPTP) parameters. Table 4.2 shows the required parameters to configure a GPTP port. If the propagation to the link neighbor exceeds the *neighborPropDelayThresh* from Table 4.2, it is assumed that a buffering repeater is in the link. Therefore, this port is ignored for the time-synchronization. The format in which this value is represented is defined in Table 4.3. The announce message is used to establish the synchronization hierarchy. The *initialLogAnnounceInterval* defines the initial interval for publishing the announced messages. The value is given in log_2, which means $0 \rightarrow 1s$. The last parameter defines the initial interval of sending synchronization messages.

Parameter	Description	Type
priority	Priority for the grandmaster selection	two uint32
PortDataSet	A list of settings for the ports of the bridge	List of structs according to Table 4.2

Table 4.1: Parameters for TSN 802.1AS of one bride

Parameter	Description	Type
portNumber	The number of the port, for which this set of parameters should apply	uint32
neighborPropDelayThresh	Propagation time threshold, above which the port is not considered for the time-synchronization	struct according to Table 4.3
initialLogAnnounceInterval	Initial value of the announce interval to exchange clock information	int32 $[log_2(s)]$ $(-128...127)$
initialLogSyncInterval	Initial value of the time-synchronization transmission interval	int32 $[log_2(s)]$ $(-128...127)$

Table 4.2: Struct PortDataSet

Parameter	Description	Type
Hs	The most significant 32 bits of the time value	uint32 ($2^{-16}ns * 2^{64}$)
Ms	The second most sign. 32 bits of the time value	uint32 ($2^{-16}ns * 2^{32}$)
Ls	The least significant 32 bits of the time value	uint32 ($2^{-16}ns$)

Table 4.3: Struct TSN time

Bridge

The bridge has a configuration that needs to be set up at the initial commissioning. Table 4.4 shows the required parameters. The *name* identifies the bridge, and it is used by the interface to link the bridge and interface. The *address* defines a MAC address, which is used for the Spanning Tree Protocol (STP), RSTP, and Multiple Spanning Tree Protocol (MSTP), in case one of these protocols is used. According to Table 4.5, the *component* parameter defines the components which the bridge comprises. Each component has its name and a unique MAC address. To use VLANs on a bridge, they must be defined in the component according to Table 4.6. Each interface links to one of these components; therefore, only VLANs defined in this component can be used.

Parameter	Description	Type
name	Name of the bridge	string
address	MAC address, which the bridge should use for STP, RSTP and MSTP	string (mac address)
component	List of components	List of structs according to Table 4.5

Table 4.4: Parameters for one TSN bridge

Parameter	Description	Type
name	Name of the component	string
id	Unique ID of the component	uint32
type	Type of the component (e.g. c-vlan-component, edge-relay-component, etc.)	string (type-of-component)
address	Unique MAC address	string (mac address)
bridge-vlan	A list of VLAN IDs, which are used at the bridge	List of structs according to Table 4.6

Table 4.5: Struct Component

Interface

Each interface has parameters to set some basic configurations and some parameters to set up the TSN gate control list. To configure the standard functions of an interface, the

Parameter	Description	Type
vid	ID of the VLAN	uint32
name	name of the VLAN	string

Table 4.6: Struct VLAN

parameters according to Table 4.7 are required. A unique *name* identifies the interface and the *description* allows a textual description of the interface. With the *enabled* parameter, the interface's state can be set. As described in Section 4.4.1, each interface links to a bridge component. This link must be set with the *component-name*. The *pvid* parameter sets the VLAN ID of the tag, which is added if untagged frames arrive at the port.

Parameter	Description	Type
name	A name for the interface	string
description	A description of the interface	string
enabled	Enables the interface	boolean
bridge-name	Name that links to the name of a bridge	string
component-name	Name that links to a bridge component	string
pvid	VLAN ID with which one an incoming untagged frame should be tagged	uint32 (vlan-index-type)

Table 4.7: Standard parameters for a TSN bridge interface

Table 4.8 lists all the interface parameters to configure the TSN stream according to IEEE 802.1Qbv [11]. The *default-priority* parameter sets the PCP in the IEEE 802.1Q tag on incoming untagged Ethernet frames. This PCP value is used to select a queue, as discussed in Section 2.2. The mapping between PCP and a queue is done according to the *traffic-class-v2* values. The *admin-gate-states* parameter sets the initial value of the *Transmission Gates*. When the TSN bridge is running, the *Transmission Gates* change the states according to the gate control list, which can be changed with the *admin-control-list* parameter. The TSN device starts a new gate control list cycle in an interval defined by the *admin-cycle-time* value. In order that all TSN bridges start their gate control lists synchronized, the *admin-base-time* is used as basis to calculate the next start of the gate control list. The *admin-base-time* is also used to define a point in time to apply the *admin*-values to the running configuration. Additionally, a maximal Service Data Unit (SDU) can be defined for each traffic class. By extending the values in the *queue-max-sdu-table*, the frames are dropped.

4.4.2 OPC UA Publisher

A set of parameters is required to establish a connection between OPC UA Publisher and Subscriber and transfer data between them. This subsection shows the parameters required to configure a Publisher. The described parameters are only the ones required

Parameter	Description	Type
default-priority	PCP, with which one an incoming untagged frame should be tagged	uint8 (0...7)
traffic-class-v2	A list of traffic-classes to map it with PCPs	List of structs acc. to Table 4.9
queue-max-sdu-table	A list with maximal 8 entries to define the maximal SDU size, if a frame extends it, the frame is dropped	List of structs acc. to Table 4.10
gate-enabled	Defines if the traffic scheduling is active	boolean
admin-gate-states	Initial value of the gates, the most significant Bit controls the traffic class 7	uint8
admin-control-list	A list to set the gate control list	List of structs acc. to Table 4.11
admin-cycle-time	Time of one cycle that runs the gate control list, rational number	denominator and numerator [ns]
admin-base-time	Time when the admin-values should be applied, rational number	PTP timestamp (uint64[s] and uint32[ns])
config-change	Requests a configuration change. This parameter is set by the deployment tool	boolean

Table 4.8: TSN parameters for a TSN bridge interface

Parameter	Description	Type
traffic-class	Traffic class, for which the priorities should apply	uint8 (0...7)
priority	A list of priorities, which should be mapped to the according traffic class	uint8 (0...7)

Table 4.9: Struct Traffic-class-v2

Parameter	Description	Type
traffic-class	Traffic class where the limit applies	uint8 (0...7)
queue-max-sdu	Limit for the SDU	uint32

Table 4.10: Struct SDU size

to set up a PubSub connection to transfer SPDUs for OPC UA Safety. The OPC UA Safety standard defines many more parameters that are not required for the purpose described here. In order to establish a connection between the deployment tool and the OPC UA device, the Publisher's IP address is required. A username and password are also required if the authentication is enabled on the Publisher.

Parameter	Description	Type
time-interval-value	Time for how long this entry should apply	uint32 [ns]
gate-states-value	Defines the value of the gates, when the entry is executed,the most significate Bit controls the traffic class 7, where 0 indicates a closed gate	uint8

Table 4.11: Struct admin-control-list

OPC UA PublishedDataSet

As discussed in Section 2.1.1, a Publisher contains a Connection and a PublishedDataSet. The PublishedDataSet selects the data from the OPC UA address space, which are published. Table 4.12 shows the required parameters to configure the PublishedDataSet. The name of PublishedDataSet shown in the address space is configured with the *pubDataSetName* parameter. This parameter is also used by the OPC UA Connection object to link to the PublishedDataSet. The *fieldNameAliases* is part of the DataSetMetaData and provides information to decode the DataSetMessage if the Subscriber is configured manually. The *nodeIDs* select the data from the address space that should be published.

Parameter	Description	Type
pubDataSetName	Name of the PublishedDataSet object in the address space	string
fieldNameAliases	Array of Names for the published variables	array of strings
nodeIDs	The NodeIDs from the address space of the published variables. Length must be equal to the length of the FieldNameAliases	List of NodeIDs acc. to Table 4.13

Table 4.12: Parameters for OPC UA PublishedDataSet

Parameter	Description	Type
namespaceIndex	Namespace of the Node in the address space	uint16
identifier	ID of the Node in the address space	uint32

Table 4.13: Struct of a NodeID

OPC UA Connection

The second object that has to be configured on a Publisher is the Connection. It has a *name*, which is used in the address space. The *address* parameter defines where the Publisher should send the NetworkMessage. For the use case described in this work, this is always the IP address of the Subscriber and not a multicast address. As discussed in Section 2.1.1 and shown in Figure 2.3, a connection of a Publisher contains WriterGroups which contains Writers. The name of the Writer and WriterGroup is defined by the

writerGroupName and *writerName* parameters. For this work, a Publisher uses only one WriterGroup and one Writer because we want to transfer data only to one Subscriber with the same interval. The *publisherID*, *writerGroupID* and *dataSetWriterId* are used by the Subscriber to identify and filter received messages, which are sent in an interval defined by the *publishingInterval* parameter.

Parameter	Description	Type
connectionName	A name to identify the connection in the address space	string
address	URL of the Subscriber	string
publisherID	ID, to identify the Publisher. Depending on the *NetworkMessageContentMask*, it is included in the *NetworkMessages* for filtering.	uint64
writerGroupName	Each WriterGroup has a name, which is visible in the address space	string
writerGroupID	ID of the WriterGroup	uint16 (0...32,767)
publishingInterval	Interval for sending a *NetworkMessage*	float64 [ms]
writerName	Each Writer has a name, which is visible in the address space	string
dataSetWriterId	ID of the Writer	uint16 (0...32,767)

Table 4.14: Parameters for one OPC UA Publisher

4.4.3 OPC UA Subscriber

The OPC UA Subscriber needs parameters to filter out the information from the Publisher and parse them correctly. Like the OPC UA Publisher, the Subscriber contains a Connection responsible for communicating with the Publisher. The deployment tool also needs the IP address and, if required, username and password of the Subscriber under configuration.

OPC UA Connection

Table 4.15 shows the required Parameters. The *connectionName*, *readerGroupName* and *readerName* defines the name of the associated objects. The *address* defines on which interface the OPC UA server should listen for messages from the Publisher. The received messages are filtered and identified by the *publisherID*, *writerGroupID*, and *dataSetWriterId*, which must be equal to the parameters on the Publisher partner.

Parameter	Description	Type
connectionName	A name to identify the connection in the address space	string
address	IP of the Subscriber	string
readerGroupName	Name of the Object ReaderGroup in the address space	string
readerName	Name of the Object Reader in the address space	string
publisherID	ID to identify the Publisher. Needs to be the same as the *PublisherID* on the Publisher side	uint32
writerGroupID	ID of the selected *WriterGroup* from the Publisher	uint16
dataSetWriterId	ID of the DataSet selected from the Publisher	uint16

Table 4.15: Parameters for one OPC UA Subscriber

Metadata and TargetVariables

The variables received from the Publisher need to be put in the address space. Table 4.16 list all the parameters required to configure the Metadata and the TargetVariables.

The Name of the data set, in which the received data are represented in the address space, is defined with the *dataSetName* parameter. The array of *subDataName* defines the variable names of the received values. In order to decode the received data correctly, the datatype of all received variables needs to be defined with *subDataName*. The *targetNodeID* defines, which NodeID will be assigned the the variable in the address space. All parameters, except the *dataSetName* must have the same array length. The order of the parameters must be equivalent to the order with the array in the PublishedDataSet from Section 4.4.2.

Parameter	Description	Type
dataSetName	Name of the DataSet, in which the subscribed values are presented	string
subDataName	Names of the subscribed Data	array of strings
builtInType	IDs of the subscribed Datatype	int
targetNodeID	The NodeID, in which the subscribed data are represented	array of NodeIDs according to Table 4.13

Table 4.16: Parameters for one Metadata- and Target-variables of the OPC UA Subscriber

4.4.4 OPC UA Safety

To establish a safety connection between OPC UA SafetyProvider and SafetyConsumer, some parameters need to be set for each safety connection. Since there is no OPC UA Safety stack available, the deployment of an OPC UA Safety configuration could not be tested. The parameters at Table 4.17 are defined in the OPC UA Part 15 standard [8] as part of the Safety Parameter Interface (SPI).

Parameter	Description	Type
safetyProviderIDConfigured	Set the Provider-ID which is normally used	uint32
safetyBaseIDConfigured	A Globally Unique Identifier (GUID), which is a 128-bit value to identify the Provider	GUID
safetyStructureSignature	A signature to check the number, data types, and order of the transmitted safety data	uint32
safetyStructureSignatureVersion	Used version to calculate the *SafetyStructureSignature*	uint16
safetyStructureIdentifier	A string to describe the data type of the safety data	string
safetyProviderDelay	Time in μs, which indicates the maximum time that the Provider has from receiving the RequestSPDU until it starts to transmit the ResponseSPDU	uint32 [μs]
safetyConsumerIDConfigured	Set the SafetyConsumer-ID which is normally used	uint32
safetyProviderLevel	The SIL, that the SafetyConsumer expects from the SafetyProvider	byte (1 -4)
safetyConsumerTimeOut	Time for how long the SafetyConsumer waits to receive an error-free ResponseSPDU, before a timeout-error is triggered	uint32 [μs]
safetyOperatorAckNecessary	Sets if an acknowledgment from a user is required after an error occurred	boolean
safetyErrorIntervalLimit	Time Interval in which only one communication error is allowed to occurr	uint16 (6,60,600) [min]

Table 4.17: Parameters for OPC UA Safety

4.5 Centralized Safety Configuration Setup

The central aim of this work is to deploy a configuration on the three technologies, namely TSN, OPC UA PubSub, and OPC UA Safety. As discussed in Section 3, there are different approaches to configuring a TSN network. It can be done directly via NETCONF or with an additional SDN Layer, which mainly uses NETCONF for the SBI. Since the configuration is generated by an external service, which is out of scope for this work, there is no need for an SDN layer. Therefore, the configuration will be directly deployed via NETCONF.

OPC UA, on the other hand, already comes with an interface, which gives an OPC UA client the ability to change the OPC UA configuration via an RPC. Because of the requirement that the configuration must be deployed on three different layers, a central deployment tool will be used. This architecture has the advantage that the whole deployment can be controlled from one single point without the need for synchronization between different services. Figure 4.4 shows the architecture of a simple safety system with all data paths. As illustrated with the blue arrows, the deployment tool must deploy the configuration of those three layers on all safety devices and the TSN bridges. These three layers depend on each other, which must be considered for the configuration sequence. All three layers can be configured already in advance. The critical point in time is the activation of the configuration, respectively the deactivation in case of the *Remove safety device* use case. For an initial configuration, the TSN configuration must be activated at first because the two layers above depend on it. The used NETCONF brings the advantage of different datastores (cf. Figure 2.10), which gives the deployment tool the possibility to deploy the configuration upfront without any changes in the TSN network. In case of a wrong configuration, the deployment tool gets an error notification that does not require rollback actions. If all NETCONF servers on the TSN devices accept the configuration, it can be applied. As soon as the TSN configuration is used, Publisher and Subscriber can be activated (cf. Figure 2.4). The last layer, the OPC UA Safety depends on the two layers below. In case they are not working correctly, the black channel principle of the OPC UA Safety layer detects it, and the safety layer stops the manufacturing system. In case of a configuration error on the two OPC UA layers, all previously performed configurations need to be reversed.

