Università di Genova | **DIBRIS** DIPARTIMENTO DI INFORMATICA, BIOINGEGNERIA, ROBOTICA E INGEGNERIA DEI SISTEMI

Enhancing Security in Industrial Control Systems Through Programmable Kernel-Level Microsegmentation

Milo Galli

Master Thesis

Università di Genova, DIBRIS
Via Opera Pia, 13 16145 Genova, Italy
https://www.dibris.unige.it/

Università di Genova

MSc Computer Science
Software Security and Engineering Curriculum

Enhancing Security in Industrial Control Systems Through Programmable Kernel-Level Microsegmentation

Milo Galli

Advisor: Enrico Russo

Examiner: Giovanni Lagorio

March, 2025

Abstract

This work presents a comprehensive perspective on an alternative to contemporary naval environments' internal networking management. Specifically, it explores the rationale and decisions that guided the implementation of a Layer 7 switch utilizing the Rust programming language and AF_XDP technology. Additionally, it addresses the imperative for working with data that aligns with legacy systems' standards, precluding the introduction of additional hardware components or novel software communication standards. Finally, a comparative analysis will be conducted between our solution's performance and that of the state-of-the-art model, highlighting the distinctions, advantages, and disadvantages.

Table of Contents

Chapter 1

Introduction

1.1 Motivation and Context

The motivation behind our effort is trying to provide a drop-in solution to Operational Techonlogy (OT) systems' main problem: protocols vulnerabilities in system-to-system communication.
OT systems are hardware and software solutions that monitor and control physical devices, processes, and infrastructure in industrial environments.
The aforementioned are characterized by a internal communication structure that is completely unrestricted, unfiltered and omnidirectional, thereby facilitating uninhibited behavior within the network. Finding a solution to this problem results in a particularly hard task because of OT systems' nature:

- The composition is characterized by the presence of legacy code, the optimization of which can be a laborious and time-consuming process that necessitates a considerable investment of resources
- The communication protocols used are entirely devoid of any encryption mechanism, thereby rendering information exchange remarkably vulnerable and unreliable
- The implementation of substantial modifications is not a straightforward process, as they are static assets that play a crucial role of fundamental systems that are indispensable to our daily lives

Leveraging these characteristics such solution should be **retrocompatible, transparent and should not introduce additional overhead**.

1.2 Goals

The goal of this work is to implement a simulation of a OT Naval system which leverages a Layer7 software switch with the following features:

- should be agnostic to the underlying software and hardware of the running architecture with the ability to be integrated seamlessly into existing infrastructures;
- should be capable of abstracting OT systems' components leveraging also policy rules that regulates communication among them;
- should preserve normal communication flow for messages that do not conform to "National Marine Electronics Association (NMEA)";
- should be able to process, filter and drop NMEA sentences considering policy rules previously defined;
- preserve or enhance legacy solutions' performances

The architecture we aim to build can be visualized as follows:

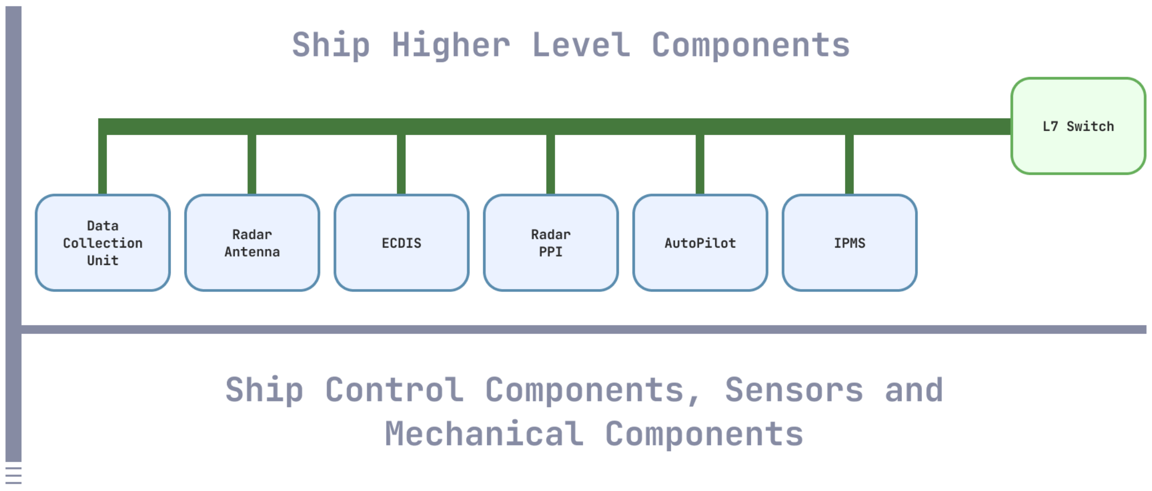
Ship Higher Level Components
L7 Switch
Data Collection Unit
Radar Antenna
ECDIS
Radar PPI
AutoPilot
IPMS
Ship Control Components, Sensors and Mechanical Components

Figure 1.1: Project architecture visualization

The final implementation will be, as displayed above, a tool whose purpose will be managing higher level ship component network traffic with the ability to forward, filter and drop information flowing inside of the vessel.

1.3 Content of the thesis

The first topics presented are relevant background information (chapter 2) of Naval Networks and Protocols, Software Defined Networking and the underlying technologies of the L7-Switch, EBPF and AFXDP.
Next the Switch's architecture and functionalities will be analyzed (chapter 3) discussing the normal packet flow and the filtered one.
Following implementation choices will be reviewed (chapter 4) exploring the decisions behind the project's structure and pondering the resulting performances (chapter 5). Lastly a selection of existing works and proposed solutions will be presented considering fundamental aspects of each one (chapter 6).

Chapter 2

Background

In this chapter, we delineate the principles of contemporary naval communications, emphasizing the characteristics and deficiencies of prevailing standards (NMEA). The subsequent discussion will address the necessity for "software-defined networking" and the innovative technologies underpinning network administration, EBPF and AF_XDP.

2.1 Naval Networks

A naval network, otherwise referred to as an **integrated shipboard communication system** is a sophisticated and meticulously organized system designed to facilitate communication, data exchange, and operational coordination whithin subsystems and devices on the vessel. Given the numerous elements that contribute to the ship's apparatus, such as navigation systems, sensors and monitoring systems our focus will be directed exclusively towards the communication limb that directly interests our final goal.

2.1.1 Communication systems

Communication systems facilitate seamless communication of the vessel both internally and externally ensuring the continuous exchange of data. The field of external communication can be subdivided into three primary categories :

- **Command and Control (C2)**,for long-range capabilities such as communication with shore-based command centers, other ships and aircrafts
- **Tactical Coordination**, for short-to-medium range communication which are critical for coordination and emergency communications

- **Emergency Communication**,for real time data exchange among various components allowing coordinated operations among ships, aircrafts and ground entities

External side communication's capstone can be identified in the usage of the Satellite Communication (SATCOM) protocol that enables beyond-line-of-sight (BLOS) data exchange providing global, high-bandwidth, and secure connectivity. However, it is part of a broader ecosystem that includes radio communication, data links, and underwater communication systems.
Internal communication, on the other hand, although it comprises crew coordination systems the main body consists of devices that provide a unified picture among navigation, propulsion and management environments. Vessels are designed to establish an informational circulatory system within them that consists of a physical apparatus composed of various components such as switches, routers, gateways, and servers. The flow of information within it falls into five distinct categories:

- **Navigation and Situational Awareness**, data transmitted over the network and integrated into the Management System, providing a real-time operational picture
- **Engine Monitoring and Control**, data allowing the crew to monitor and optimize the ship's propulsion and power systems
- **Crew Communication**, intercom system's data to coordinate crew actions

2.1.2 NMEA

Given the wide range of the aforementioned messages in the early years of 1980 the **National Marine Electronics Association (NMEA)** introduced the first kind of protocol to standardize communication for marine electronics. Since its inception, the protocol has evolved significantly to accommodate advancements in technology and the expanding needs of the naval and gps fields.