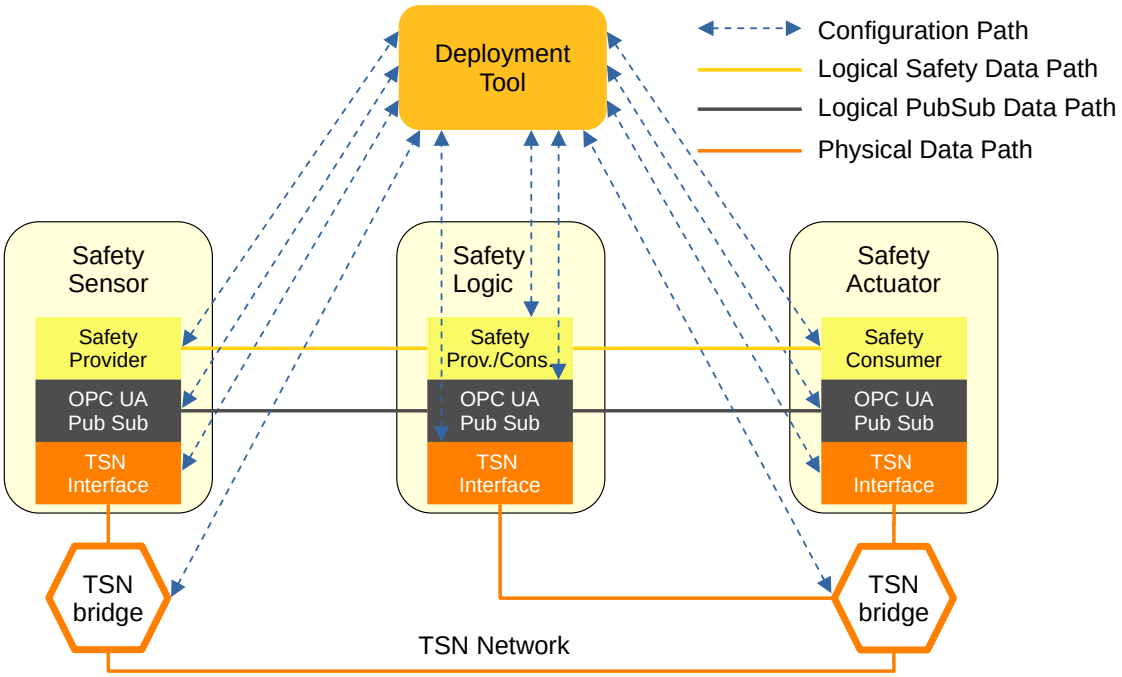


Figure 4.4: Central deployment architecture

CHAPTER 5

Deployment Tool

A big challenge of deploying a configuration on multiple devices with different technologies is the timing. For changing a configuration on TSN devices, the TSN standard provides the possibility to define a point of time where all devices change the configuration simultaneously. OPC UA, on the other hand, does not provide such a feature. Additionally, OPC UA Safety uses the black channel principle, which allows a connection interruption for a certain amount of time, but the time of a communication interruption during a configuration change is very limited. As shown in Figure 4.1, there are different use cases. In this chapter, the *Initial commissioning* use case is discussed.

5.1 Structure and Interfaces of the Deployment Tool

As discussed in Section 4.5, the deployment tool uses a central architecture that allows to control and schedule of the deployment process by one central tool. As shown in Figure 5.1, the deployment tool needs to modify the TSN, OPC UA PubSub, and the OPC UA safety layer. The NETCONF interface implements a set of YANG models, which is responsible for configuring the TSN bridges. The YANG models defined in IEEE 802.1Qcw exist in multiple versions, which differ significantly. This makes it necessary to set up the NETCONF interface according to the YANG models used in the TSN bridge. Additionally, it needs to communicate with a knowledge base to get the configuration data, and it should be platform-independent. A programming language that fulfills these criteria is Go [30], also known as Golang. Go code can be compiled for multiple hardware architectures and offers libraries for NETCONF, OPC UA method calls, and SPARQL for the knowledge base communication and is therefore used for the deployment tool.

5.1.1 NETCONF Interface

The deployment tool uses NETCONF to deploy the configuration on all TSN devices. The TSN deployment happens in two steps. At first, the configuration is deployed to the

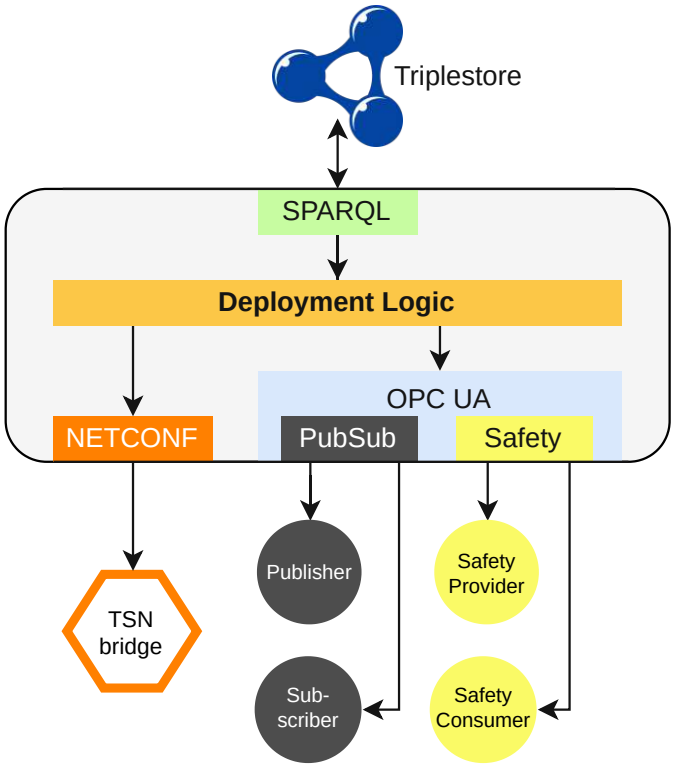


Figure 5.1: Structure of the deployment tool interfaces

candidate datastore (see Figure 2.10). In case of a wrong configuration, the deployment tool gets notified by the response code and can react to that error. The second step is the *commit*, which transfers the configuration to the *running* datastore and applies the configuration to the device. The configuration of a TSN interface has an additional feature to apply the configuration on all TSN devices simultaneously. For that, the deployment tool adds a time on which the TSN device should apply the configuration from the *running* datastore.

5.1.2 OPC UA Interface

OPC UA allows adding and removing nodes in the address space with methods that are also available in the address space. OPC UA PubSub already provides all necessary methods. OPC UA Safety, on the other hand, does not define such methods yet.

OPC UA PubSub

OPC UA PubSub provides methods in its address space, which can be used to add new Subscribers and Publishers. For the Publisher, it is first necessary to create a *PublishedDataSet*, where the information about the published data is stored. After that, the deployment tool can create a new connection, which links to the *PublishedDataSet* and

also contains the Subscriber IP address. On the Subscriber side, creating the connection and all necessary components with one method call is possible. Those method calls would return an error code if the configuration process was not successful.

OPC UA Safety

The current OPC UA Part 15, Release 1.05.00, [8] does not define any configuration methods yet, but it defines a SPI to modify the SafetyConsumers and SafetyProviders. The mechanisms to change the parameters are vendor-specific, though [8]. Additionally, this interface does not provide an enable flag. For this work, we assume that the SPI and a enable flag from the safety application can be modified via OPC UA.

With this assumption, the deployment tool can configure the SafetyConsumer and SafetyProvider and enable it as soon as it finishes the configuration of the layers below. In this way, the safety application can be configured in advance by the vendor tool without getting into a timeout when the TSN and the OPC UA PubSub devices are not configured. As soon as a connection is not needed anymore, it can be disabled without a vendor application.

5.1.3 Knowledge Base Interface

The deployment tool does not store any configuration by itself. This task is performed by the knowledge base. To recognize when a new configuration needs to be deployed, it must communicate with the knowledge base. For the sake of simplicity, the deployment tool polls the knowledge base in a fixed interval. In case of a new configuration, the deployment tool requests all necessary parameters and performs the deployment.

5.2 Use Case: Initial commissioning

To perform the initial commissioning, the involved devices must fulfill some preconditions to allow a connection with the deployment tool. If these preconditions are fulfilled, the TSN devices can be equipped with the basic setup and the safety connections between SafetyPublisher and SafetyConsumer can be established. In order to get a clear overview of the communication between the components, the Transmission Control Protocol (TCP), handshakes, and similar details are omitted in the sequence diagrams used in this chapter.

5.2.1 Preconditions

The basic network settings, like IP address and credentials, are required so that the deployment tool can establish a connection to those devices. Those settings are not the responsibility of the deployment tool and need to be set in advance.

Additionally, the three layers have some individual requirements:

- **TSN:** All TSN devices must run a NETCONF server to deploy the network configuration.
- **OPC UA PubSub:** The OPC UA server on the devices must provide methods to add and remove *Connections* and *PublishedDataItems*. These methods are optional according to the OPC UA Part 14 standard [8] but essential for the deployment process.
- **OPC UA Safety:** The OPC UA SafetyConsumer and SafetyProvider communicate via an SAPI to their safety application, which receives or sends all safety data. Since the safety application is vendor-specific, it must be configured in advance. In addition to the OPC UA Safety standard [8], we require an OPC UA interface to the safety application, to set an enable flag as, discussed in Section 5.1.2.

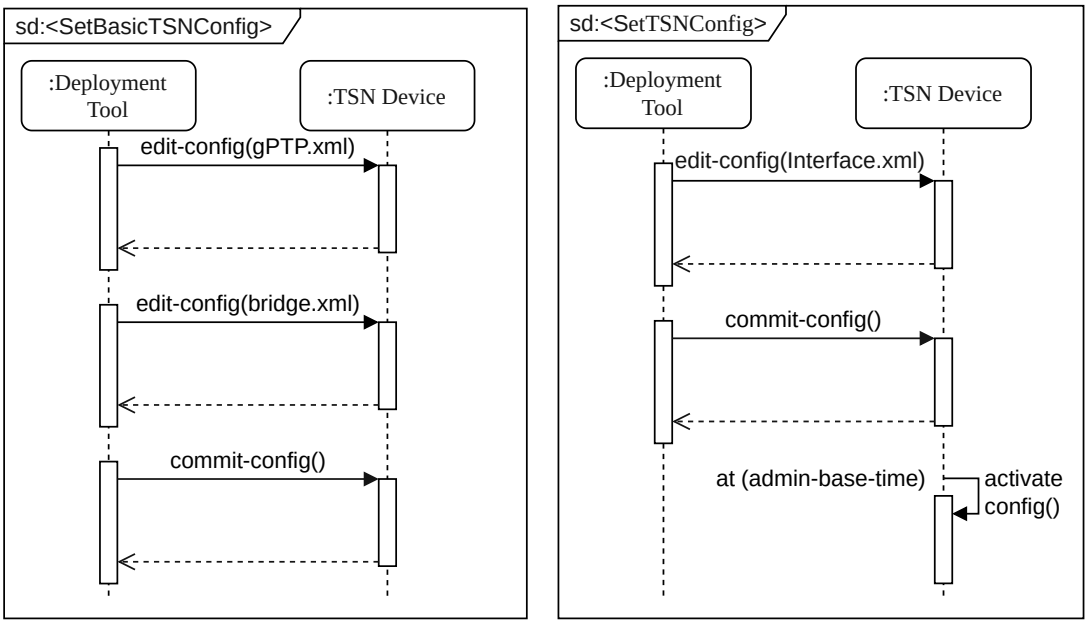
5.2.2 Basic Setup

Before it is possible to add a safety connection between SafetyConsumer and SafetyProvider, the deployment tool needs to deploy some basic configurations on the TSN devices. For the OPC UA PubSub, it is not necessary to make any configuration beforehand since a new connection between Publisher and Subscriber contains all necessary parameters. The OPC UA Safety layer also does not need any configuration in advance and can be configured completely when the safety connection is added. The configuration for the safety application, which uses OPC UA Safety, has no standardized interface for configuration and is therefore not configured by the deployment tool.

Only the TSN Layer needs a basic configuration before an TSN stream can be added. This needs to be only performed if new TSN devices are added, so the time between the TSN devices can be synchronized. Figure 5.2a shows the sequence of such a basic configuration. At first, the gPTP settings are deployed in order that the TSN network can select a grandmaster for the time synchronization. Additionally, all TSN bridges get their basic configuration, which is required to add a stream to the network. These basic settings are not time-critical since they are always performed at the commission of a new TSN bridge. Therefore, the settings are applied directly with a *commit-config()*.

5.2.3 Add Connections

Adding new safety connections to a safety system is a central task for the deployment tool since a safety system contains multiple safety connections. Figure 5.3 shows the communication between the deployment tool and the three layers for two communication partners for adding a new safety connection. The corresponding sub-sequence diagrams can be found in Figures 5.2, 5.4, 5.5 and 5.6. The sequence diagram shows the configuration of n Safety Connections and m TSN connections, but for a better overview, only two instances are graphically represented in each case. All devices are configured



(a) Basic TSN setup (b) Change TSN streams

Figure 5.2: TSN deployment

in multiple steps where each step is performed on all devices simultaneously, as the *par* blocks indicate. The first step, which sets up the basic TSN configuration, must be performed only if new TSN devices are added, as discussed in Section 5.2.2.

This clear separation of each step is only one option to perform the configuration, but this way makes it easy to evaluate the performance of the deployment and simplifies the rollback in case of an error. Nevertheless, there are four dependencies that need to be respected:

1. The basic setup must be configured as described in Section 5.2.2 before deploying the TSN config. This allows the configuration of the TSN interfaces to link to the basic configuration, as defined in the YANG models.
2. Publisher and Subscriber need the information which SPDU needs to be published and where to write them after the Subscriber receives them. Therefore, they need to be added after the SafetyProvider and SafetyConsumer. An overview of the relation between configuration components and the effected part of the messages can be found in Figure 2.3.
3. The Publishers and SafetyConsumers can only be enabled after they have been created.
4. To guarantee the latency of SPDUs sent over the PubSub connection, the TSN deployment has to be completed before the Publisher starts to publish.

After the basic TSN deployment, the deployment tool adds the OPC UA SafetyConsumer and SafetyProvider, followed by the Publishers and Subscribers on both sides. Since the OPC UA Safety standard is quite new, it does not define the possibility of a configuration via OPC UA yet. Therefore, no detailed sequence diagrams for OPC UA Safety are provided. Before any communication can be enabled a new stream in the TSN network must be added. After all TSN devices have successfully applied the new configuration, the Publishers can start to publish the data from the OPC UA Safety layer. Since the communication is established at this point, the OPC UA SafetyConsumer can be enabled, which completes the integration of the new safety connection. The OPC UA SafetyProvider and the OPC UA Subscriber are enabled with an enable parameter (cf. Section 4.4) when they are added to the OPC UA server. This is possible because they are causing only network traffic when they get contacted by their communication partner.