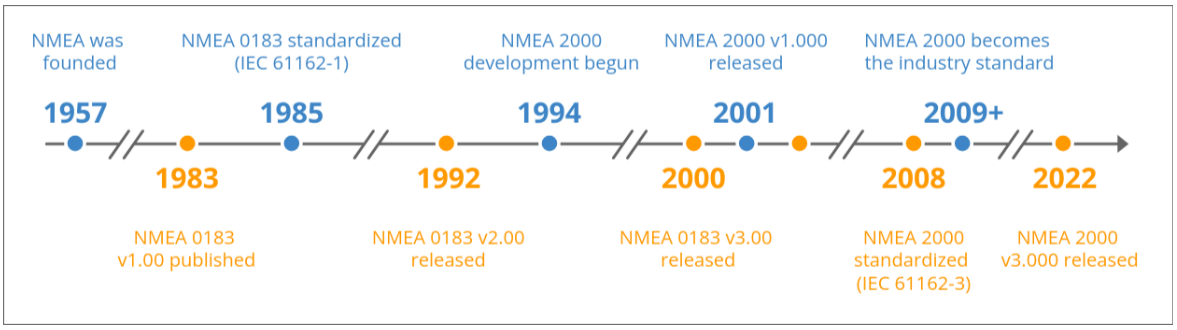
NMEA was founded
1957
NMEA 0183 standardized (IEC 61162-1)
1985
NMEA 2000 development begun
1994
NMEA 2000 v1.000 released
2001
NMEA 2000 becomes the industry standard
2009+
1983
NMEA 0183 v1.00 published
1992
NMEA 0183 v2.00 released
2000
NMEA 0183 v3.00 released
2008
NMEA 2000 standardized (IEC 61162-3)
2022
NMEA 2000 v3.000 released

Figure 2.1: NMEA Protocols' Creation timeline

NMEA 0183

The protocol underwent a series of modifications, beginning with the 0180 and 0182 versions. However, these initial implementations were swiftly discarded in favor of the subsequent iteration, the 0183. This new version's adoption as the primary standard for serial communications can be attributed to its simplicity and comprehensive functionality, providing the vessel's components a complete overview of other one's status through simple text.

This standard utilizes a rudimentary ASCII serial communications protocol, which delineates a method where data are transmitted in a **sentence** from a single **talker** to multiple **listeners** concurrently. The protocol enables a talker to engage in a unidirectional conversation with an almost limitless number of listeners by employing intermediate expanders, and it facilitates communication between multiple sensors and a singular port through the use of multiplexers.

Although the protocol stipulates the use of RS-422 for electrical transport, a de facto standard has emerged in which sentences are **encapsulated in UDP datagrams** and **transmitted over IP networks** taking the name of LightWeight Ethernet (LWE).

2.1.2.0.1 Message structure

All messages must be composed of printable ASCII characters between *0x20* (space) to *0x7e* (tilde) including a set of special characters that define the syntax

Syntax characters		
ASCII	HEX	USE
<CR>	0x0d	Carriage return
<LF>	0x0a	Line feed, end delimiter
!	0x21	Start of encapsulation sentence delimiter
$	0x24	Start delimiter
*	0x2a	Checksum delimiter
,	0x2c	Field delimiter
\	0x5c	TAG block delimiter
^	0x5e	Code delimiter for HEX representation of ISO/IEC 8859-1 (ASCII) characters
~	0x7e	Reserved

Moreover, adherence to established syntax rules is imperative for the purpose of ensuring compliance with the protocol, in particular :

- Messages have a maximum length of 82 characters, including the $ or ! starting character and the ending <LF>
- The start character for each message can be either a $(For conventional field delimited messages) or ! (for messages that have special encapsulation in them)
- The next five characters identify the **Talker Id** (two characters) and the **Sentence Type** (three characters)
- All data fields that follow are comma-delimited
- Where data is unavailable, the corresponding field remains blank (it contains no character before the next delimiter
- The first character that immediately follows the last data field character is an asterisk, but it is only included if a checksum is supplied
- The asterisk is immediately followed by a checksum represented as a two-digit hexadecimal number. The checksum is the bitwise exclusive OR of ASCII codes of all characters between the $ and *, not inclusive.
- <CR><LF> ends the message

The following is a list of the most common messages that appeared in the early days of this technology, constituting the first kind of standardized communication:

NMEA 0183 Common Sentences	
Sentence Type	Description
$Talker ID+GGA	Global Positioning System Fixed Data
$Talker ID+GLL	Geographic Position—Latitude and Longitude
$Talker ID+GSA	GNSS DOP and active satellites
$Talker ID+GSV	GNSS satellites in view
$Talker ID+RMC	Recommended minimum specific GPS data
$Talker ID+VTG	Course over ground and ground speed

2.1.2.0.2 Protocol security

The protocol's conception predates the conceptualization of vehicle hacking, thus lacking any inherent security measures to protect network information flows from potential attacks. To date, the research community has conducted only a limited number of studies on the security of network marine protocols. Moreover, there is a paucity of well documented research on risk analysis for this type of communication [KTT21].

As is the case with other networking protocols, marine protocols are susceptible to malicious attacks and unintentional human errors. It is incumbent upon manufacturers to consider the extended Confidentiality, Integrity, and Availability (CIA) model, which encompasses Authentication, Authorization, and Non-repudiation from the creation to implementation and maintenance of network device firmware.

As of the present date, only naive solutions have been implemented in order to address these primary security concerns :

- **Confidentiality** : Due to the limitation of bandwidth in many marine network protocols, encryption mechanisms are not implemented. Consequently, the predominant approach to ensure confidentiality for these systems is primarily through access control.
- **Integrity** : Vessels generate substantial surges in network traffic, which often surpasses the bandwidth capacity of the ship. This phenomenon can result in service disruptions and latency as various services compete for bandwidth. One proposed solution to this challenge is the implementation of a master device in conjunction with multiple remote devices. In instances where a high-priority bandwidth requirement is identified, a remote device can be temporarily allocated to an auxiliary channel, thereby facilitating the management of network traffic.
- **Availability** : The majority of maritime communication protocols are designed to support slow transfer speeds. Therefore, the protection for availability is contingent upon the hardware specification and configuration, allowing for data transfer at the maximum data rate of the protocol. In the context of critical systems that necessitate uninterrupted accessibility, the implementation of hardware redundancy and backup mechanisms becomes imperative.
- **Authentication** : A select number of shipping protocols incorporate optional authentication mechanisms that are not specialized for the protocols themselves. These include the Global Navigation Satellite System (GNSS) and systems adopted from vehicular networks. Consequently, under theory, these protocols possess the potential to be adapted to support authentication for marine networks; however, this functionality is not currently available.
- **Authorization** : A considerable number of protocols incorporate a form of an ID tag to identify the communication nodes, along with their privileges, as well as access rights, inside a network. However, in the absence of adequate inspection mechanisms,

a malicious device can potentially generate its own ID with a high level of authorization, thereby circumventing access controls and gaining unauthorized access to sensitive information.

- **Non-Repudiation** : This concern has witnessed a marked rise in prominence, particularly in the context of forensics and legal challenges. Non-repudiation is poised to assume greater significance; however, at present, no non-repudiation technique has been implemented in any substantial manner.

As the protocol undergoes continuous development and enhancement, **theoretical analyses** were conducted to evaluate the current state of data exchange security in response to prevalent attacks. The objective of this endeavor was to identify vulnerabilities that could be exploited by an attacker to compromise the protocol. The subsequent list of attacks is not intended to be exhaustive; rather, it is designed to illustrate the most critical violations that can be perpetrated in networks that utilize the protocol:

- **Denial of Service (DoS)**: targets the availability of data
- **Spoofing**: targets the integrity of data.
- **Packet sniffing**: targets the confidentiality of data.
- **Replay/Man-in-the-Middle (MITM)**: targets both confidentiality and integrity of data.

It is important to note that the NMEA 0183 protocol does not incorporate any mechanisms for authentication, encryption, or validation. The absence of security measures inherent to the protocol renders it vulnerable to any of the aforementioned attack if utilized effectively by an adversary

Attacks against the NMEA 0183 protocol	
Attack	Methodology
Denial of Service	NMEA 0183 supports a baud rate of 4800 (9600 bit-s/s). While being a ethernet specific this makes a DoS attack potentially effective against any devices using this protocol depending on the means through which data is transferred to an application
Spoofing/Packet Sniffing	Spoofing and sniffing has been shown to be trivial against the standard using simple traffic simulation software
Replay/Man-in-the-Middle	Leveraging software that that intercepts and manipulates data exchanged modifiying critical missions' informations such as coordinates coming from sensor devices

2.1.2.1 NMEA 2000

The fourth iteration of the protocol resulted in a more advanced implementation, characterized by more sophisticated structural choices that increased performance. Regrettably, the security standards remained unaltered.

2.1.2.1.1 Key changes

The backbone of this new version's enhancements can be identified in the standard communication being based on the **Controller Area Network (CAN)** Bus protocol. CAN protocol utilizes a priority-driven and message-based architecture capable of operating at either 512 Kbps or 128 Kbps.

Each device that transmits CAN Bus signals is considered a node with its own priority level and during transmission, nodes continuously check the signal on the bus comparing it with the signal they are transmitting. If a mismatch occurs, an error is raised and the internal error counter is incremented to prevent having broken nodes continuously flooding the network. The right of a node to raise errors can be disabled depending on the value of these counter granting that communication over CAN Bus is reliable.

The advent of this novel information flow structure has enabled the transmission of data messages at elevated speeds as data frames characterized by the incorporation of robust error checking mechanisms, confirmed frame delivery, and guaranteed latency times. Additionally, the transmission of information as binary messages, as opposed to ASCII text, facilitates more efficient and secure data exchange. This was done mantaining a relatively simple message structure organized as follows

- **GN (Parameter Group Number)** :Defines the type of data being transmitted
- **SPN (Suspect Parameter Number)**: Identifies specific data fields within a PGN
- **Data Length**: Specifies the length of the data in bytes

This implementation choice was made also in light of the following key advantages concerning the physical characteristics of this type of communication:

- **Simple and Cheap**: Components communicate via a single CAN system instead of via direct complex analog signal lines reducing errors, weight, wiring, and costs.
- **Fully centralized architecture** : Only one point of entry is set to communicate with all network components enabling central diagnostics, data logging, and configuration.
- **Extremely Robus and Efficient** : the system is robust towards electric disturbances and electromagnetic interference - ideal for safety-critical applications AN frames are prioritized by ID numbers. The top priority data gets immediate bus access, without causing interruption of other frames.

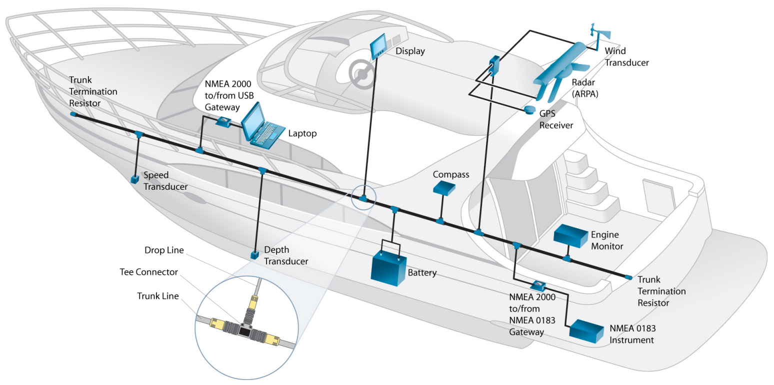

Figure 2.2: NMEA2000 Protocol with CAN Bus communication

2.1.2.1.2 Protocol Security

The NMEA 2000 standard did not incorporate any inherent security features related to confidentiality, authentication, authorization, or non-repudiation, which were not present in the previous versions. The implementation of controls for potential attack vectors is delegated to nodes, which utilize their own **application-specific** protocols operating on top of the marine network. Therefore, an analysis of the protocol itself, absent any top-level software add-on, reveals that the vulnerabilities identified in previous iterations have reemerged, albeit with modified attack methodologies.

Attacks against the NMEA 2000 protocol	
Attack	Methodology
Denial of Service	Leveraging the protocol's low-bandwidth design malicious devices or poorly configured ones could produce a large amount of traffic without raising errors on the main bus message channel, preventing other devices from communicating studies
Spoofing	Exploiting situations where the Parameter Group Number (PGN) used to identify the sending node is copied and used by a malicious device. This imposes more problems since PGN is also used to indicate the priority of messages.
Packet Sniffing	Since NMEA 2000 operates on a single broadcast domain where all nodes on the message channel receive all messages. Since the responsibility to discern which messages to discard as unneeded is delegated to nodes passive sniffing of messages is a trivial task, assuming physical access is possible.
Replay/Man-in-the-Middle	Considering the protocol's broadcast principle, an inline device could be placed between the target node and the rest of the channel (bus). This malicious inline device would be able to pass along messages from either the target node or the rest of the channel in either direction.

2.1.2.2 Considerations over the standard

NMEA 0183 and NMEA 2000 have achieved widespread acceptance among manufacturers and maritime agencies worldwide, receiving frequent updates and support.
Attempting to modify these standards would necessitate years of adjustments, aiming to align with current hardware infrastructures that are utilized on a daily basis by a wide range of essential services.
However, as previously emphasized, the absence of security measures necessitates the urgent development of a compatible and transparent solution for the naval ecosystem.

2.2 Software Defined Networking - SDN

The urgent pursuit of alternatives that would seamlessly integrate with current naval systems' architecture without introducing new communication standards has prompted the consideration of Software Defined Networking (SDN).

This technology can be defined as an approach to networking that uses software-based controllers or application programming interfaces (APIs) to communicate with underlying hardware infrastructure and direct traffic on a network.
Contrary to state-of-the-art solutions which utilize dedicated hardware devices (e.g., routers and switches), SDN possesses the capability to create and control a virtual network or to manage a traditional hardware configuration through the use of software.

This new way of intending data flow differs also from network virtualization where either a multitude of virtual networks is segmented within a single physical one in order to separate components or different physical networks are merged to create a single virtual one. In contrast, software-defined networking enables a new way of controlling the routing of data packets through a centralized server while leaving physical and virtual configurations unaltered.

2.2.1 Operational Process

The technology's paradigm relies on software rather than hardware for its operations. This inherent characteristic bestows upon it a superior degree of flexibility when compared with traditional networking methodologies.

With SDN administrators exercise comprehensive control over the network via a single pane of glass, modifying configuration settings, allocating resources, and augmenting network capacity.

Since software is decoupled from the hardware the three fundamental components of SDN can be situated in disparate physical locations:

- **Applications** which communicate resource requests or information about the network as a whole
- **Controllers** which use the information from applications to decide how to route data packets
- **Networking devices**, which receive information from the controller about where to move the data

Eventually, additional software components may be embedded in either the software or the hardware to assume the responsibilities of physical switches and consolidate their functions into a single, intelligent switch. The switch is responsible for verifying the integrity of data packets and their virtual machine destinations, thereby facilitating the efficient transmission of packets.

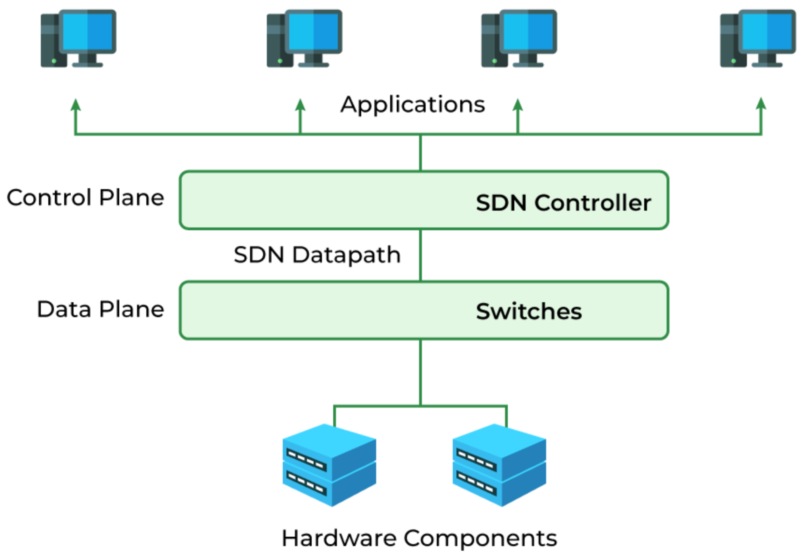

Figure 2.3: Example of Software Defined Networking structure

2.2.2 SDN Benefits

Software defined networking facilitates the seamless and regulated migration of data among distributed locations, a functionality that is paramount for systems comprising numerous components, each overseeing distinct workloads and transmitting signals across disparate domains. The subsequent aspects can be regarded as the pivotal characteristics that contribute to the technology's value and reliability:

- **Increased control with greater speed and flexibility** : Instead of manually programming multiple vendor-specific hardware devices, developers can control the flow of traffic over a network simply by programming an open standard software-based controller. Networking administrators also have more flexibility in choosing networking equipment, since they can **choose a single protocol** to communicate with any number of hardware devices through a central controller.
- **Customizable network infrastructure** : Administrators can configure network services and allocate virtual resources to change the network infrastructure in real time **through one centralized location**. This allows network administrators to optimize the flow of data through the network and prioritize applications that require more availability.
- Robust security: A software-defined network delivers visibility into the entire network, providing a more holistic view of security threats where operators can **create separate zones for devices that require different levels of security**, or immediately quarantine compromised devices so that they cannot infect the rest of the network.