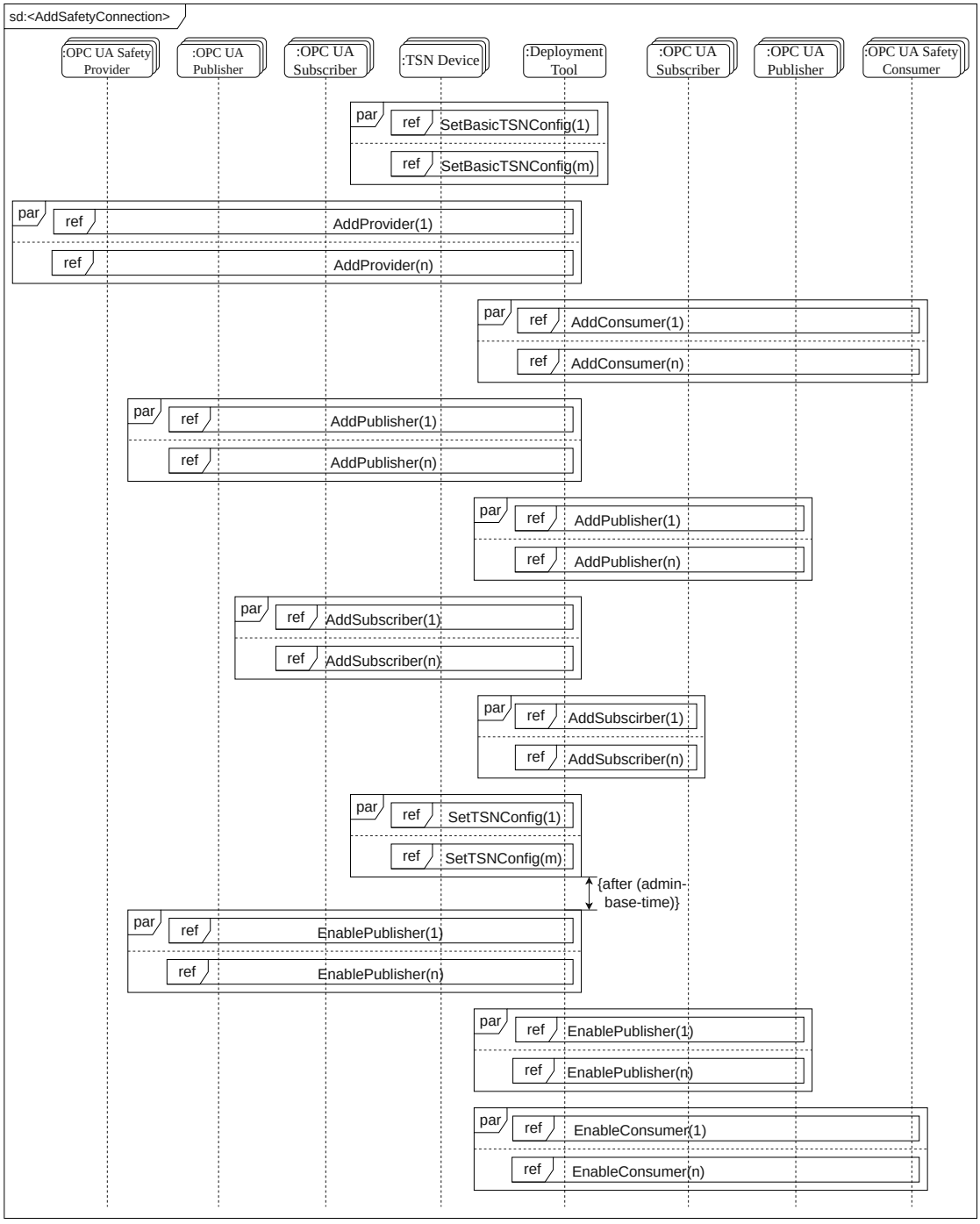


Figure 5.3: Sequence diagram add safety connections

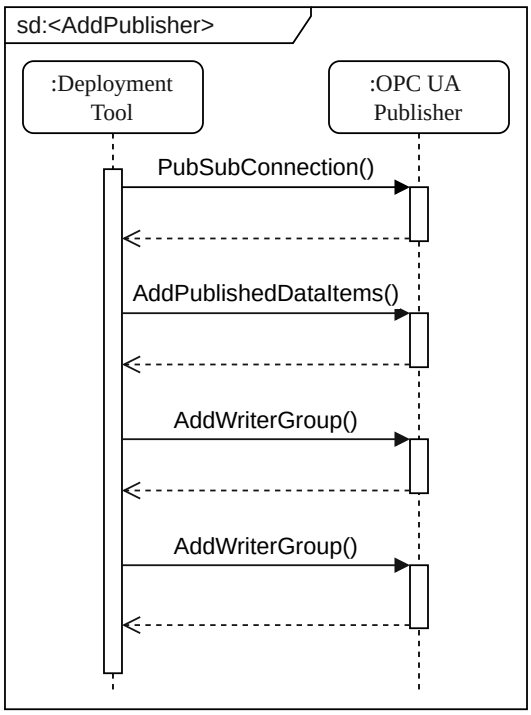


Figure 5.4: Add new Publisher

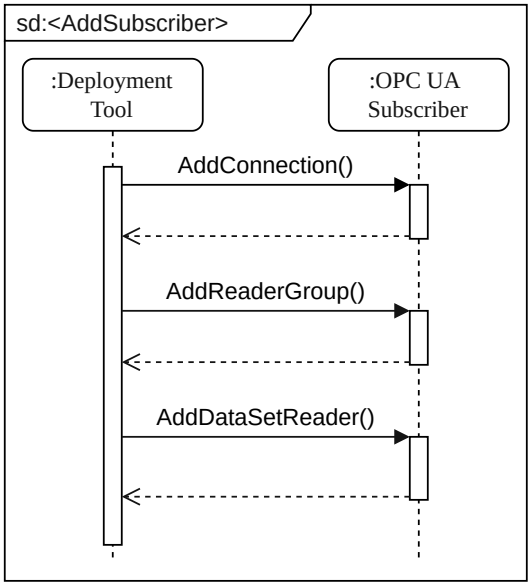


Figure 5.5: Add new Subscriber

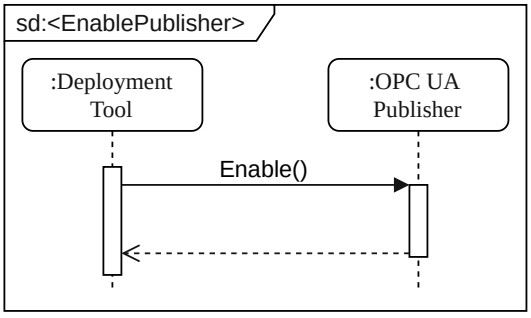


Figure 5.6: Enable new Publisher

5.3 Safety and Reliability Analysis

An RSS combines multiple technologies, which opens many possibilities for different errors. In order to identify all of them, a systematic method is needed as safety and reliability tools. Safety and reliability tools were first used in the air and space industry, but now, it is broadly used in many industrial fields [60]. Nowadays, many different analyses are available to identify fault states and their causes to improve the safety and reliability of a system.

5.3.1 Fault Tree Analysis

One of that analyses is the Fault Tree Analysis (FTA), which Bell Telephone Laboratories first developed to study the Minuteman Missile launch control system [58]. It was then used as a safety and reliability tool for complex dynamic systems such as nuclear reactors. The fundamental concept is the graphical representation of structured logic (fault tree). In order to construct a fault tree, mainly AND and OR elements are used to combine specified causes that lead to a top event [41].

5.3.2 Failure Mode and Effects Analysis

A further method is the Failure Mode and Effects Analysis (FMEA), which is used to analyze the potential failure model, the damage level, and the occurrence possibility of each kind of failure. For that, FMEA decomposes the system into components and their relationships. The functions of those components and relationships get analyzed, and then the failure table is formed with this information [60]. According to Bluvband and Grabov [17], FMEA is done in 6 steps:

1. Failure Models Identification
2. Ranking Procedure
3. Total Risk Estimate
4. Critical Items Identification
5. Corrective Action & Prevention Action
6. FMEA Effectiveness Evaluation

FTA and FMEA are both methods to improve the quality and reliability of the system. For complex systems, FMEA leads to a very long failure table, which is not well organized. FTA, on the other hand, splits the top events into sub-events and can be easily represented as a graph. A comparison between the Fault Tree Analysis and the FMEA can be found at [60].

5.3.3 Hazard and Operability Study

The last method presented here is the Hazard and Operability Study (HAZOP). It was developed by the Imperial Chemical Industries in the United Kingdom in the 1960s [31]. HAZOP uses guide words to identify the source of risks that may represent risks to personnel or equipment or prevent efficient operation [18]. Those guide words (e.g. *More, Less, or Reverse*) represent deviations of parameters to identify their consequences. The following steps represent a simplified procedure of a HAZOP [54]:

1. select a team leader and a multidisciplinary team with four to six members [18]
2. collect information on the plant and represent them in an appropriate way, such as Piping and Instrumentation Diagram (P&ID) or Engineering Line Diagram (ELD)
3. performs for each node in the plant following steps:
   a) explain the intention of the node
   b) assign to each parameter (e.g. pressure or flow) a guide word and identify the consequences of this deviation
   c) assess the consequences and define actions against these consequences if necessary

In contrast to the HAZOP, FMEA focuses less on guide words than on the cause factor [56]. For the deployment tool, we only need to identify the possible faults since any fault results in a rollback of the RSS and a notification to the operator. The FTA is the most straightforward analysis of the three introduced methods, which fulfills the requirement of identifying the fault states. In addition, it provides an ideal graphical representation. For that reason, the FTA is used in this work.

5.3.4 Fault Tree Analysis for the Deployment Tool

Figure 5.7 shows a Fault Tree Analysis for the RSS focusing on the deployment process. The top node indicates an error in the RSS. This can be split into two subgroups. An operational error and a configuration error. The operational error can happen at any time independently of the deployment. Therefore those errors are not monitored or handled by the deployment tool. The configuration errors, though, need to be detected and handled by the deployment tool.

Three different faults can cause a configuration error: an error response from an OPC UA or a NETCONF server, or if a device is not reachable. A not expected configuration causes an error reply from one of the two servers. If a device is not reachable, it can be caused by a device or network problem. A network error can be caused by a network overloaded with data traffic, a physical network problem, or a loss of messages in the network. A device error can be caused by either a NETCONF or an OPC UA server. The error handling of those errors is described in Section 5.2.3.

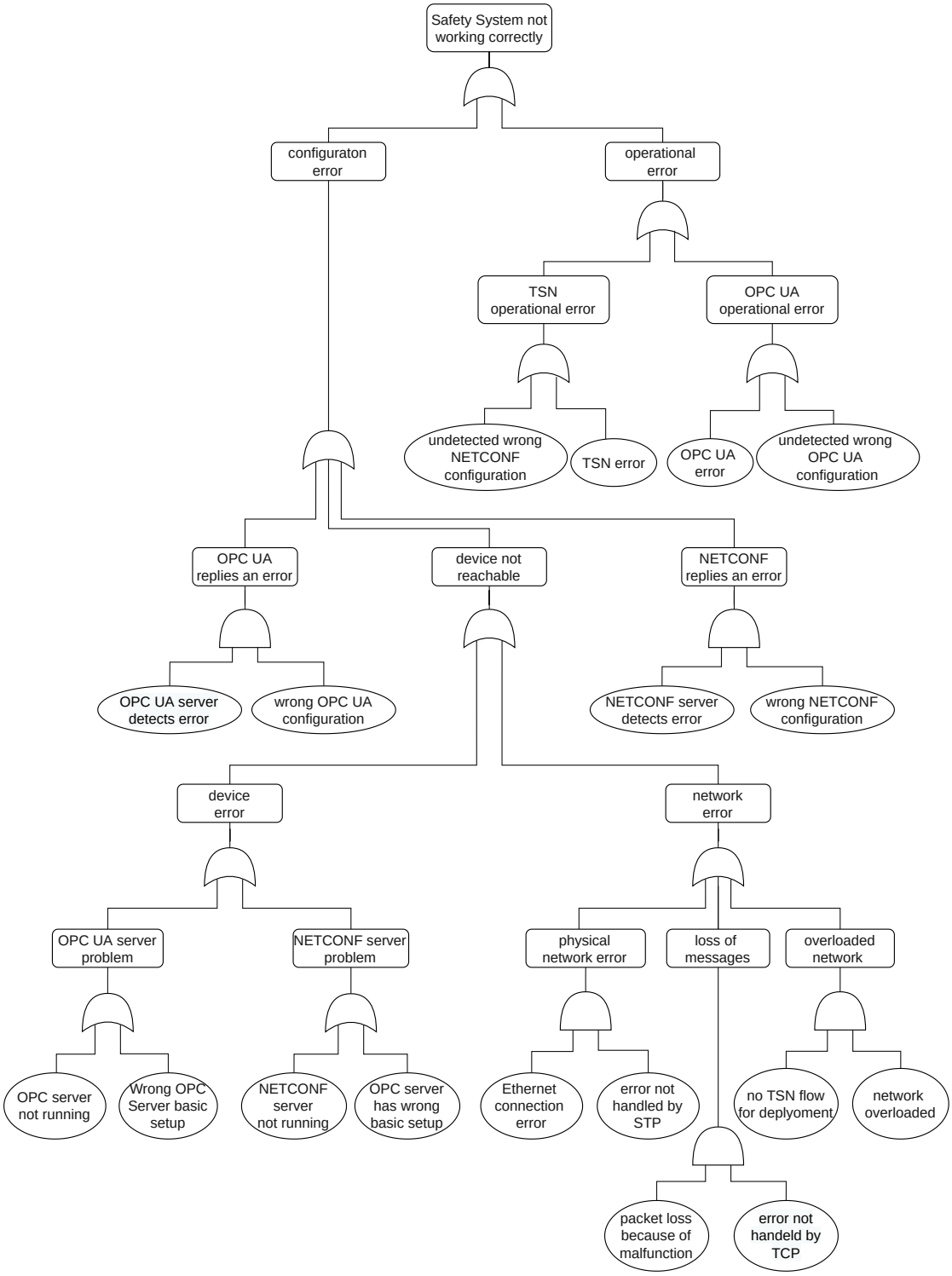


Figure 5.7: Fault Tree Analysis

5.4 Deployment Procedure with Error Handling

In Section 5.2, the communication between the deployment tool and the three layers is discussed. This section shows the procedures inside the deployment tool. Figure 5.8 shows an activity diagram for the initial commissioning of a safety system, including error handling. In order to keep the diagram clear and simple, the TCP connection handling is not represented in Figure 5.8. The deployment tool, though, establishes a TCP connection to all clients and closes it right before the deployment terminates. After the basic TSN configuration, as described in Section 5.2.2, a new connection is added. At first, all SafetyProviders are added, then all SafetyConsumers, and so on. After each step, the deployment tool checks if an error occurs on any node during the configuration. If an error occurs during the configuration deployment, all previously performed actions on the OPC UA server are reversed. It is unnecessary for the TSN bridges to reverse the configuration since the RSS can not be used after a failed deployment, and the old configuration is overwritten during the next deployment. In case of an error during such a reverse, it cannot be handled by the deployment tool because removing a once added entity can only be caused by external factors which are out of the scope of the deployment tool. As the last step, all Publishers are activated, followed by the SafetyConsumers. An error in those two steps also leads to a reversal of the previously performed actions.

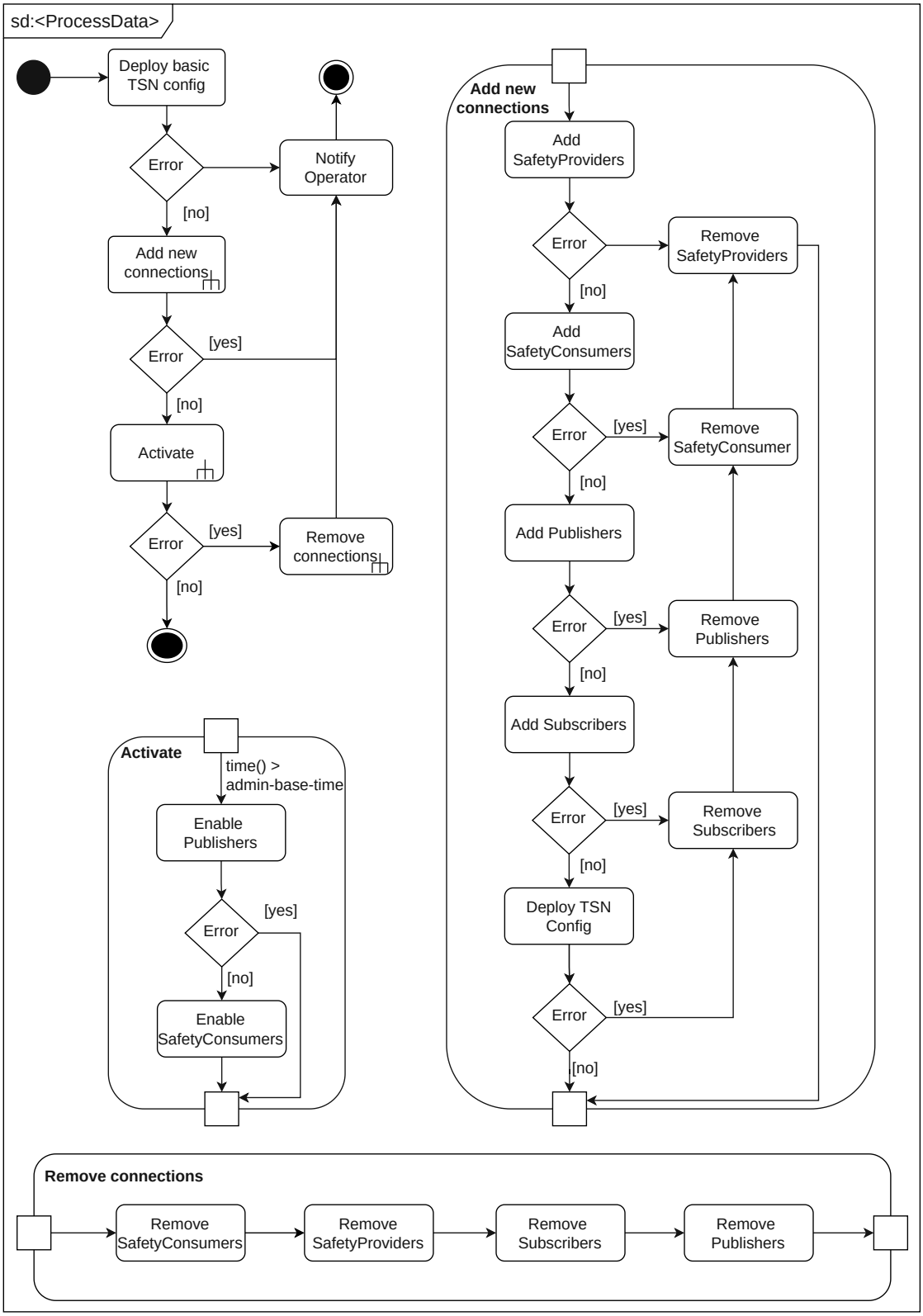


Figure 5.8: Activity diagram initial commissioning

CHAPTER 6

Evaluation

The evaluation of the deployment tool is performed with an emulated ns-3 network [1] and Linux Container (LXC) containers, where the TSN and the OPC UA servers are running. In order to compare the emulated network with a hardware network, the LXC containers run on distributed hosts, which are connected over a hardware network. The results are compared and discussed at the end of this chapter.

6.1 Used Software

Since there is just a limited amount of hardware yet available with which the deployed tool could be tested, we use software services to simulate the devices. For the TSN simulation, *Netopeer2* [22] is used as a NETCONF server and *open62541* [4] as an OPC UA server.