2.2.3 Application to Legacy Systems

The concept of centralized software regulating the flow of data in switches and routers is applicable to all software-defined networking (SDN). However, there are distinct SDN models, each with its own unique application environments and customized to address specific requirements.

The solution that best fits the role of integrating modern data flow strategies while being limited by hardware and software standards is Hybrid SDN.

This model combines software-defined networking with traditional networking protocols in a single environment, thereby supporting diverse functions on a network. Standard networking protocols continue to direct some traffic, while SDN takes on responsibility for other traffic, **such as maritime message protocols**. This allows network administrators to introduce SDN in stages to a legacy environment.

This approach engenders a remarkably adaptable environment, wherein technology that is no longer supported interacts with cutting-edge innovations that address the deficiencies of outdated infrastructures. Considering that and with SDN being merely a structural paradigm the performance limitations that could be experienced are a result of the hardware and software solutions adopted to implement such systems.

As previously mentioned anyway, the objective of this initiative is not to incorporate any additional hardware; so rather, the emphasis will be on selecting the most optimal software solution available to serve as the foundation for the final SDN architecture's implementation. The solution under consideration is the AFXDP Socket technology that enables fast and efficient data processing bypassing the traditional data flow through the Linux Network Stack.

2.3 Next generation networking

The Linux Kernel Network Stack is in charge of all network communications, processes incoming and outgoing network packets, implements various network protocols and provides application interfaces for network interaction. It can be thought as a layered architecture which follows the Open Systems Interconnection (OSI) model processing data from the physical network interface up to the application layer.

2.3.1 Data Flow Path

Several networking concepts together build the layered architecture mentioned earlier, where data is transformed from raw packets into structured information, and key terminology must be known in order to grasp the inner complexity of the various steps, such as:

- **Ring Buffers** : Data structure particularly efficient to process data streams, continuously handling new data that overwrites the old one. Network Interface drivers uses them to allocate Receive (RX) and Transmission (TX) queues in the device memory.
- **Socket Buffers** : Data structure used to represent network packets used as the fundamental building block for data handling at kernel level, containing metadata and data in a layered format enabling partial editing as information gets higher in the abstraction layers.
- **Kernel Interrupts** : They are signals used to stop the CPU from what it is doing and work on the interrupter's part instead. They are divided into two kinds, top-half and bottom-half, differentiated by computational cost and calling reasons.

A brief but complete analysis of the entire process can be made once these concepts are internalized:

1. At Kernel boot up, the CPU allocates packet buffers (RX and TX buffers), and builds file descriptors
2. CPU informs the Network Interface(NIC) that new descriptors has been created to be used
3. Direct Access Memory (DMA) fetches descriptors
4. Packet arrives at the NIC
5. DMA writes the packet to the RX Ring buffer
6. NIC informs the driver which informs the CPU that new traffic is ready to be processed using Hardware Interrupt (top-half)
7. After the first Hardware interrupt, the Interrupt handler masks it, and instead the driver adopts the use of Software Interrupt (bottom-half) which is computationally cheaper for the CPU
8. The Software Interrupt invokes the New-API subsystem, which calls the NIC Driver's Polling function
9. CPU process the incoming packets
10. Some time after the Software Interrupt budget runs out and the CPU moves on to the next task

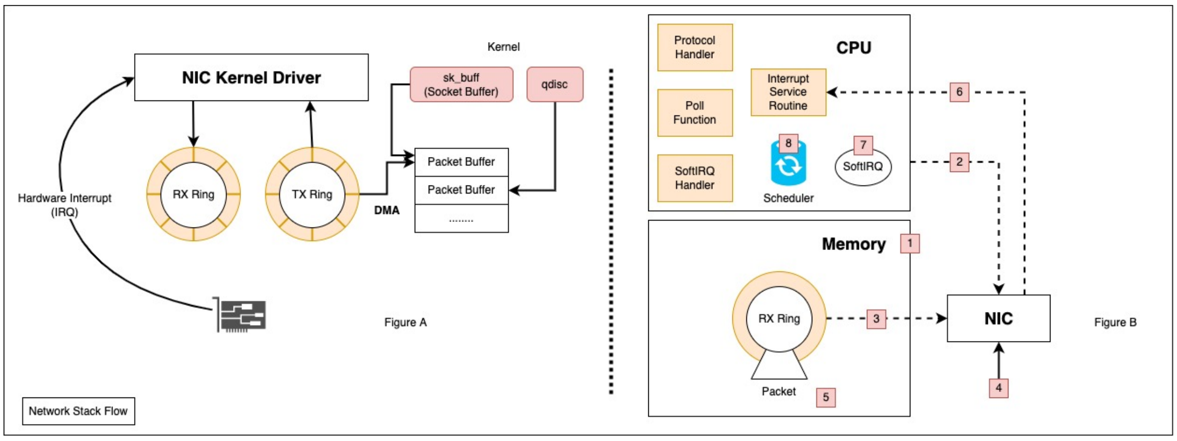

Figure 2.4: Data flow visualization in the Linux Network Stack

2.3.2 Drawbacks

While the architecture of the Linux network stack appears to be highly capable, numerous flaws become apparent in performance-critical environments, where understanding its limitations becomes necessary to design efficient and scalable networks. Key considerations include :

- **Performance Overhead** : Frequent transitions and multiple packet copies between kernel and user space introduce overhead which results in low-throughput workloads. The use of buffering and queuing at multiple levels worsen general performance and traditional interrupt-driven packet processing leads trivially to high latency as New-API or polling mitigations are not enough to grant acceptable results.
- **Configuration** : The stack's high flexibility requires deep expertise to be leveraged in order to optimize performance for high workloads and tune each layer reducing bottlenecks
- **Scalability** : Steps of the network stack, such as the kernel's TCP/IP processing, can become bottlenecks in multi-core systems being Single-threaded by design, limiting scalability.
- **Lack of Real-Time Guarantees** : Since the stack was not designed for real-time systems it cannot provide strict latency guarantees leading to non-deterministic behaviors that make it less suitable for applications like **industrial control systems**.
- **Limited Hardware Offload Support** : While modern NICs support offloading features, legacy systems may not have the same capabilities, resulting in limited performance. In addition, different hardware and drivers may implement offload features differently, resulting in an overall inconsistent experience.
- **Security Concerns** : The Linux kernel is home to the Network Stack, that allows any vulnerabilities to compromise the entire system leveraging also the large attack surface
- **Virtualization Overhead** : In virtualized environments, the network stack can introduce additional overhead, reducing performance compared to bare-metal systems.
- **Debugging and Troubleshooting** : Diagnosing network issues can be difficult due to the complexity of the stack and the interactions between different components. In addition, network stack monitoring and debugging tools may not provide sufficient visibility into all layers.

2.3.3 eBPF

One of the most critical aspects of this complex structure is the difficulty of extending or fixing the kernel's functionalities. The traditional way to achieve this is to implement kernel modules, which are pieces of code that can be loaded and unloaded as needed. While using them it is possible to modify almost any feature, modules' development can be quite challenging adding complexity and risk compared to user-space software development.

In a scenario where kernel module development seemed to be the only way forward, a complete revolution took place in 2011 when the first in-kernel just-in-time (JIT) compiler for the Berkley Packet Filter (BPF) was merged.

This event led to the creation of many extensions. A simple network packet filtering mechanism like BPF was transformed, conjugating its functionalities in the first non-networking tasks, first of all a way to filter system calls with a tool called seccomp-bpf. The big breakthrough came in March 2014, when the first extended Berkley Packet Filter (eBPF) interpreter appeared, capable of transforming classic BPF instructions into more powerful 64-bit ones.

From then on, it became clear that eBPF would become a viable alternative to kernel module development due to its ease of use and very little basic knowledge of the Linux kernel required to develop programs. eBPF allows running sandboxed program inside hooks points located at several places of the kernel architecture while safety is provided through an in-kernel verifier which performs static code analysis rejecting programs which crash, hang or otherwise interfere with the kernel negatively.

Doing so developers are able to add additional capabilities to the operating system at runtime as safety and execution efficiency are granted. This has led to a wave of eBPF-based projects covering a wide array of use cases, including next-generation networking, such as Express Data Path (XDP).

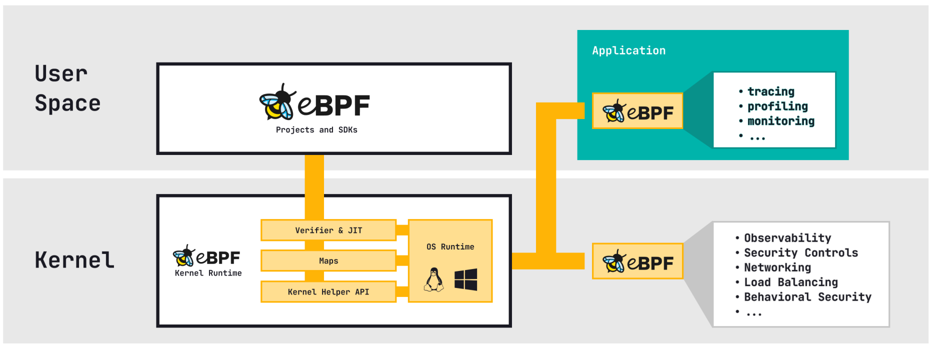

Figure 2.5: EBPF Structure visualization

2.3.4 XDP and AF_XDP

As mentioned earlier, critical flaws can be highlighted whenever the Linux network stack has to deal with large workloads in environments where high throughput and low latency are key factors. One of the solutions proposed to solve this problem was Express Data Path (XDP), a hook point located in the kernel architecture at the earliest stages of the communication data flow, inside the network device driver.

XDP grants high performance data receive and transmission letting the user bypass almost the whole networking stack let the supplied eBPF program decide the fate of packets. Located right after the interrupt processing and before any memory allocation this technology skips the heaviest steps of networking enabling commodity hardware to elaborate 26 million packets per core per second.

The program is allowed to do anything with the packet and once finished an action code determines the packets fate.

XDP Action Codes	
Action Code	Description
XDP_PASS	Let the packet continue through the network stack
XDP_DROP	Silently drop the packet
XDP_ABORTED	Drop the packet with trace point exception
XDP_TX	Bounce the packet back on the same NIC it arrived on
XDP_REDIRECT	Redirect the packet on another NIC or user space socket

The scope of the XDP program is limited by the kernel space environment, so a new address family has been introduced for even greater expressiveness without introducing packet identification ambiguity, AF_XDP. This innovation is nothing more than a raw socket optimized for high performance packet processing using zero-copy techniques between the kernel and user space. Leveraging this new communication paradigm, a direct channel is created from the NIC to a user application, bypassing the Linux network stack entirely and eliminating all of its associated problems.

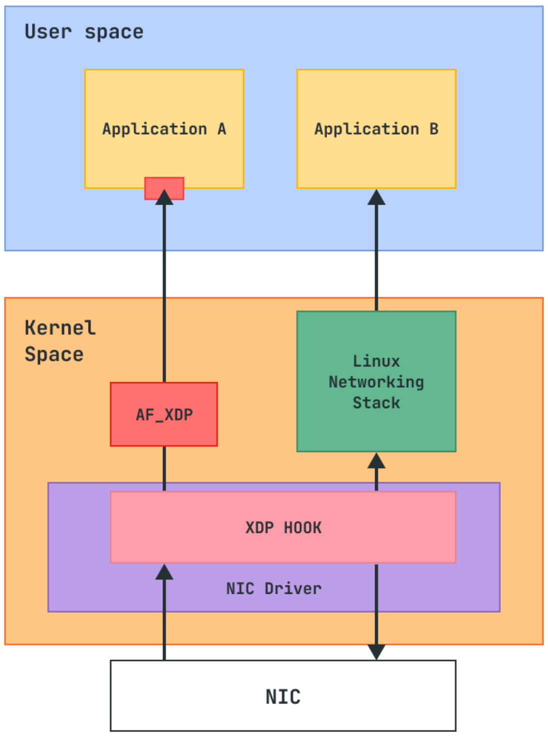

Figure 2.6: XDP and AFXDP visualization

2.3.4.1 AF_XDP Structure

Very few components contribute to the creation of this ingenious technology. A memory area, called a UMEM, is shared by both the application and the driver and it is divided into equally sized frames which can be owned by only one of the above at a time. The UMEM relies on two memory rings, Fill (FILL) and Completion (COMP), which are single consumer / single producer, meaning that only one entity can read / write at a time. They are used to transfer ownership of the UMEM frames between user space and kernel space.

The socket itself has two more rings, RX and TX, used for the actual exchange of the packets. The rings do not contain the actual packets but the addresses of the frames in the UMEM, as well as the length of the data for the last two rings.

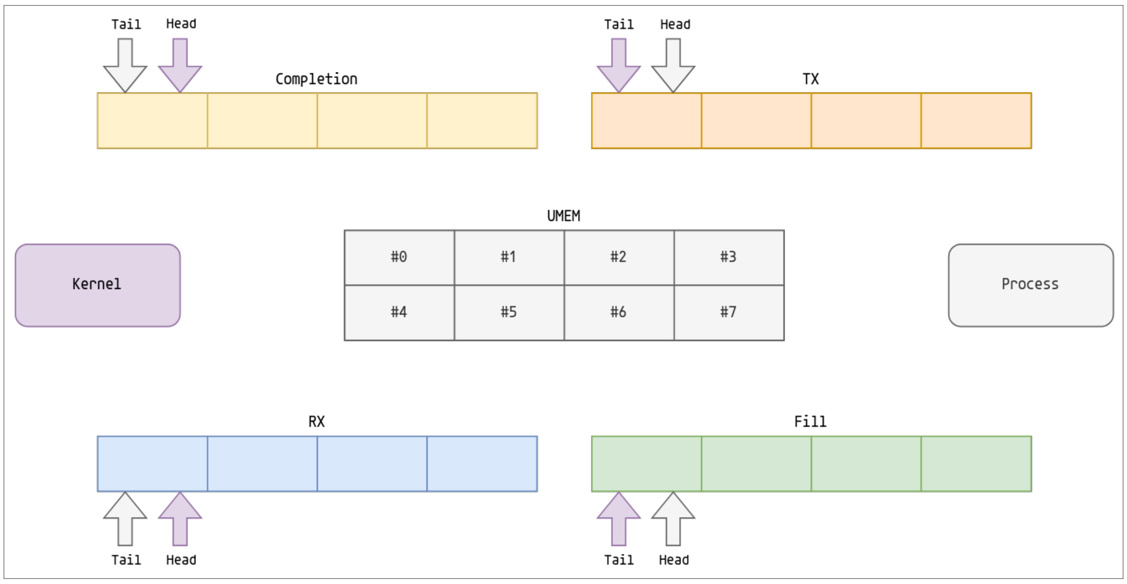

Figure 2.7: AF_XDP Socket Structure Visualization

2.3.5 AF_XDP Optimizations

As the Linux Networking Stack is bypassed, most of the processing is done either in user-space by the application or in the driver of the NIC. Therefore, in a simple application, most of the possible performance improvement will be obtained by modifying NIC hardware and driver settings as well as the AF_XDP socket parameters.

2.3.5.1 Hardware and Driver Settings

Many of the optimizations brought to the networking hardware and drivers are set with the goal to achieve higher rates, in terms of bits per second (bps) or packets per second (pps). While optimizing for throughput, it's trivial to assume that some components may induce higher latency especially at lower packet rates. A few tweaks have been discovered to lead to better performances:

- **Interrupt delaying** : Considering the fact that an interrupt signal is sent for each received packet, the traditional paradigm has to be changed when considering high packet rates. Two solutions have been considered: interrupt coalescing by New-API or by the NIC itself. The first hypothesis assumes that instead of sending an interrupt for each packet, the New-API triggers an interrupt to put the driver in poll mode. In this mode, the driver can be polled later by the kernel to process packets received since the last interrupt, saving CPU time. When operating on the NIC instead, coalescing delays trigger an interrupt when a packet is received within a certain time or frames, reducing interrupt overhead. Both mechanisms involve waiting before processing a packet, which may have negligible overhead at high packet rates, but can be significant at lower rates.