6.1.1 OPC UA

To run an OPC UA server, we use the *open62541* stack. *Open62541* is an open-source implementation of OPC UA, written in C. It includes an implementation of the OPC UA PubSub standard but not the OPC UA Safety. For the evaluation, we create a Publisher and a Subscriber, with enabled PubSub Information Model methods. These methods are used by the deployment tool to create the connections between the Publisher and Subscriber. The *open62541* does not include an Enable and Disable method to activate the Publishers, according to Figure 5.8. The activation, though, is essential to evaluate because a quick activation is a key feature for a configuration change, which can be done in future work. Therefore, we implemented these two methods in the open64541, which can enable and disable a Publisher, but without the additional functionalities described in Section 2.1.1.

6.1.2 TSN

The configuration for TSN devices is handled by a NETCONF server. For the evaluation, we use the *Netopeer2* server, which handles the NETCONF communication and uses *Sysrepo* [23] as a data store. *Sysrepo* also stores all the YANG models, which model the configuration of the network devices. For the evaluation, the following YANG models are used:

- ietf-interfaces@2018-02-20
- iana-if-type@2020-01-10
- ieee802-types@2020-10-23
- ieee802-dot1q-types2020-10-23
- ieee802-dot1q-bridge@2020-11-24
- ieee802-dot1q-sched@2020-07-07 with enabled *scheduled-traffic* feature
- ietf-gptp@2018-03-28 with slight modification that the dependencies are fulfilled

6.1.3 ns-3

ns-3 is an open-source discrete-event network simulation, mainly used for research and education [1]. A simulation can be either written in C++ or Python. ns-3 uses the following abstractions to create a simulated network [13]:

- **Node:** represents a network device like a computer or a mobile device
- **NetDevice:** is installed a *node* and represents Network Interface Card (NIC)
- **Channel:** represents a connection between *nodes* and is connected to the *NetDevices* of the *nodes*
- **Helpers:** arrange a connection between *nodes*, *NetDevices*, and *Channels*
- **Application:** represents a user application that generates some events in the simulated network

ns-3 can also include TAP devices where user applications can use ns-3 as an emulated network. These user applications generate the events in real-time and replace the ns-3 *Applications*. The ns-3 network behaves then for the user applications like a real network with the same timings [21].

6.1.4 LXC

LXC allows the creation of multiple isolated Linux virtual environments on a host machine. These virtual environments are called containers, which can be used to run applications. The difference between virtual machines and virtual environments is that virtual environments do not emulate hardware, only the operating system. This makes it much faster to start a container than a virtual machine [48]. LXC uses a virtual network interface to communicate with the host machine. This virtual network interface can then be connected to other physical and/or virtual network interfaces with the help of a Linux network bride. Docker is also software for creating virtual machines. A comparison between Docker and LXC is performed by Moravcik et al. [48].

6.2 Evaluation Environment with ns-3

To test the deployment tool and evaluate the performance, an environment is needed where the deployment can be tested with multiple OPC UA and NETCONF servers. However, the problem is that there is no device with OPC UA Safety available, and TSN switches with NETCONF are just in limited quantity available by the end of this project. Therefore, we decided to simulate the RSS and perform the evaluation on that simulation.

In order to make a meaningful evaluation, the deployment tool should be tested with multiple OPC UA and NETCONF clients. Figure 6.1 shows a network structure of an RSS, which is used for the first evaluation. The structure is split into multiple cells containing a maximum of four SafetyProviders or SafetyConsumers. Each cell has one bridge connected to a backbone switch, which is directly connected to the deployment tool. In order to simulate the NETCONF server from a TSN bridge, each bridge is connected to an LXC with a NETCONF server. In some use cases, it is necessary that the end stations use TSN and, therefore, also NETCONF. This scenario would require slightly modifying the evaluation environment, but this is out of the scope of this evaluation.

6.2.1 LXC setup

All the OPC UA and NETCONF servers are running in separate LXCs with Ubuntu 22.04. The *snap* package, which is by default activated in Ubuntu, generates traffic, which can influence the performance evaluation. For that reason, we deactivated the *snap* daemon. Because there is no OPC UA Safety stack available yet, all SafetyConsumer and SafetyProvider containers only contain an OPC UA server with a PubSub stack. The containers with a NETCONF server include all the models described in Section 6.1.2, but it does not apply the configuration on any hardware, which could also increase the deployment time. Those containers are connected over an ns-3 network that is running on the host machine. In order to simulate different network states, an additional delay and error model can be added to each Ethernet line, which is represented as orange lines in Figure 6.1.

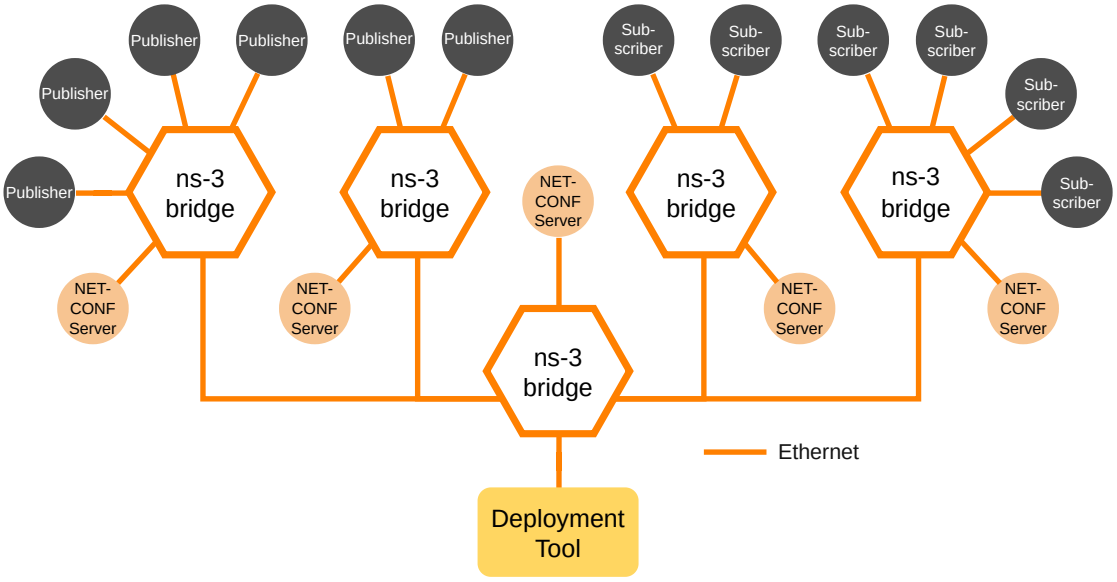


Figure 6.1: Structure of ns-3 evaluation environment

6.2.2 Connecting Components with ns-3

The ns-3 network simulator can either simulate an application to generate traffic or get real-time data from a running application. Because we want to test the whole combination between the deployment tool, network, and the device under configuration, we chose the approach with the real-time application data. As mentioned before, each instance of an OPC UA or NETCONF server runs in separate LXCs. Those containers must be connected via the emulated ns-3 network with the deployment tool. Figure 6.2 shows how those connections are realized. Each LXC has a virtual Ethernet interface, which can be accessed by the host of the LXCs. This *veth* interface is connected to a Linux bridge, which also has a connection to a separately created TAP interface. These TAP interfaces can then be connected to the ns-3 network simulator. The same can be done to connect the host with the ns-3 network. After deleting the network routes on the host, the traffic between the LXCs and the deployment tool is routed over the simulated network.

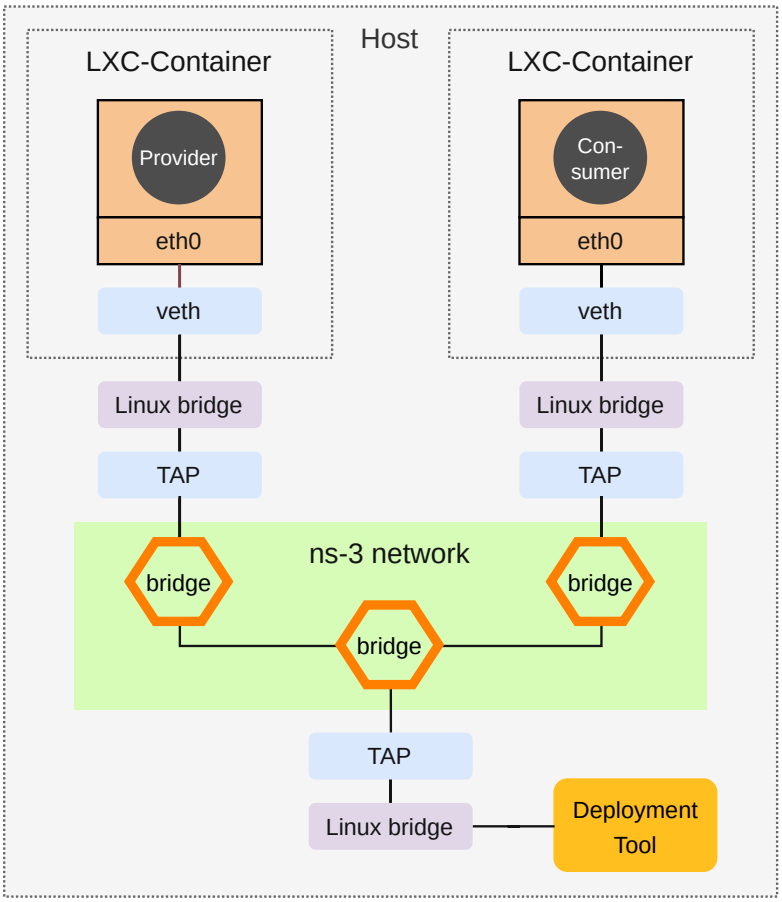


Figure 6.2: Connection between LXC and ns-3

6.3 Results

The evaluation is split into two parts. At first, a qualitative evaluation is performed, where the correct functionality of the deployment tool is tested. The second part evaluates the performance of the deployment tool in different configurations.

6.3.1 Qualitative Evaluation

The qualitative evaluation checks the correct response on all the faults which were found with the fault tree analysis shown in Figure 5.7. The deployment tool must behave on those faults as defined in the activity diagram in Figure 5.8. Table 6.1 lists all tested faults, how they are simulated, and how the deployment tool responds to them.

error	way of simulating the error	expected behavior	result
OPC UA server problem	disable the OPC UA server	reply an error and stop the deployment	✓
NETCONF server problem	disable the NETCONF server	reply an error and stop the deployment	✓
physical network error	remove connection to a single LXCs	reply an error and try to rollback all modified OPC UA devices to the initial condition	✓
loss of messages	use an error model in ns-3, that the TCP can not handle	reply an error and try to rollback all modified OPC UA devices to the initial condition	✓
overloaded network	increase the latency of the Ethernet links in the ns-3 network, which cause a timeout in the deployment tool	reply an error and try to rollback all modified OPC UA devices to the initial condition	✓
OPC UA replies an error	deploy a wrong OPC UA configuration	reply an error and try to rollback all modified OPC UA devices to the initial condition	✓
NETCONF replies an error	deploy a wrong NETCONF configuration	reply an error and try to rollback all modified OPC UA devices to the initial condition	✓

Table 6.1: Qualitative tests for the deployment tool

6.3.2 Quantitative Evaluation with ns-3 Network

The second part of the evaluation is the quantitative evaluation. In this section, the performance of the deployment tool is tested. Figure 6.1 gives an overview of the used evaluation structure. Since the deployment tool deploys the configuration on OPC UA Publishers, OPC UA Subscribers, and TSN bridges at the same time, we define the unit communicationSet. This unit is used in the evaluation to define the number of devices under configuration. One communicationSet contains one Subscriber, one Publisher, and multiple TSN bridges. Because the relevant part for the configuration of the TSN bridges is the NETCONF server, only the NETCONF server was used for the evaluation. All involved devices, namely Subscriber, Publisher, and NETCONF servers are running on separate LXCs. Figure 6.1 shows the evaluation environment with six communicationSets. A precise relation between a communicationSets and the individual instances can be found in Formulas 6.1 to 6.4.

$$\#ProviderNodes = \#ConsumerNodes = \#communicationSets \tag{6.1}$$

$$\#Publishers = \#Subscribers = \#communicationSets * 2 \tag{6.2}$$

$$\#NETCONF\_servers = \frac{\#communicationSets}{2} + 1 \tag{6.3}$$

$$\#LXC\_Containers = \frac{5}{2} * \#communicationSets * +1 \tag{6.4}$$

Duration of Deployment

Figure 6.3 shows Box-Plots of the deployment duration with a different number of communicationSets. The duration measurement starts when the deployment process starts until all configuration modifications get a positive reply from devices under configuration. This includes the TCP and SSH handshake, but not the connection termination.

The Box-Plots in Figure 6.3 shows a mainly linear increase of the deployment duration until 44 communicationSets. Starting with 48 communicationSets, the variation increases, and the number of outliers also increases with the number of communicationSets. As it is clearly visible in the Box-Plot with 72 communicationSets, the deployment duration increases significantly and is not linear anymore. The reason for that is the architecture of the evaluation environment shown in Figure 6.2. All LXCs, the ns-3 network, and the deployment tool are running on one single host. On a deployment event, all NETCONF and OPC UA servers need to process the new configuration simultaneously. The ns3-network needs to transfer all the configurations at the same time as well. For that reason, the number of communicationSets is limited by the performance of the host machine. To limit the influence of the host machine on the evaluation results, we limit for further evaluations the number of communicationSets to 40.

Figure 6.4 shows the same data as Figure 6.3, but the number of nodes is limited to 40. This allows seeing the statistical information of the deployment process with fewer communicationSets in more detail. By looking at the median values, it is possible to see that the increase in the deployment duration is mainly linear. If we compare the time between 20 and 40 NodeSets, the time doubles from 612 ms to 1285 ms. The comparison between 12 and 24 nodes (431 ms and 693 ms) shows that duration increases even less. This is caused by the fact that the deployment tool needs to establish an SSH connection to all NETCONF servers, which takes a certain amount of time, even with one node. The SSH connection establishment takes, therefore, a bigger part of the whole duration. This will be discussed in more detail in Section 6.3.2.

Enable OPC UA Publisher

Enabling an OPC UA Publisher can be a time-critical task for future works, where the configuration needs to be changed without interrupting the RSS. Therefore, Figure 6.5 shows Box-Plots of the Publisher enable duration. For that measurement, a TCP

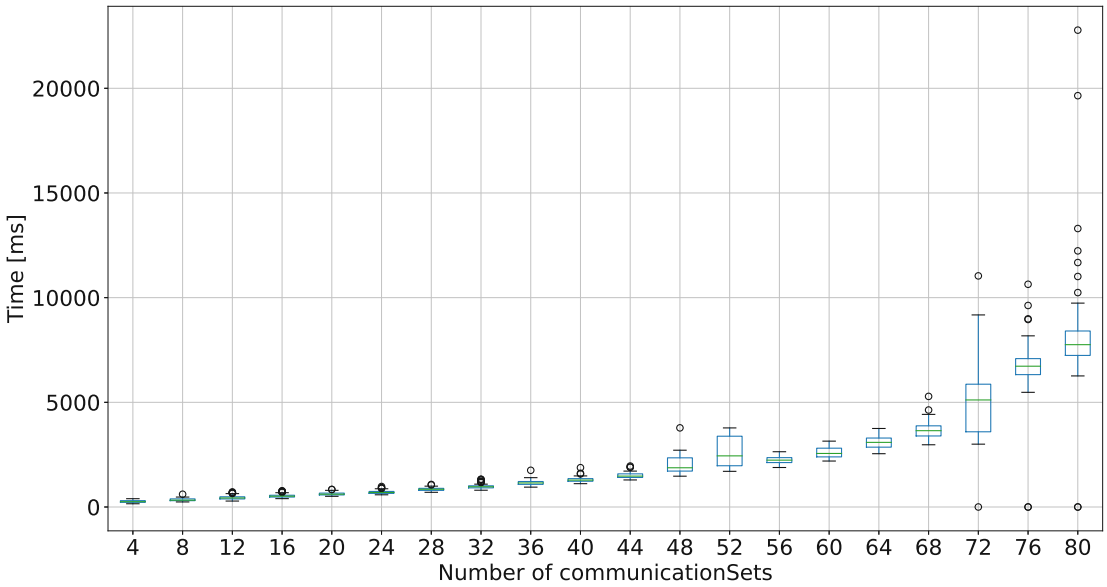


Figure 6.3: Duration of deployment for ns-3 network

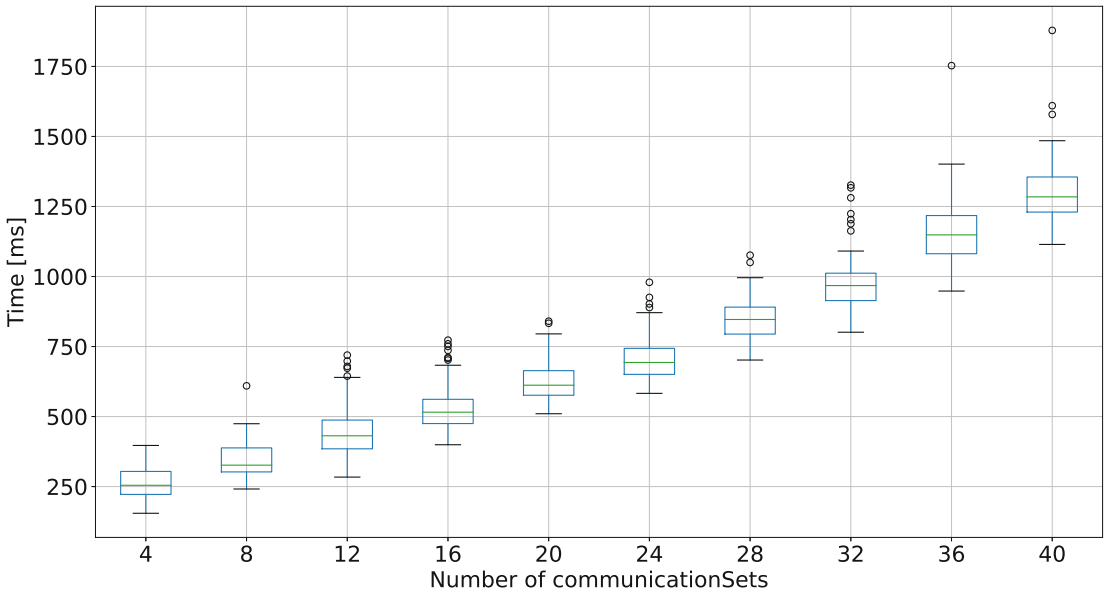


Figure 6.4: Duration of deployment until 40 communicationSets for ns-3 network

connection is established in advance. The measurement starts when the OPC UA method *Enable()* is called and stopped when the last positive reply arrived from the OPC UA Server. As we can see in the Box-Plots, the duration increases rapidly, starting with 32 communicationSets. It can be assumed that this steep rise is based on the chosen

evaluation environment. In order to prove this assumption, a comparison with the distributed hardware environment will be performed in Section 6.3.3.

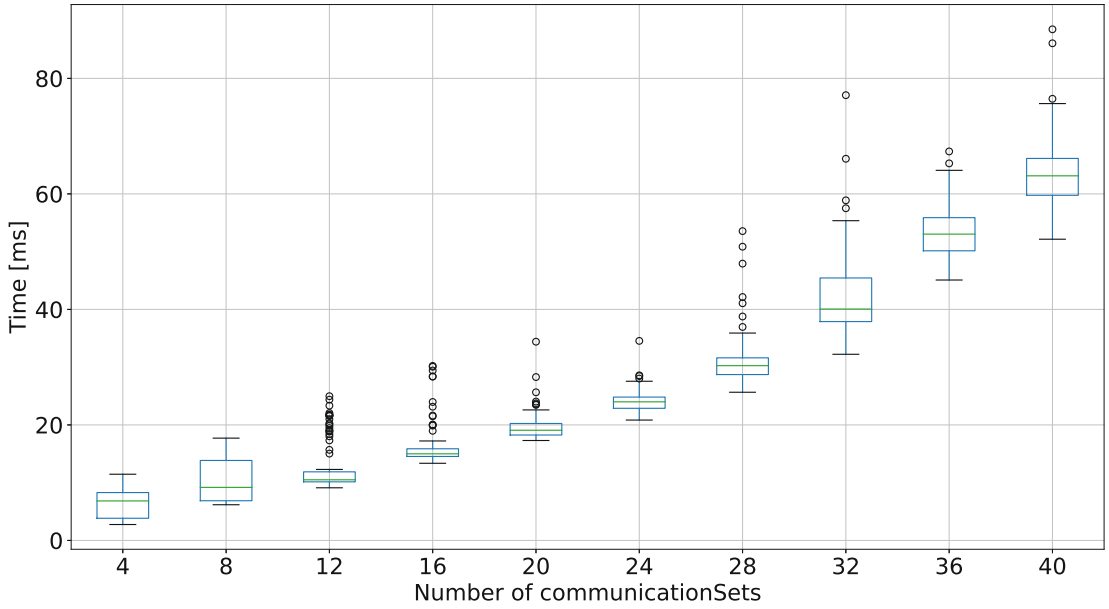


Figure 6.5: Duration of enabling the OPC UA Publishers for ns-3 network

Composition of Deployment Duration

As discussed in Section 5.4, the initial commissioning deployment process consists of multiple steps. Figure 6.6 shows the percentage of the total time required for the individual steps. To keep the pie chart clear, the *deploy basic TSN config* and *Deploy TSN Config* were combined into one slice. The left pie chart in Figure 6.6 shows the deployment with 20 communicationSets and the right with 40 communicationSets. The orange part is the duration for TSN, and the gray part for OPC UA. Most of the time is used to establish the connection to the servers. Since NETCONF requires an SSH connection, it takes longer than the TCP connection establishment to the OPC UA servers. It stands out that the connection to NETCONF servers at the 20 communicationSets evaluation uses with 34.1% a more prominent part than the 40 communicationSets evaluation. To explain that, we look at Table 6.2, which compares the single median durations of the deployment steps. The NETCONF connection duration increases from 206 ms to 277 ms, relatively little compared to the other values. This can be explained with the SSH handshake, which requires an algorithm negotiation and a key exchange. Since this requires some time where the deployment tool waits for a response, the parallelization can improve performance better than the other steps.

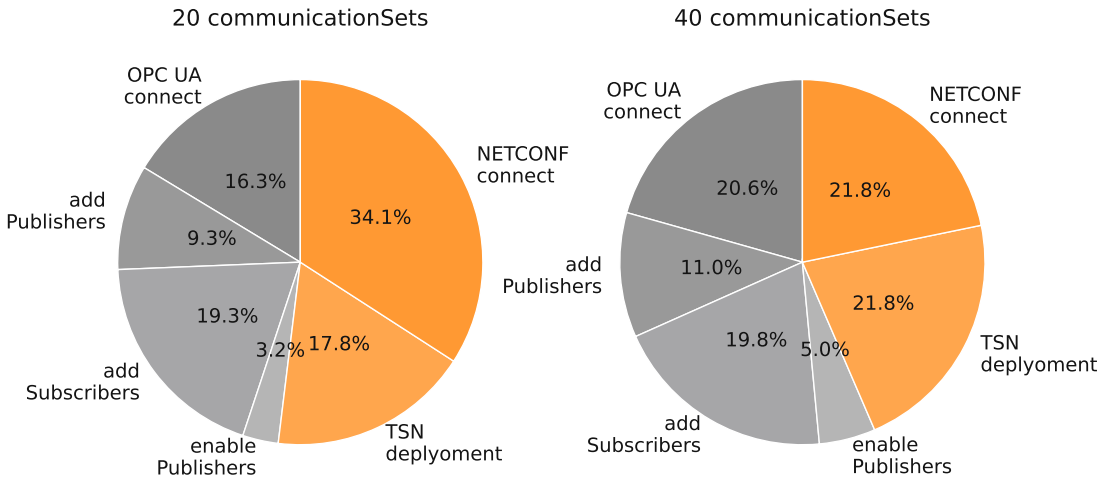


Figure 6.6: Distribution of duration

communicationSets	20	40
OPC UA connect	98.78 ms	261.04 ms
add Publishers	56.27 ms	139.99 ms
add Subscribers	116.33 ms	251.33 ms
enable Publishers	19.07 ms	63.12 ms
NETCONF connect	206.08 ms	276.80 ms
TSN deployment	109.16 ms	275.45 ms

Table 6.2: Qualitative tests for the deployment tool

Distribution of Measurements

Each deployment process variates on the deployment time, even if the same data are transmitted. This is caused by the involved host machines and the network, which run under different loads over time. In order to achieve a comprehensive result, 100 measurements were performed for each number of communicationSets. Figures 6.7 and 6.8 show the distribution of the deployment duration of 20 and 40 communicationSets. The deployment duration starts with the TCP connection and ends with a positive response from all devices under configuration. The dashed line represents the median value from the measurements. If we compare the two histograms, we can see that the duration increases with a higher number of communicationSets.

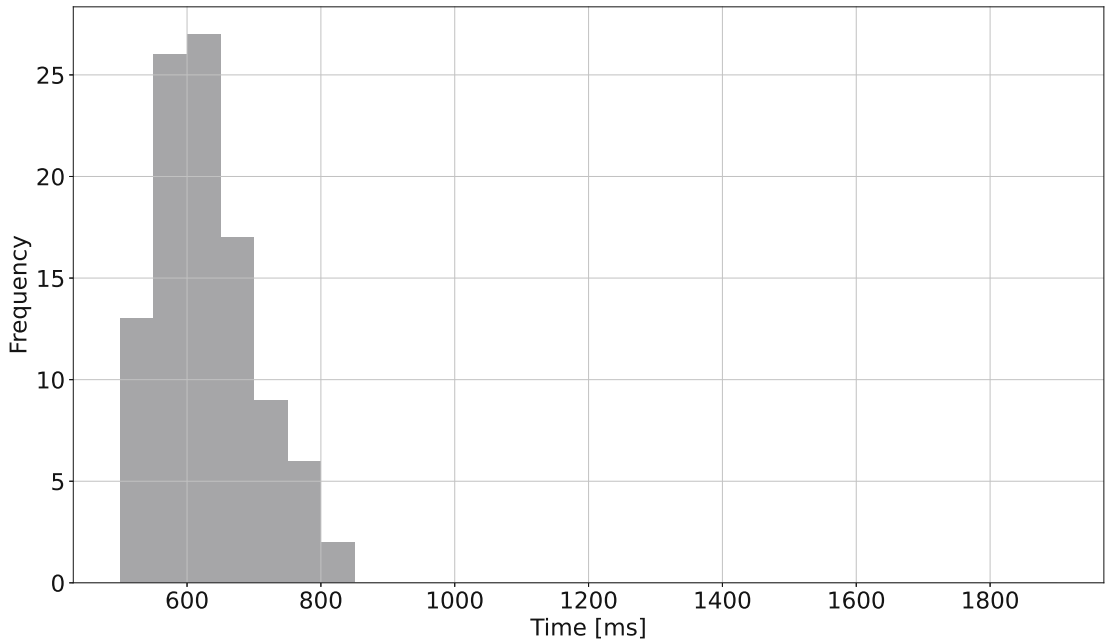


Figure 6.7: Histogram of deployment duration with 20 communicationSets for ns-3 network

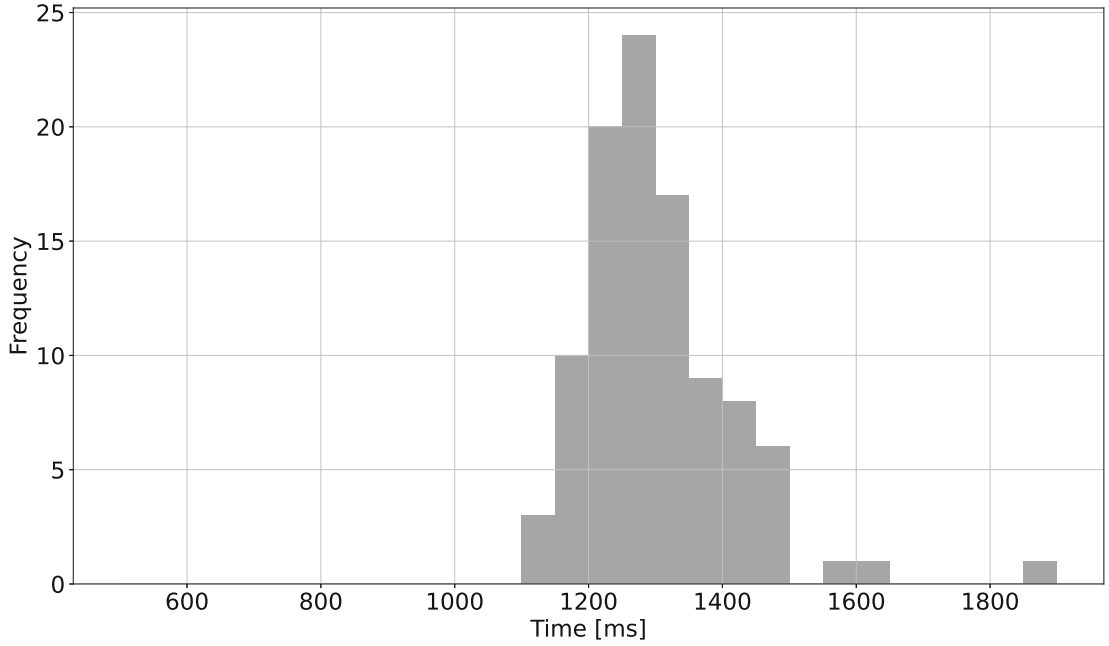


Figure 6.8: Histogram of deployment duration with 40 communicationSets for ns-3 network

6.3.3 Comparison between ns-3 and Hardware Network

As discussed in the previous section, the number of communicationSets is limited because of the host's performance limitation. In order to reduce this limitation, the LXCs were distributed on four different hosts, and the ns-3 bridges were replaced with Linux bridges, as shown in Figure 6.9. Each node runs multiple containers with OPC UA and NETCONF servers. A Linux bridge connects a maximum of four containers with an OPC UA server and exactly one container with a NETCONF server. Each host uses one Linux bridge as a backbone to connect all bridges.

The LXCs were distributed in a way that the load is balanced equally, even if only four communicationSets were used. The connection between the hosts is established with a *MikroTik CRS112-8P-4S-IN* switch and an additional Ethernet interface on each host. The network speed on the deployment tool is reduced to 10 Mbit/s to keep the evaluation with the hardware switch comparable with the ns-3 network. To evaluate the influence of the network speed on the deployment duration, a second test with 100 Mbit/s was performed.

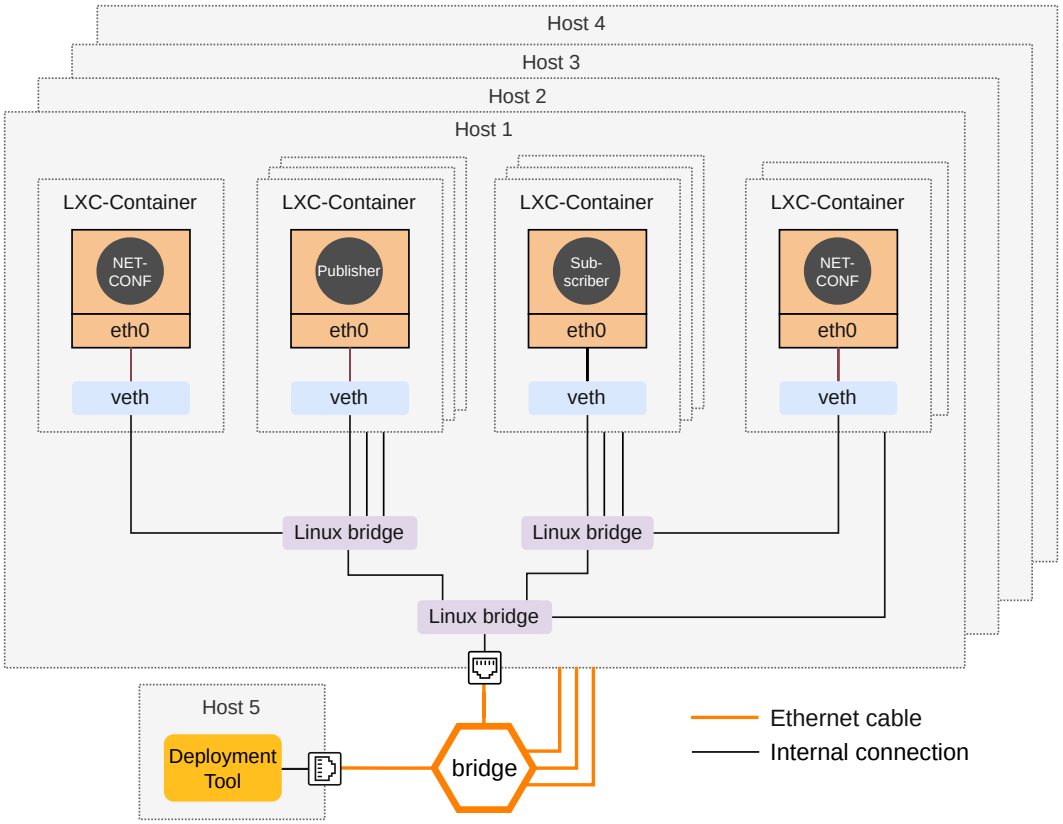


Figure 6.9: Structure of hardware evaluation environment

Figure 6.10 compares the median deployment duration of the ns-3 and the hardware evaluation environment. The blue line shows that above 60 communicationSets, the

deployment duration increase exorbitantly on the ns-3 network. The comparison between the orange and the green line shows that also the network speed has a significant impact on the deployment duration. In order to see the deviation and the outlines of the duration with a hardware network, Box-Plots are used in the following section, as before with the ns-3 network.