- **Energy Saving Mechanisms**: CPUs' integrated energy saving mechanisms, notably C-States, allow a CPU to enter a lower power state and disable some parts of itself when idle. This can lead to higher latency and jitter as the CPU may be in a energy saving mode when receiving a packet, especially at lower rates. Disabling these embedded strategies lead to better performances trivially.

2.3.5.2 Socket Options

The socket itself and the associated UMEM are also configurable. For the UMEM and the rings, the number of frames as well as their size can be configured. However, more options are available for the socket:

- **Zero Copy** : By default, sending data from user space requires at least one copy of the packet between an application-owned region of memory and a kernel-owned one in order to properly format the data before sending it to the network. However support, for sending and receiving packets without copying intermediate buffers to the NIC, can be added. Also, zero-copy in AF_XDP works on both the receiving and sending side, saving latency in both directions.

- **XDP_USE_NEED_WAKEUP** : By setting the XDP_USE_NEED_WAKEUP option, the application enables the need_wakeup flag on the TX and FILL ring. When it is false, enables the application to write and read from the rings directly without sending a syscall to the driver beforehand which highly decreases latency. When the flag is true, the application runs in the usual mode.

- **Busy Polling** : In order to signal applications that packets are available, the underlying network driver will send top-half interrupts/IRQs. This downgrades overall performance if it runs on the same core as the one sending the interrupts due to context switches. Applications that leverage busy polling has to wake up the driver so it can start processing packets both on the rx and tx sides. This enables all processing to be performed on one core and therefore prevent inefficient core-to-core cache transfers.

Chapter 3

Method

In this chapter, we present the chain of thought behind the implementation of our Layer-7 switch using AF_XDP technology and its application to a simulated maritime operational technology system. In particular, we will highlight the differences between a traditional Layer-2/Layer-3 switch and our implementation, with a focus on its smart-filtered data flow and the capabilities of this new paradigm.

3.1 From L2 to L7

When we talk about a "layer" in switch terminology, we are referring to the Open System Interconnection (OSI) layer the switch operates on. Today's implementations, applied to maritime OT systems, work on the the Data Link layer (L2) operating eventually on the Network Layer (L3). The lower one being responsible for MAC-Address based forwarding of data frames.

Traditionally network device interfaces (NICs), each one with a unique MAC address, are connected to the switch via physical Ethernet ports or fiber optic ports depending on speed requirements. When a NIC sends data to the switch its MAC address is inserted inside a very simple data structure, called the MAC address table, which contains basic routing information. If the table contains the packet's destination address, the packet is routed to the associated port otherwise it is sent to all ports except the sender's, creating a packet "flood".

Requiring no routing algorithm, and not needing IP addresses to forward data, Layer 2 switches are very fast, and cost less than routers. However, broadcast traffic is not controlled leading to network congestion while handling high workloads. Lastly, these kind of switch cannot pass data between different VLANs precluding network segmentation.

Faced with the practical and performance limitations of lower-layer switches, there has been a move toward Layer 3 implementations, especially in industrial network topologies. To ensure flexible application planning a host of Network Layer features are quickly became "must haves".

Layer 3 packet routing is performed by routers that use IP addresses instead of MAC addresses, making this new paradigm ideal for local area networks. Switches that use this strategy can connect different VLANs, provide more security features, and apply Quality of Service (QoS) controls for maximum efficiency.

Using an ARP table data structure which shows both MAC and IP addresses, switches will either forward the packet like a Layer 2 switch, or route it according to a routing protocols.

L2/L3 Features	
L2	L3
Sends data "frames" to destination MAC address	Routes data "packets" based on MAC or IP address
OSI Layer 2 (Data Link Layer)	OSI Layer 3 (Network Layer)
Cannot connect different VLANs	Able to connect different VLANs
One broadcast domain	Multiple broadcast domains
Communicates with local network	Can connect to outside (multiple) networks

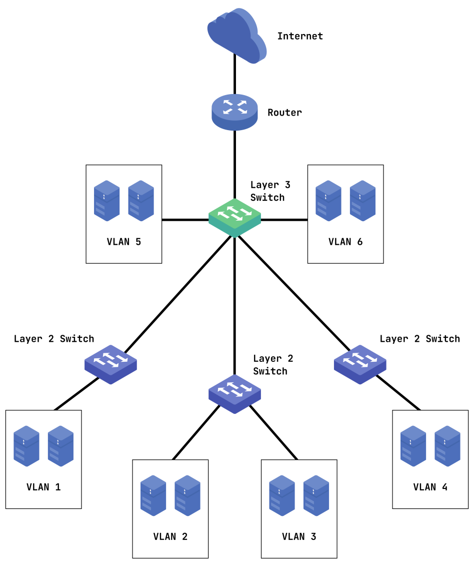

Figure 3.1: Visualization of architecture that leverages L2 and L3 switches

The two types of switches analyzed so far are capable of routing network traffic using different strategies, some more optimized than others. However, their functionality can be described as a "passive" way of managing data exchange, dictated by the switches' own implementation. There is no active process of handling network traffic, such as simply controlling that two components are allowed to "talk" to each other because the task is simply delegated to components' specific applications.

The solution to this problem resides in the Application Layer (L7) where expressive power over packets is almost limitless while being capable of low level routing. As said before this is an expression of the Software Defined Networking paradigm, where packet routing tasks are shifted from hardware to programmable software.

The baseline is the creation of a piece of software, written in any programming language, that creates an eBPF program with a XDP hook that attaches to the all the NICs of a designed architecture. Once the program is attached to the network interfaces an AF_XDP socket is created for each one and all the traffic sent and received by them is redirected to a user-space application. The application mentioned above is completely responsible for any action performed on the packets, being that dropping, filtering or simply redirecting them to the original destination.

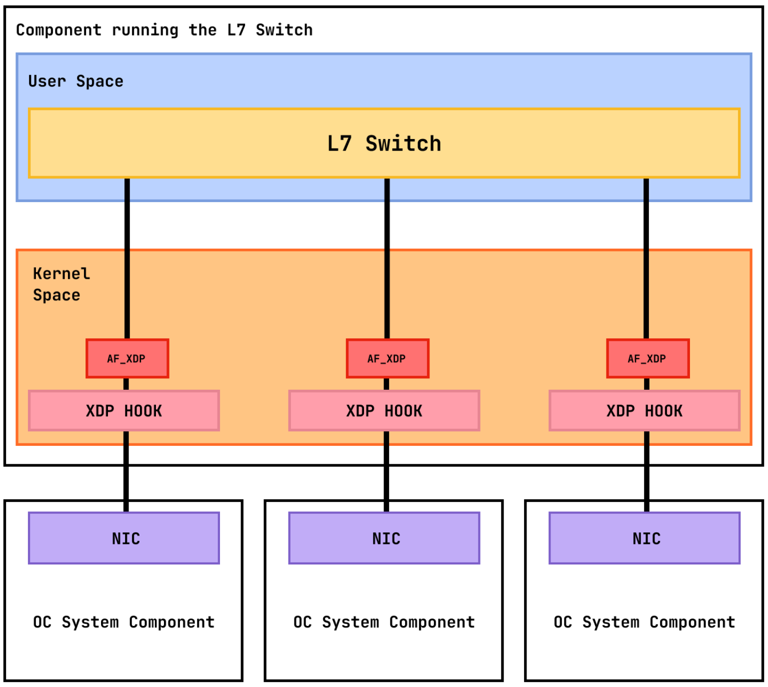

Figure 3.2: Visualization of architecture that leverages a L7 switch

3.2 L7 Networking

Once packets are redirected to user space by AF_XDP sockets, the data flow becomes extremely flexible. Hardware components such as Layer-2 and Layer-3 switches can be abstracted into data structures, as well as the network traffic itself. Following this principle, it's trivial to assume that fully personalized data-flows can be implemented according to predetermined sets of rules, which we'll call **policies**.

Since the ultimate goal is also to be completely hardware agnostic, it is necessary to preserve basic routing, previously physically handled, for packets that are not bound to network policies. Thereby two main flows can be identified, basic and filtered.

3.2.1 Basic Data Flow

In this scenario the goal is to reproduce the behaviour of a Layer-2 switch in user-space. Focusing on that a data structure that will take the physical hardware's place will be created, such as hash-map, as well as one additional other to store the traffic received by our application, that will be a vector. A struct called pollfd will be associated as well to the AF_XDP socket bound to the components' network interfaces. This will allow us to know which components have data to read and optimize iteration over components.

Once these data structures are set the algorithm can be break down as follows:

1. The pollfd structure bound to a component's socket is checked in order to know if there's new data available

2. The RX Ring Buffer of the socket is read and a network packet will be available for processing

3. Since we are not interested in handling data more than necessary in this implementation we just check if the packet's sender is already in the "hash-map-switch" and if not we add it creating an association between the source MAC address and the system's component

4. If the destination address is known we insert the packet, as long as the associated component, into the data structure that represents the system's traffic, otherwise we should apply a "flood" technique, which broadcasts the packets to all components to hit the correct one. To do so we add to the traffic multiple copies of the packet, one for each component

5. The packet is then added to the Fill Ring Buffer of the socket's UMEM, indicating complete receival, and new memory frames are added to the Fill Ring Buffer.

6. Iterating over the traffic vector packets are then added to the socket TX Ring Buffer and transmitted to the destination MAC address if known, broadcasted to all addresses if not

7. Once transmission is over packets are added to the Completion Ring Buffer of the socket's UMEM, indicating complete transmission

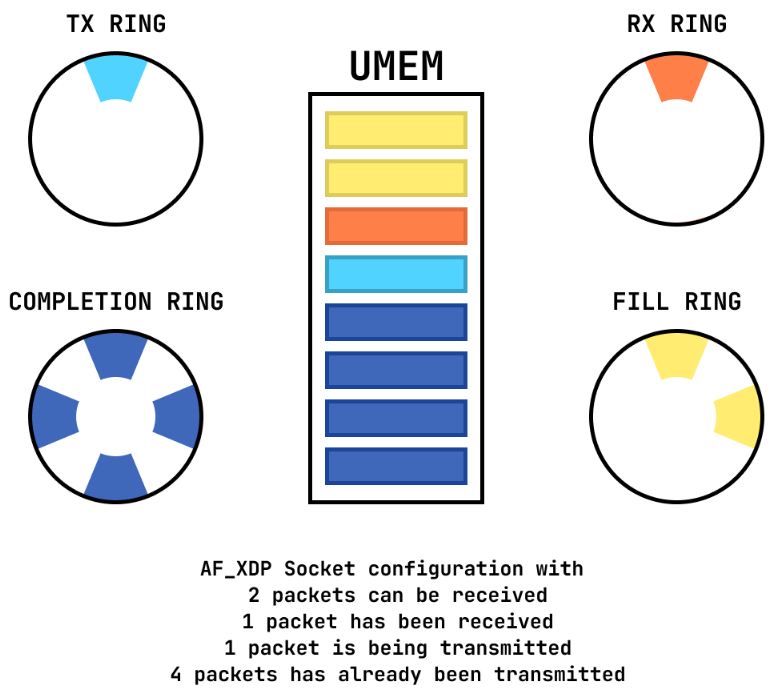
TX RING
UMEM
RX RING
COMPLETION RING
FILL RING
AF_XDP Socket configuration with
2 packets can be received
1 packet has been received
1 packet is being transmitted
4 packets has already been transmitted

Figure 3.3: Visualization of an AF_XDP socket

3.2.2 Filtered Data Flow

Once the basic flow is implemented, modifying it to provide filtering capabilities becomes trivial. Once the packet is in the RX ring buffer and fully received, almost any task can be performed before sending or dropping it, such as applying rule sets.

3.2.2.1 Policy Based Filtering

A policy-based network is easier to automate and therefore more responsive to changing needs. Many common tasks, such as adding devices and inserting new services can be easily accomplished.

Well-defined policies can benefit a network :

- Aligning the network with systems needs
- Providing consistent services across the entire infrastructure
- Bringing agility through greater automation
- Making performance dependable and verifiable

An even greater benefit to systems is the security provided by policies. Protection of sensitive data is enhanced by the granular definition of policies that give devices the least amount of access to resources. Violations can be quickly detected and mitigated, while zero-trust security measures reduce risk, contain threats, prevent malicious lateral movement, and help verify regulatory compliance. Key steps in the creation of sound policies can be identified in :

- **Identify** : Figure out the various components that form the target network. Having a complete view over the whole infrastructure is fundamental in order to define interactions among devices. Identifying partially devices and their security postures would lead to inconsistent policies
- **Visualize** : Understand how components communicate, which is a particularly hard task when working with legacy systems whose infrastructure may have been modified consistently during the years
- **Define** : Once gained enough architecture knowledge over the network that is being used, policies can be defined that will permit, deny, or modify data flows
- **Model** : Before applying policies a "dry run" must be actuated to determine what effects the policies will have on users, traffic, and performance.

3.2.2.2 Policy Application in Maritime Enviroments

Our implementation targets maritime OT systems whose internal communication is based on the NMEA-0183 and NMEA-2000 protocols, where information is sent via unencrypted UDP data frames containing ASCII payloads, with no restrictions on the flow direction of the communication.

The policy applied to the packets should be created according to the above principles, prioritizing the directional restriction rules that determine which NMEA messages can be sent and received by each component. Once the conformity of UDP frames with the policy is verified, the fate of the packets branches into two possible outcomes, which are

- **Packet Drop** : The packet is not added to the system's traffic data structure and is therefore not considered in the data transfer phase.
- **Packet Multicast Transmission** : Following the design of the NMEA protocol, the packet is sent to any component of the system that is allowed to receive it according to the policy rules. On this principle, the packet is multiplied and its destination addresses and checksums are processed to ensure its structure conforms to the recipient specifications.

3.3 Security Considerations

This paradigm results in absolutely strict network environments, especially for security concerns. The possibility of misbehavior and possible attacks against Layer 7 monitored systems is reduced to a minimum. With a granularity level that acts on packet payloads, the precision over rule application that can be achieved is the maximum possible.

When analyzing the vulnerabilities that affected the traditional NMEA 0183/2000 protocol communication flow, the increased level of security is immediately apparent when the traffic is handled by a L7 switch.

NMEA Protocols' security with L7 switch handled network	
Attack	Methodology
Denial of Service	Analyzing and identifying malicious traffic patterns provides protection against Dos and DDoS attacks. Policies that define the amount of traffic that can be sent over a given time span can also be used to enhance security. In addition, the vast amount of data exchanged at the application layer provides security analysts with a wealth of raw data for research
Spoofing & Packet Sniffing	This type of attack relies on the protocols' broadcasting principle, so this vulnerability is easily fixed by implementing routing rules that actually direct traffic only to a set of known and allowed devices.
Man in the Middle	This attack is the least likely of all the attacks that could be performed because it requires direct physical access to the system's network lowest components. A physical device should be placed exactly between the system's component and the switch. Assuming this, the attack would actually land against the network.

Scenarios where an attack would actually land, such as exploiting the man-in-the-middle vulnerability, are generally very difficult or even impossible with simple control of the application layer data. Each component is bound by functional and security constraints implemented in the logic of the component's control systems. Moving to the physical layer itself would be necessary to perform larger scale attacks, gaining control over analog drives, triggers, and emergency mechanisms.

Furthermore, layers of abstraction over internal networks can be used to enhance security through segmentation and indirection, helping to create even more complex systems that require a high level of technical insight to fully understand even basic data flow.

Utilizing such a layered architecture of L7 switches, different levels of data encryption can be achieved by applying specific policies, which would also enhance the privacy of internal data exchanges. In this way, the amount of unencrypted data available to each component of an OT system is reduced to a minimum.

Operating at the highest level of abstraction also enables statistical packet analysis and trace routing capabilities, providing a complete view of the internal network's metadata and data direction information, which is essential in emergency scenarios where identifying the root cause of a system failure is critical.

Chapter 4

Implementation

In this chapter, the focus will be on the implementation choices made for the realization of the Layer 7 switch. We will analyze the programming language, algorithms, data structures, and libraries used to optimize efficiency, modularity, and interaction with the underlying technologies discussed previously.