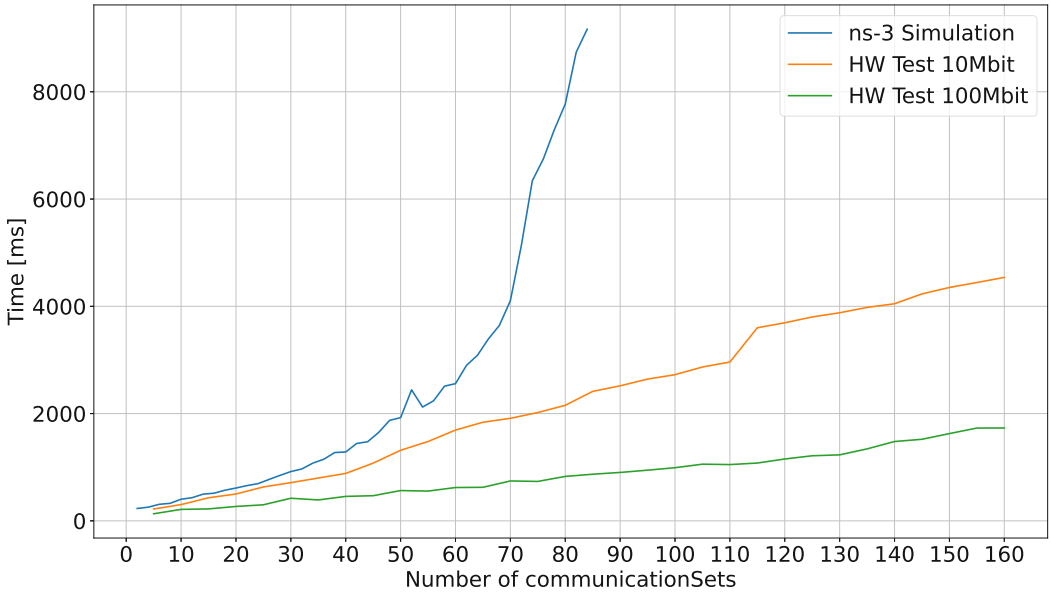


Figure 6.10: Compare ns-3 with hardware evaluation environment

Compare Duration of Deplyoment

Figures 6.11 and 6.12 show the Box-Plots for the duration of the whole deployment process, as already defined in Section 6.3.2. A comparison with the ns-3 network in Figure 6.3 shows that the outliers start later with more NodeSets. This proves the assumption that the load on a single host machine influences the duration of the deployment.

An interesting observation can be made by comparing Figures 6.11 and 6.12. The outliers start at the 100 Mbit/s network already with 80 NodeSets and at the 10 Mbit/s network just at 110 communicationSets, even though the median duration is much shorter. A closer evaluation of the raw data shows that these outliers are caused by the TCP and SSH handshake at the connection establishment. The actual cause was not investigated, but a possible reason is that the flood of connection requests is higher if they are sent from a 100 Mbit/s interface. This causes a high CPU usage on the hosts, which can then cause a packet loss or a delayed reply.

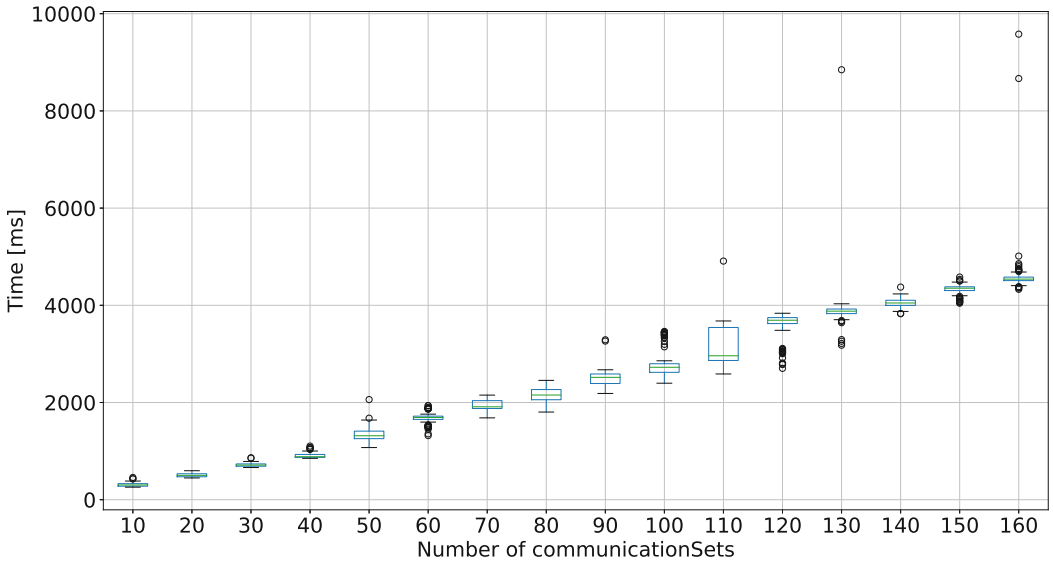


Figure 6.11: Deployment duration for 10 Mbit/s network

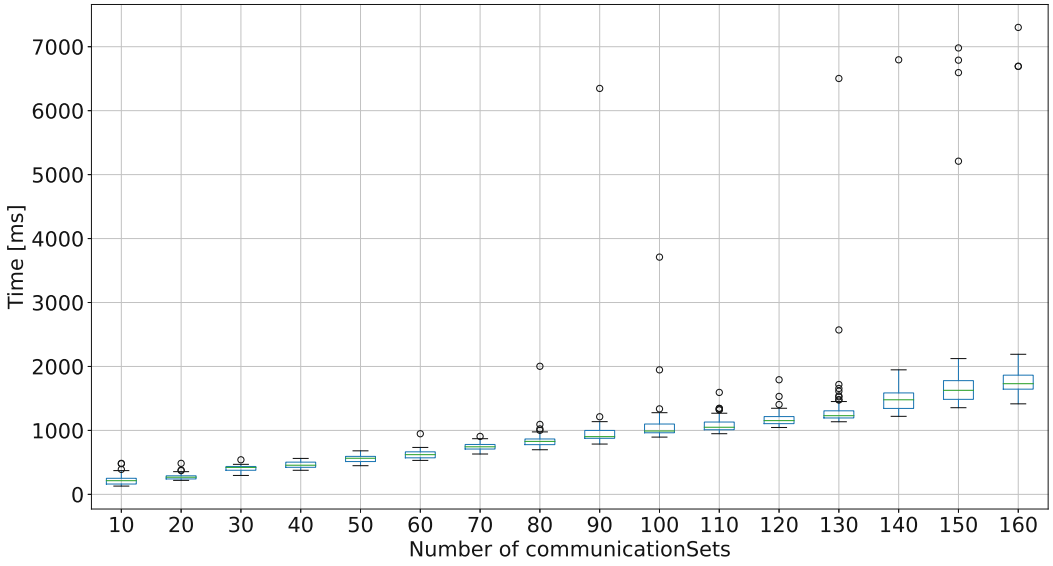


Figure 6.12: Deployment duration for 100 Mbit/s network

Enable OPC UA Publisher

As described in the evaluation of the ns-3 network in Section 6.3.2, enabling a Publisher can be a time-critical task for future work. In Section 6.3.2, we assume that the rapid increase of the *Enable Publisher* duration with higher communicationSets is caused by the structure of the evaluation environment. Therefore, that analysis is repeated with a hardware network. In Figures 6.13 and 6.14, it is clearly visible that the network

bandwidth has a significant influence on the *Enable Publisher* duration. On the 10 Mbit/s network, outliers show that the duration of enabling Publishers for 140 communicationSets can take above 240 ms. On the other hand, the 100 Mbit/s network shows outliers already at 50 communicationSets. The duration, though, is always below 25 ms. The impact of the network bandwidth of the *Enable Publisher* duration compared with the other steps is shown in the following section.

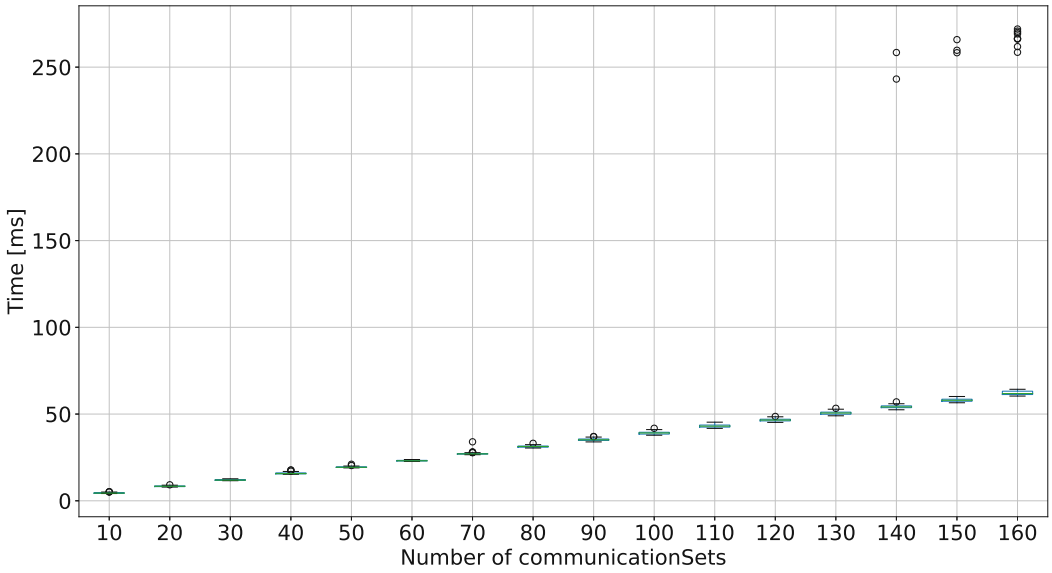


Figure 6.13: Duration of enabling the OPC UA Publishers for 10 Mbit/s network

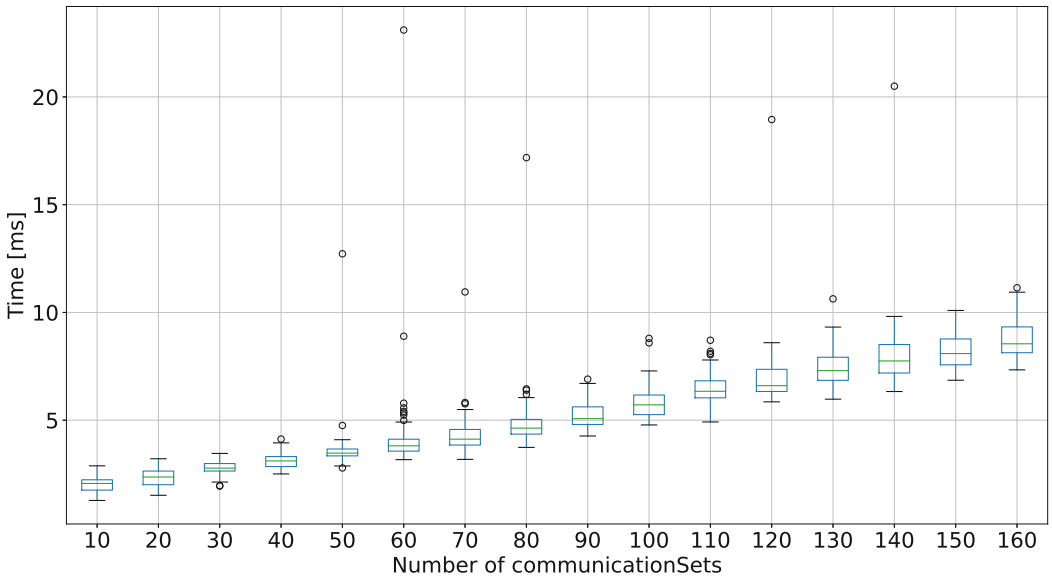


Figure 6.14: Duration of enabling the OPC UA Publishers for 100bit/s network

Composition of Deployment Duration

Figures 6.15 and 6.16 show the percentage of each step of the deployment duration for a 10 and a 100 Mbit/s network, where the TSN part is orange and the OPC UA part gray. The trend that a higher network bandwidth reduces the *Enable Publisher* duration can also be seen here. For 40 communicationSets with the 10 Mbit/s network, the *Enable Publisher* duration takes 1.8% of the whole deployment time, while on the 100 Mbit/s it reduces to 0.7%. This shows that the network bandwidth has a relatively high impact on the *Enable Publisher* duration.

The TSN connection establishment and adding an OPC UA Subscriber and a Publisher takes relatively longer with higher bandwidth. This is because these steps takes more time on the hosts than on the network. For example, enabling a Publisher only changes the Publisher's state, while adding a Subscriber requires multiple steps on the OPC UA server, such as adding a Connection, ReaderGroup, and DataSetReader and adding the subscribed variables afterward in the address space. The absolute time is always decreasing with higher bandwidth, as seen in Table 6.3.

NodeSets	20			40		
Network	ns-3	10 Mbit/s	100 Mbit/s	ns-3	10 Mbit/s	100 Mbit/s
OPC UA connect	98.78	57.32	8.82	261.04	112.12	15.24
add Publishers	56.27	67.10	39.45	139.99	131.39	75.71
add Subscribers	116.33	131.17	99.30	251.33	257.91	194.68
enable Publishers	19.07	8.39	2.36	63.12	15.68	3.11
NETCONF connect	206.08	108.40	28.66	276.80	194.86	38.91
TSN deployment	109.16	130.35	86.30	275.45	166.09	130.12

Table 6.3: Duration in ms of single steps for 20 communicationSets

NodeSets	80		160	
Network	10 Mbit/s	100 Mbit/s	10 Mbit/s	100 Mbit/s
OPC UA connect	321.31	26.60	1068.36	56.01
add Publishers	442.84	148.18	698.07	300.84
add Subscribers	513.33	387.23	1036.15	783.78
enable Publishers	31.30	4.63	61.81	8.54
NETCONF connect	612.16	61.10	1106.32	106.54
TSN deployment	284.86	190.96	554.24	456.60

Table 6.4: Duration in ms of single steps for 80 and 160 communicationSets

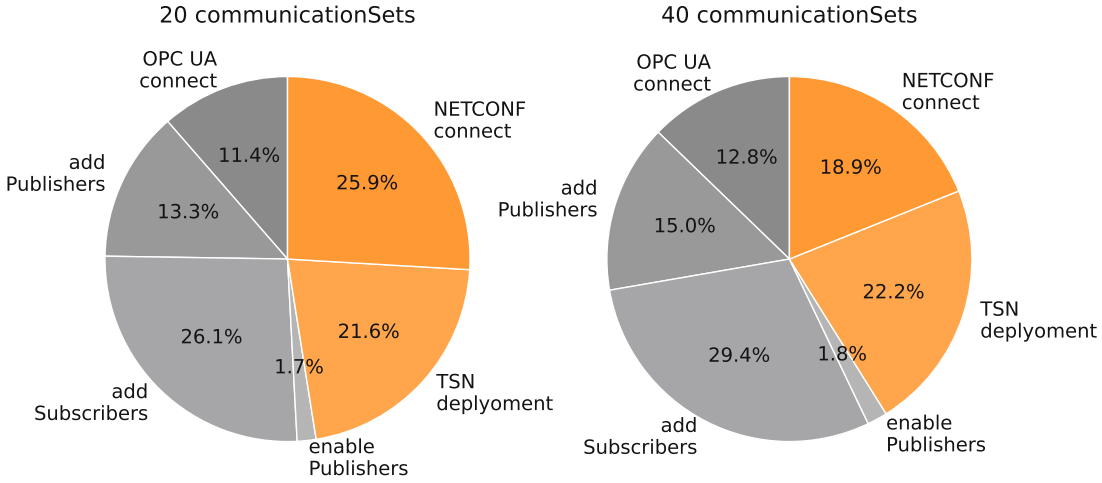


Figure 6.15: Distribution of duration for 10 Mbit/s network

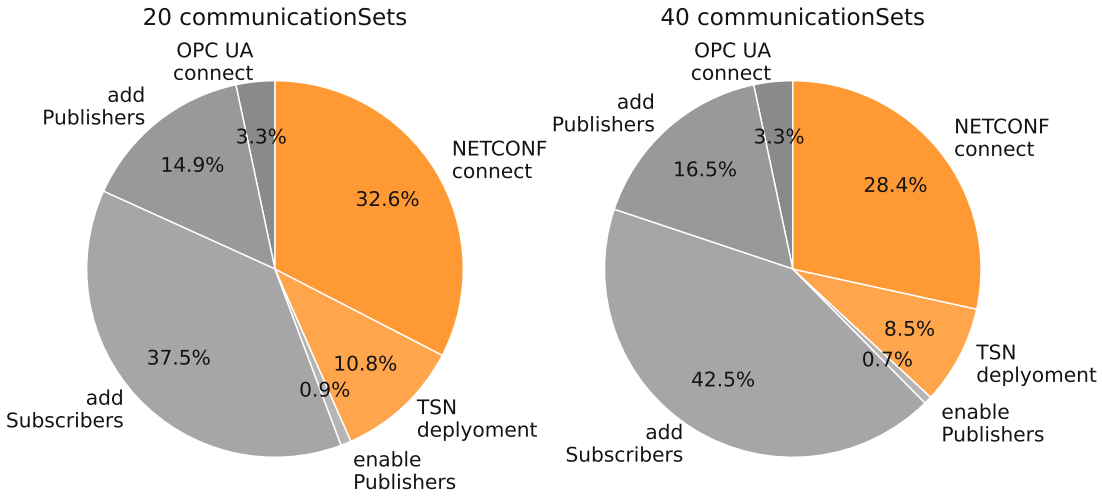


Figure 6.16: Distribution of duration for 100 Mbit/s network

Distribution of Measurements

For the distribution evaluation of the deployment duration process, histograms were used. The first two histograms in Figures 6.17 and 6.18 show the distribution, as it is defined in Section 6.3.2, at the ns-3 duration measurements. It is possible to see that the distributions increase with a rising number of communicationSets, as it was already the case with the ns-3 network. The histogram in Figures 6.19 and 6.20 show a rising distribution for the 100 Mbit/s network with a rising number of communicationSets as well.

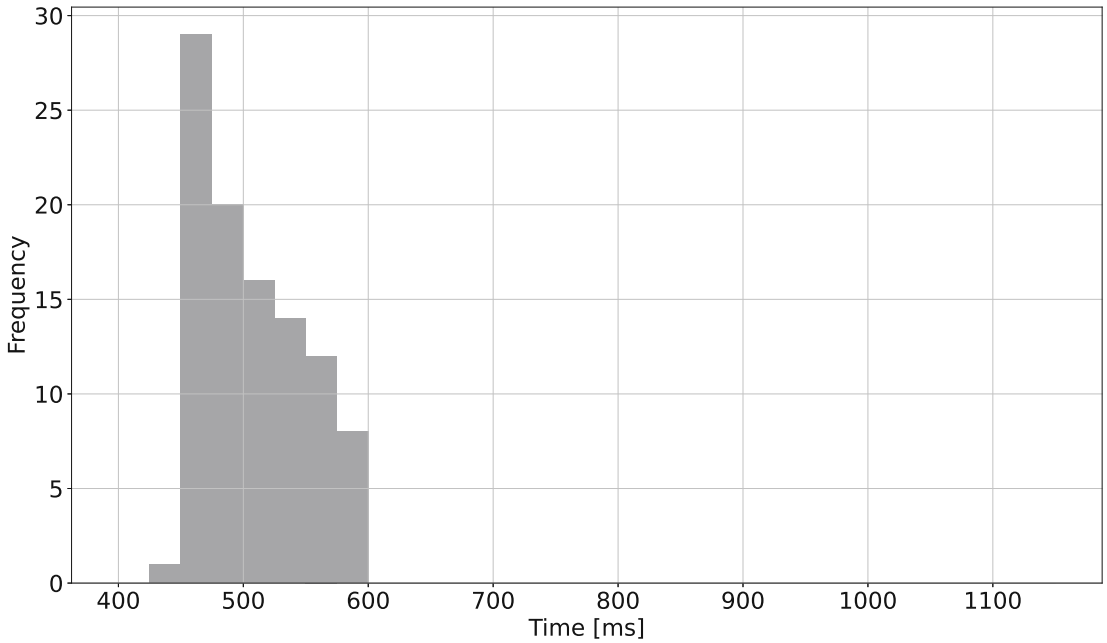


Figure 6.17: Histogram of deployment duration with 20 communicationSets for 10 Mbit/s network

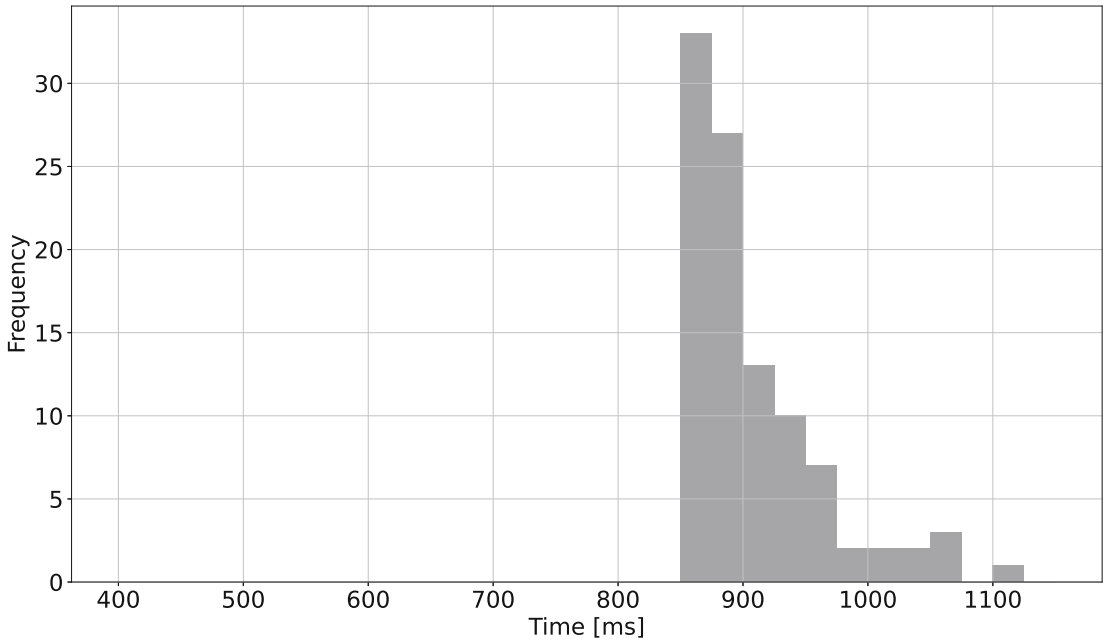


Figure 6.18: Histogram of deployment duration with 40 communicationSets for 10 Mbit/s network

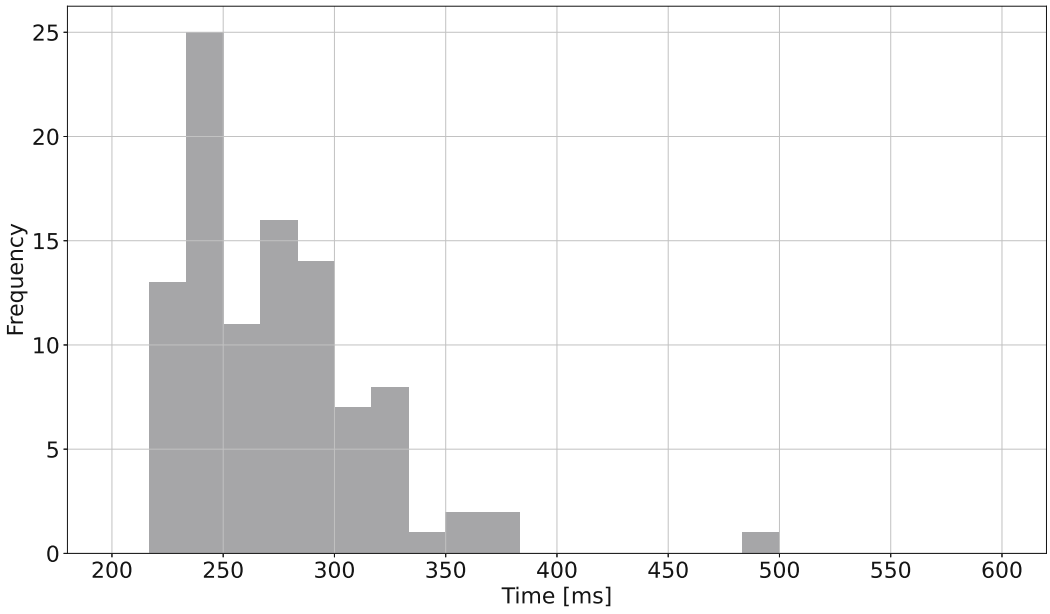


Figure 6.19: Histogram of deployment duration with 20 communicationSets for 100 Mbit/s network

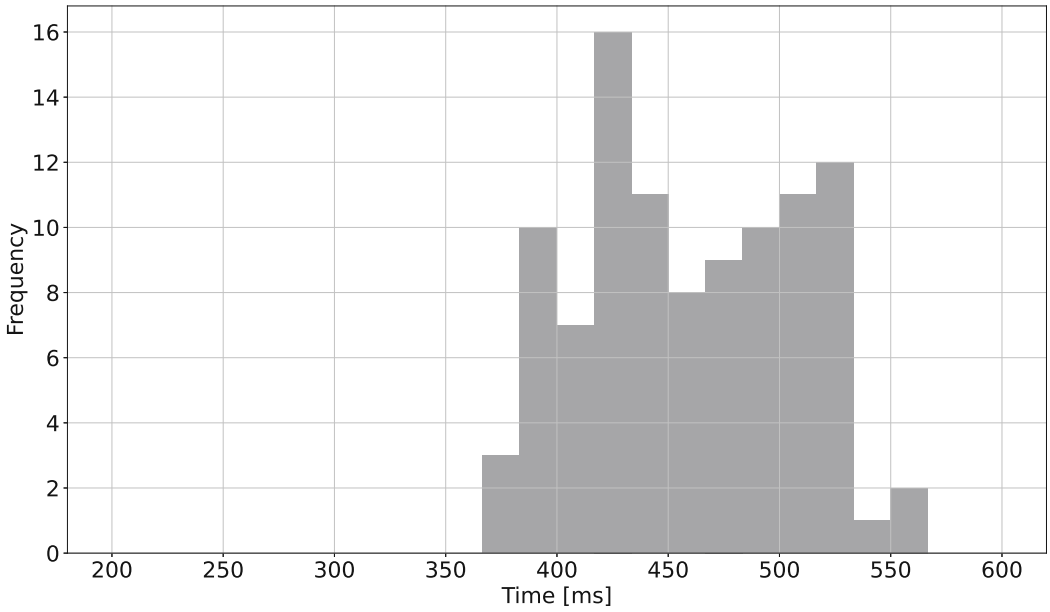


Figure 6.20: Histogram of deployment duration with 40 communicationSets for 100 Mbit/s network

CHAPTER 7

Conclusion & Outlook

Today's manufacturing systems are required to perform the production of small batches or even lot-size one. This makes it necessary to use flexible systems which can be quickly adjusted for new products. RMSs can fulfill those requirements, but the safety systems at RMSs are still static. Some use cases require additionally a safety system, which can be reconfigured with minimal downtime. This makes it necessary to extend a conventional safety system, which consists of a safety sensor, the safety logic, and the safety actuator, by an interoperable network. RSSs use the network to flexibly connect the three conventional parts of a safety system. This allows adjusting the safety system according to the current requirements with minimal downtime.

In this work, we showed how a safety stack consisting of OPC UA Safety, OPC UA PubSub, and a TSN network can be configured with one central tool. The central deployment tool uses OPC UA RPCs to configure the OPC UA server and NETCONF to configure the TSN network. Nevertheless, there are still some problems with the standardization of these interfaces. The OPC UA Safety standard [8] defines an SPI for configuration. Unfortunately, the mechanism for setting the parameters over that interface is vendor-specific and not part of the standard. Therefore, it is impossible to configure OPC UA Safety with a hardware-independent tool. Additionally, the SPI does not define a parameter to enable or disable a SafetyConsumer or a SafetyProvider over that interface, which is required for an RSS. It can be just hoped that the OPC UA Safety standard gets extended by a proper configuration interface. The configuration of the OPC UA PubSub and TSN bridges worked as defined in the standard. However, the standardized YANG models [29] for TSN are available in different revisions. These revisions often differ significantly, which requires an implementation of each revision at the deployment tool to be compatible with all hardware vendors. Because of the limitation of the OPC UA Safety and the fact that there is no OPC UA Safety stack publicly available, it was left out of the evaluation.

For the evaluation, we used an emulated ns-3 network and a hardware network with 10 Mbit/s and 100 Mbit/s. For the NETCONF and the OPC UA servers, we used netopeer2 [22] and open62541 [4] and let them run in multiple LXCs. The first evaluation was performed with an emulated ns-3 network on a single host. It turned out that the number of communicationSets with an emulated ns-3 network is limited by the host's performance. Therefore, the second evaluation was performed with a hardware network. The evaluation showed that the bandwidth has a different influence on the duration of all deployment steps. While a change from a 10 Mbit/s network to a 100 Mbit/s network speeds up the enablement of 80 Publishers by factor 5, it only speeds up the creation of 40 Subscribers only by factor 1.3.

Because the available TSN bridges are limited on the market, the evaluation with hardware devices is left for future work. A network, which consists of TSN switches and OPC UA servers from different vendors, could be built. This network can be compared with the devices simulated in the LXC containers.

Another part that can be done in future work is implementing and evaluating use cases defined in Chapter 4 since in this work, only the *Initial commissioning* use case was closer discussed. In that future work, the main focus could be reconfiguring an RSS during runtime. For that, it could be necessary to try different sequences of the deployment process or a different parallelization of the deployment steps.

As mentioned in this work, the deployment tool was designed to be implemented in a self-organizing safety system model, as shown in Figure 3.1. The integration of the deployment tool, which includes integrating an interface to the KBS, is also left for future work.

List of Figures

1.1 Safety Function in an RMS 2

2.1 The foundation of OPC UA 6
2.2 OPC UA address space 7
2.3 PubSub overview 9
2.4 OPC UA PubSub state machine 10
2.5 OPC UA Safety stack 11
2.6 OPC UA SafetyConsumer state machine 12
2.7 Transmission selection with gate control list 13
2.8 Fully centralized model 15
2.9 NETCONF layers 16
2.10 NETCONF datastore 17

3.1 Self-organizing safety system model 23
3.2 OPC UA TSN communication architecture 24
3.3 SDN overview 26

4.1 Use case diagram 28
4.2 Three layer safety stack 30
4.3 Process data of safety devices 31
4.4 Central deployment architecture 41

5.1 Structure of the deployment tool interfaces 44
5.2 TSN deployment 47
5.3 Sequence diagram add safety connections 49
5.4 Add new Publisher 50
5.5 Add new Subscriber 50
5.6 Enable new Publisher 50
5.7 Fault Tree Analysis 53
5.8 Activity diagram initial commissioning 55

6.1 Structure of ns-3 evaluation environment 60
6.2 Connection between LXC and ns-3 61
6.3 Duration of deployment for ns-3 network 64
6.4 Duration of deployment until 40 communicationSets for ns-3 network . . . 64

6.5 Duration of enabling the OPC UA Publishers for ns-3 network 65
6.6 Distribution of duration 66
6.7 Histogram of deployment duration with 20 communicationSets for ns-3 network 67
6.8 Histogram of deployment duration with 40 communicationSets for ns-3 network 67
6.9 Structure of hardware evaluation environment 68
6.10 Compare ns-3 with hardware evaluation environment 69
6.11 Deployment duration for 10 Mbit/s network 70
6.12 Deployment duration for 100 Mbit/s network 70
6.13 Duration of enabling the OPC UA Publishers for 10 Mbit/s network . . . 71
6.14 Duration of enabling the OPC UA Publishers for 100bit/s network 71
6.15 Distribution of duration for 10 Mbit/s network 73
6.16 Distribution of duration for 100 Mbit/s network 73
6.17 Histogram of deployment duration with 20 communicationSets for 10 Mbit/s network 74
6.18 Histogram of deployment duration with 40 communicationSets for 10 Mbit/s network 74
6.19 Histogram of deployment duration with 20 communicationSets for 100 Mbit/s network 75
6.20 Histogram of deployment duration with 40 communicationSets for 100 Mbit/s network 75

List of Tables

2.1 TSN standards overview 14

3.1 Ethernet and safety protocols 22

4.1 Parameters for TSN 802.1AS of one bride 32
4.2 Struct PortDataSet 32
4.3 Struct TSN time 33
4.4 Parameters for one TSN bridge 33
4.5 Struct Component 33
4.6 Struct VLAN 34
4.7 Standard parameters for a TSN bridge interface 34
4.8 TSN parameters for a TSN bridge interface 35
4.9 Struct Traffic-class-v2 35
4.10 Struct SDU size 35
4.11 Struct admin-control-list 36
4.12 Parameters for OPC UA PublishedDataSet 36
4.13 Struct of a NodeID 36
4.14 Parameters for one OPC UA Publisher 37
4.15 Parameters for one OPC UA Subscriber 38
4.16 Parameters for one Metadata- and Target-variables of the OPC UA Subscriber 38
4.17 Parameters for OPC UA Safety 39

6.1 Qualitative tests for the deployment tool 62
6.2 Qualitative tests for the deployment tool 66
6.3 Duration in ms of single steps for 20 communicationSets 72
6.4 Duration in ms of single steps for 80 and 160 communicationSets 72

Acronyms

AMQP Advanced Message Queuing Protocol. 6

API Application Programming Interface. 15

CIP Common Industrial Protocol. 22

CLI Command-Line Interface. 20

CNC Centralized Network Configuration. 14, 15, 25

CPU Central Processing Unit. 69

CRC Cyclic Redundancy Check. 11

CUC Centralized User Configuration. 15, 25

ELD Engineering Line Diagram. 52

EPSG Ethernet Powerlink Standardization Group. 22

ETG EtherCAT Technology Group. 22

EtherCAT Ethernet for Control Automation Technology. 22

FMEA Failure Mode and Effects Analysis. 51, 52

FTA Fault Tree Analysis. 51, 52

gPTP generalized Precision Time Protocol. 32, 46

GUI Graphical User Interface. 25

GUID Globally Unique Identifier. 39

HAZOP Hazard and Operability Study. 52

HTTP HyperText Transfer Protocol. 6

IE Industrial Ethernet. 21, 22, 29, 84

IP Internet Protocol. 7, 8, 14, 18, 24, 25, 32, 35–38, 45

IT Information Technology. 1, 20, 25

KBS Knowledge-Based System. 22, 28, 78

LXC Linux Container. 57, 59, 60, 62, 63, 68, 78

MAC Media Access Control. 14, 33

MNR MonitoringNumber. 10, 11, 13, 30

MQTT Message Queuing Telemetry Transport. 6, 24

MSTP Multiple Spanning Tree Protocol. 33

NETCONF Network Configuration Protocol. xi, xiii, 5, 15–18, 20, 21, 25, 26, 32, 40, 43, 46, 52, 57–60, 62, 63, 65, 66, 68, 72, 77, 78

NIC Network Interface Card. 58

OPCFX Open Platform Communications Field eXchange. 23

OPC UA Open Platform Communications Unified Architecture. xi, xiii, 2–13, 20–25, 27, 29–31, 34–40, 43–46, 48, 52, 54, 57, 59, 60, 62–66, 68, 71, 72, 77–81

OSI Open Systems Interconnection. 5, 7, 13

OT Operation Technology. 15, 20

P&ID Piping and Instrumentation Diagram. 52

PCP Priority Code Point. 13, 30, 34, 35

PFD Probability of dangerous Failure on Demand. 10

PFH Probability of dangerous Failure per Hour. 10

PLC Programmable Logic Controller. 5, 23, 24

PNO PROFIBUS Nutzerorganisation e. V.. 22

PnP Plug-and-Produce. 23

RAPIEnet Real-time Automation Protocols for IE. 22

RMS Reconfigurable Manufacturing System. xiii, 1–3, 21, 22, 77, 79

RPC Remote Procedure Call. 15, 16, 18, 40, 77

RSS Reconfigurable Safety System. xi, xiii, 1–5, 20, 21, 27, 29, 30, 51, 52, 54, 59, 63, 77, 78

RSTP Rapid Spanning Tree Protocol. 25, 33

SAPI Safety Application Program Interface. 10, 13, 31, 46

SBI SouthBound Interface. 25, 26, 40

SCL Safety Communication Layer. 10, 11

SDN Software Defined Networking. 25, 26, 40

SDU Service Data Unit. 34, 35, 81

SIL Safety Integrity Level. 10, 39

SNI Safety Network International e.V.. 22

SNMP Simple Network Management Protocol. 20

SPARQL SPARQL Protocol And RDF Query Language. 43

SPDU Safety Protocol Data Unit. 10, 11, 13, 30, 31, 35, 39, 47

SPI Safety Parameter Interface. 39, 45, 77

SSH Secure Shell. 16, 20, 63, 65, 69

STP Spanning Tree Protocol. 33

TCP Transmission Control Protocol. 45, 54, 62, 63, 65, 66, 69

TDMA Time Division Multiple Access. 13

TSN Time-Sensitive Networking. xi, xiii, 4, 5, 13–15, 20–23, 25–35, 40, 43–48, 54, 57–59, 62, 65, 66, 72, 77–79, 81

UADP Unified Architecture Datagram Protocol. 6

UDP User Datagram Protocol. 6–8, 23, 30, 31

VLAN Virtual Local Area Network. 14, 30, 31, 33, 34, 81

XML Extensible Markup Language. 6, 16, 20

Bibliography

[1] About ns-3. https://www.nsnam.org/about/. Accessed: 2022-07-23.

[2] Application of TSN in EtherNet/IP Networks. https://iebmedia.com/technology/industrial-ethernet/application-of-tsn-in-ethernet-ip-networks/. Accessed: 2022-08-14.

[3] Hirrschmann whit paper: TSN – Time Sensitive Networking". https://www.belden.com/dfsmedia/f1e38517e0cd4caa8b1acb6619890f5e/7897-source. Accessed: 2022-01-01.

[4] OPC UA stack. https://open62541.org/. Accessed: 2021-12-20.

[5] OPC Unified Architecture Field eXchange (UAFX) Part 80: UAFX Overview and Concepts. Release Candidate 1.00.00 RC3 2022-05-30.

[6] OPC Unified Architecture Part 1: Overview and Concepts. Version 1.04 released on 2017-11-22.

[7] OPC Unified Architecture Part 14: PubSub. Version 1.04 released on 2018-02-06.

[8] OPC Unified Architecture Part 15: Safety. Version 1.05.00 released on 2021-11-10.

[9] Time-Sensitive Networking (TSN) Task Group . https://1.ieee802.org/tsn/. Accessed: 2021-12-25.

[10] Local and metropolitan area networks– Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks– Corrigendum 2: Technical and Editorial Corrections. *IEEE Std 802.1AS-2011/Cor 2-2015 (Corrigendum to IEEE Std 802.1AS-2011)*, pages 1–13, 2016.

[11] IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic. *IEEE Std 802.1Qbv-2015 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, and IEEE Std 802.1Q-2014/Cor 1-2015)*, pages 1–57, 2016.

[12] IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks – Amendment 31: Stream Reservation Protocol (SRP) Enhancements and Performance Improvements. *IEEE Std 802.1Qcc-2018 (Amendment to IEEE Std 802.1Q-2018 as amended by IEEE Std 802.1Qcp-2018)*, pages 1–208, 2018.

[13] Adel Aldalbahi, Michael Rahaim, Abdallah Khreishah, Moussa Ayyash, Ryan Ackerman, James Basuino, Walter Berreta, and Thomas D.C. Little. Extending ns3 to simulate visible light communication at network-level. In *2016 23rd International Conference on Telecommunications (ICT)*, pages 1–6, 2016.

[14] Masood Ashraf and Faisal Hasan. Configuration selection for a reconfigurable manufacturing flow line involving part production with operation constraints. *International journal of advanced manufacturing technology*, 98(5-8):2137–2156, 2018.

[15] M. Bjorklund. YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF). RFC 6020, RFC Editor, October 2010. `http://www.rfc-editor.org/rfc/rfc6020.txt`.

[16] M. Bjorklund, J. Schoenwaelder, P. Shafer, K. Watsen, and R. Wilton. Network Management Datastore Architecture (NMDA). RFC 8342, RFC Editor, March 2018.

[17] Zigmund Bluvband and Pavel Grabov. Failure analysis of FMEA. In *2009 Annual Reliability and Maintainability Symposium*, pages 344–347, 2009.

[18] Gabriela Bogdanovská and Marcela Pavlíčková. Hazard and operability study. In *2016 17th International Carpathian Control Conference (ICCC)*, pages 82–85, 2016.

[19] Dietmar Bruckner, Marius-Petru Stănică, Richard Blair, Sebastian Schriegel, Stephan Kehrer, Maik Seewald, and Thilo Sauter. An Introduction to OPC UA TSN for Industrial Communication Systems. *Proceedings of the IEEE*, 107(6):1121–1131, 2019.

[20] Achmad Buchori, Punaji Setyosari, I Wayan Dasna, and Saida Ulfa. Mobile augmented reality media design with waterfall model for learning geometry in college. *International Journal of Applied Engineering Research*, 12(13):3773–3780, 2017.

[21] Javier Bustos-Jiménez, Rodrigo Alonso, Camila Faúndez, and Hugo Méric. Boxing experience: Measuring QoS and QoE of multimedia streaming using NS3, LXC and VLC. In *39th Annual IEEE Conference on Local Computer Networks Workshops*, pages 658–662, 2014.

[22] CESNET. netopeer2. `https://github.com/CESNET/netopeer2`, 2022. [Online; accessed 13-June-2022].

[23] CESNET. sysrepo. `https://netopeer.liberouter.org/doc/sysrepo/master/html/`, 2022. [Online; accessed 13-June-2022].

[24] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman. Network Configuration Protocol (NETCONF). RFC 6241, RFC Editor, June 2011. http://www.rfc-editor.org/rfc/rfc6241.txt.

[25] Dieter Etz, Thomas Frühwirth, and Wolfgang Kastner. Self-Configuring Safety Networks. In *Technologien für die intelligente Automation*, pages 232–245. Springer Berlin Heidelberg, 2020.

[26] Dieter Etz, Thomas Frühwirth, Ahmed Ismail, and Wolfgang Kastner. Simplifying functional safety communication in modular, heterogeneous production lines. In *2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)*, pages 1–4, 2018.

[27] Dieter Etz, Thomas Frühwirth, and Wolfgang Kastner. Flexible Safety Systems for Smart Manufacturing. In *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 1123–1126, 2020.

[28] Sascha Gent, Pablo Gutiérrez Peón, Thomas Frühwirth, and Dieter Etz. Hosting functional safety applications in factory networks through Time-Sensitive Networking. In *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 230–237, 2020.

[29] github. yang. https://github.com/YangModels/yang/tree/main/standard/ieee/draft/802.1/Qcw/, 2022. [Online; accessed 06-June-2022].

[30] Google. GO. https://go.dev/, 2022. [Online; accessed 06-June-2022].

[31] Richard F. Griffiths. HAZOP and HAZAN: Notes on the identification and assessment of hazards : by T.A. Kletz, Institution of Chemical Engineers, Rugby, 1983, ISBN 0-85295-165-5, 81 pages, paperback, £8.00 incl. postage and packing. *Journal of Hazardous Materials*, 8:385–386, 1984.

[32] Jose Miguel Gutierrez-Guerrero and Juan Antonio Holgado-Terriza. Automatic Configuration of OPC UA for Industrial Internet of Things Environments. *Electronics*, 8(6), 2019.

[33] Marina Gutiérrez, Astrit Ademaj, Wilfried Steiner, Radu Dobrin, and Sasikumar Punnekkat. Self-configuration of IEEE 802.1 TSN networks. In *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8, 2017.

[34] Lorin Hochstein and Rene Moser. *Ansible: Up and Running: Automating configuration management and deployment the easy way.* " O'Reilly Media, Inc.", 2017.

[35] IEC. Communication networks and systems for power utility automation. Technical Report IEC 61850, The International Electrotechnical Commission, 2020.

[36] ISO. Safety of machinery — General principles for design — Risk assessment and risk reduction. Technical Report ISO 12100:2010, The International Organization for Standardization, 2010.

[37] Thomas Kobzan, Immanuel Blöcher, Maximilian Hendel, Simon Althoff, Alissa Gerhard, Sebastian Schriegel, and Jürgen Jasperneite. Configuration Solution for TSN-based Industrial Networks utilizing SDN and OPC UA. In *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 1629–1636, 2020.

[38] Hermann Kopetz. *The Real-Time Environment*, pages 1–28. Springer US, Boston, MA, 2011.

[39] Yoram Koren, Xi Gu, and Weihong Guo. Reconfigurable manufacturing systems: Principles, design, and future trends. *Frontiers of Mechanical Engineering*, 13(2):121–136, Jun 2018.

[40] Thomas Kunz and Karpakamurthy Muthukumar. Comparing OpenFlow and NETCONF when interconnecting data centers. In *2017 IEEE 25th International Conference on Network Protocols (ICNP)*, pages 1–6, 2017.

[41] W. S. Lee, D. L. Grosh, F. A. Tillman, and C. H. Lie. Fault Tree Analysis, Methods, and Applications - A Review. *IEEE Transactions on Reliability*, R-34(3):194–203, 1985.

[42] Tomas Lennvall, Mikael Gidlund, and Johan Åkerberg. Challenges when bringing iot into industrial automation. In *2017 IEEE AFRICON*, pages 905–910, 2017.

[43] Yuting Li, Junhui Jiang, Changdae Lee, and Seung Ho Hong. Practical Implementation of an OPC UA TSN Communication Architecture for a Manufacturing System. *IEEE Access*, 8:200100–200111, 2020.

[44] Zepeng Liu and Patrick Bellot. OPC UA PubSub Implementation and Configuration. In *2019 6th International Conference on Systems and Informatics (ICSAI)*, pages 1063–1068, 2019.

[45] Wolfgang Mahnke, Stefan-Helmut Leitner, and Matthias Damm. *OPC Unified Architecture*. Springer, Berlin, 2009.

[46] Thomas Marangoni. Device and Link Discovery in Industrial Ethernet Networks. Bachelor's thesis, Technische Universität Wien, 2022.

[47] Alexander Mildner. Time Sensitive Networking for Wireless Network-A State of the Art Analysis. *Network*, 33, 2019.

[48] Marek Moravcik, Pavel Segec, Martin Kontsek, Jana Uramova, and Jozef Papan. Comparison of LXC and Docker Technologies. In *2020 18th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, pages 481–486, 2020.

[49] Nikolas Mühlbauer, Erkin Kirdan, Marc-Oliver Pahl, and Georg Carle. Open-Source OPC UA Security and Scalability. In *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 262–269, 2020.

[50] Maryam Pahlevan and Roman Obermaisser. Evaluation of Time-Triggered Traffic in Time-Sensitive Networks Using the OPNET Simulation Framework. In *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, pages 283–287, 2018.

[51] Maryam Pahlevan, Jonas Schmeck, and Roman Obermaisser. Evaluation of TSN Dynamic Configuration Model for Safety-Critical Applications. In *2019 IEEE Intl Conf on Parallel Distributed Processing with Applications, Big Data Cloud Computing, Sustainable Computing Communications, Social Computing Networking (ISPA/BDCloud/SocialCom/SustainCom)*, pages 566–571, 2019.

[52] Julius Pfrommer, Andreas Ebner, Siddharth Ravikumar, and Bhagath Karunakaran. Open Source OPC UA PubSub Over TSN for Realtime Industrial Communication. In *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 1087–1090, 2018.

[53] Martin Rostan, Joseph E. Stubbs, and Dmitry Dzilno. EtherCAT enabled advanced control architecture. In *2010 IEEE/SEMI Advanced Semiconductor Manufacturing Conference (ASMC)*, pages 39–44, 2010.

[54] Jean-Pierre Signoret and Alain Leroy. Hazard and Operability Study (HAZOP). In *Reliability Assessment of Safety and Production Systems*, pages 157–164. Springer, 2021.

[55] Luis Silva, Paulo Pedreiras, Pedro Fonseca, and Luis Almeida. On the adequacy of SDN and TSN for Industry 4.0. In *2019 IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC)*, pages 43–51, 2019.

[56] Liangliang Sun, Yan-Fu Li, and Enrico Zio. Comparison of the HAZOP, FMEA, FRAM, and STPA Methods for the Hazard Analysis of Automatic Emergency Brake Systems. *ASCE-ASME J Risk and Uncert in Engrg Sys Part B Mech Engrg*, 8(3), 10 2021. 031104.

[57] Shuo Tian and Yihong Hu. The Role of OPC UA TSN in IT and OT Convergence. In *2019 Chinese Automation Congress (CAC)*, pages 2272–2276, 2019.

[58] Hugh A Watson et al. Launch control safety study. *Bell labs*, 1961.

[59] Ming Yang and Guang Li. Analysis of PROFINET IO Communication Protocol. In *2014 Fourth International Conference on Instrumentation and Measurement, Computer, Communication and Control*, pages 945–949, 2014.

[60] Suiran Yu, Jiwen Liu, Qingyan Yang, and Minxian Pan. A comparison of FMEA, AFMEA and FTA. In *The Proceedings of 2011 9th International Conference on Reliability, Maintainability and Safety*, pages 954–960, 2011.

[61] Zefeng Zhou and Guochu Shou. An Efficient Configuration Scheme of OPC UA TSN in Industrial Internet. In *2019 Chinese Automation Congress (CAC)*, pages 1548–1551, 2019.