4.1 Enviroment setup

From the beginning, our switch was designed to be placed inside a maritime operational technology system, such as a ship, to monitor its internal network.

Since it was not possible to test the software against a physical infrastructure, the creation of a virtual twin seemed the most reasonable choice. In addition, the first step toward legacy compatibility was the ability to run the software on commodity hardware such as a personal computer, which has very limited resources compared to industrial system components.

4.1.1 Linux Network-Namespaces

Linux network-namespaces are a powerful feature that allows users to create isolated network environments within a single host, making them a valuable option for emulation, testing, and development, as they provide a controlled environment in which network configurations can be set up without affecting the host system or other network environments. Simulations such as the one we focused have core logic that can span multiple, separate environments, making it necessary to be network namespace, or container, aware. By creating a completely new network stack within a namespace for each system component, the containers can completely isolate their network environments from each other, perfectly simulating an industrial environment where each device is a separate entity with its own life cycle.

Taking advantage of the containerization capabilities of the technology, we automated the creation of a simple but complete OT system with a single Makefile.

```
CONTAINERS := 1 2 3 4 5 6 7 8

.PHONY: test-container-%
test-container-%:
	$(eval I := $(subst test-container-,,$@))
	sudo ip netns add test$(I)
	sudo ip netns exec test$(I) ip link set dev lo up
	sudo ip netns exec test$(I) ip link add eth0 type veth
	   peer name test$(I)
	sudo ip netns exec test$(I) ip link set dev test$(I) netns
	    1
	sudo ip netns exec test$(I) ip link set dev eth0 address
	   54:00:00:00:00:$(I)0
	sudo ip netns exec test$(I) ip addr add 10.42.0.$(I)0/24
	   dev eth0
	sudo ip netns exec test$(I) ip link set dev eth0 up
	sudo ip netns exec test$(I) ip neigh add 10.42.0.$(I)
	   lladdr 54:00:00:00:00:0$(I) nud permanent dev eth0
	sudo ip link set dev test$(I) up
	sudo ip link set dev test$(I) address 54:00:00:00:00:0$(I)
	sudo ip addr add 10.42.0.$(I)/32 dev test$(I)
	   noprefixroute
	sudo ip route add to 10.42.0.$(I)0/32 dev test$(I) src
	   10.42.0.$(I)
	sudo ethtool -K test$(I) tx off
	sudo ip netns exec test$(I) ethtool -K eth0 tx off

.PHONY: test-net
test-net: $(foreach container,$(CONTAINERS),test-container-$(
   container) )
```

This file alone is enough to configure a completely representative simulation of a possible infrastructure we might find on a ship. Eight network namespaces representing different physical components are created, and for each one a virtual Ethernet interface is also set up and paired with the loopback interface of the host machine that was previously brought up. As mentioned earlier, in keeping with namespace theory, each container is considered a separate device with its own MAC address, IP address, and network stack. To be consistent with packet routing and to have a simulation as close to a real context as possible, transmit checksum offloading has also been disabled to avoid unexpected behavior due to the abstraction layers of network virtualization.

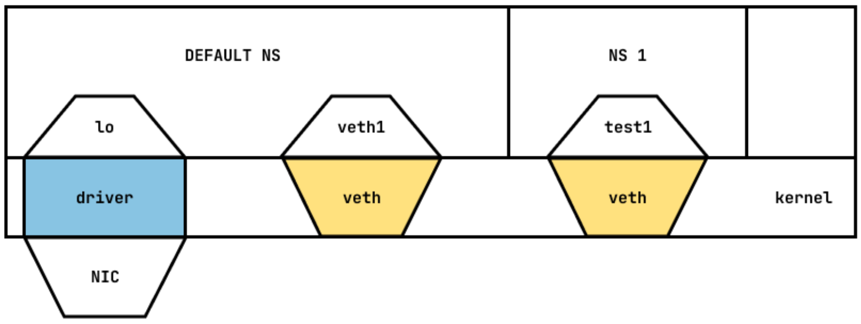

Figure 4.1: Visualization of Linux Network Namespaces

The only drawback of using network namespaces could be identified in the use abstraction layers of some kind that may give unrealistic results with slower or faster performance compared to the physical counterpart. But for the sake of research and study on the applicability of new networking strategies to legacy systems, our concerns for this part are mainly focused on the reproducibility of the network topology.

4.2 The Rust Programming Language

Upon delivery of packets to User Space, the developer assumes complete authority and accountability for the subsequent management and security of the data. This entails the obligation to ensure data is managed in a type and memory safe environment, with the preservation of information integrity being of paramount importance to the developer. Additionally, the software developed should be tailored to a performance-critical environment, where speed and efficiency are pivotal factors, as well as providing robust error and exception handling. The programming language selected for the switch development, Rust, is characterized by these features.

4.2.1 Language features

To elaborate on the aforementioned point, it should be noted that while all the features described are available in other programming languages, they are not all present simultaneously as in Rust. In addition, the following key aspects contribute to the selection of Rust as the optimal language for this implementation:

- **Algebraic Type System**, which enables the use of product and sum types that can be mixed with a strong pattern matching paradigm. This combination enables a fast and easy way of destructuring and managing of complex data aggregations, which are frequently encountered in simulations characterized by a high level of structural detail
- **Zero-cost abstractions**, which were instrumental in eliminating runtime overhead on high-level constructs while preserving the performance of the equivalent low-level counterpart. This facilitated the seamless integration of functional programming paradigms, while maintaining complete awareness of the performance-critical nature of the software.
- **Unsafe block declarations** This feature facilitates raw memory access, reminiscent of other low-level languages such as C, while retaining the safety features inherent to Rust. In case of a memory error, the process of identifying the underlying cause is expedited since low-level hardware access is permitted within a controlled environment defined by the developer

It is worth mentioning that for the same reasons as the one previously stated, the US Defense Advanced Research Projects Agency (DARPA) initiated the TRACTOR project. Consequently, this implementation choice can also be regarded as a method of standardization with leading technologies that are becoming prevalent in the software development scenario.

4.2.2 Project Layout

In order to enhance the flexibility of our software and ensure scalability as an intrinsic feature of the switch, we have adopted a modular project layout. This approach utilizes one of Rust's sophisticated features for managing large-scale projects, known as cargo workspaces. The adoption of this architectural paradigm has been shown to facilitate maintenance, construction, and testing of the project, while concurrently enabling the implementation of intelligent dependency management strategies.

The project is subdivided into a series of smaller, modular components, referred to as "crates." Each crate is assigned a particular responsibility, which can be versioned and published independently. This approach enhances reusability, management, and security. The switch we developed is composed by six crates :

- **ship_component** : Crate responsible for each ship component abstraction, managing its networking capabilities and policy application based network filtering
- **ship** : Crate responsible ship abstraction, seen as a collection of ship components, managing the traffic coming from all of them and enabling basic switch features such as packet redirection
- **packet_parser** : Crate responsible for network packet parsing and visualization
- **nmea** : Crate responsible for nmea messages recognition and parsing
- **policy_handler** : Crate responsible for policy parsing and marshaling
- **simulation** : Main crate responsible for generation of the ship and its components starting from parsed policy files.

The strength of this architectural choice is retained in its modularity. The addition of functionalities related to data visualization, security, or statistical analysis can be achieved with minimal effort by incorporating an additional crate, which is essentially a directory, into the project. This flexibility is particularly advantageous in environments characterized by constantly evolving requirements that must align with existing architectures. In such contexts, the project's layout choice stands out as a superior solution.

4.2.3 Components' Abstraction

The integration of the language's distinctive characteristics and rich type system facilitated the simulation of the ship and its components. The life cycle and the interactions within the various devices were collapsed within just two crates constituting the project's fundamental structural elements.

```rust
//ship.rs
pub struct Ship<'a> {
    components: Vec<ShipComponent<'a>>,
}

// ship_component.rs
pub struct ShipComponent<'a> {
    pub name: String,
    pub ifname: String,
    pub mac: String,
    pub ip: String,
    pub ifindex: libc::c_uint,
    pub bpf_manager: BPFRedirectManager,
    pub sock: XDPSocket<'a>,
    pub umem_allocator: UmemAllocator,
    pub poll_fd: libc::pollfd,
    pub sends: Vec<String>,
    pub receives: Vec<String>,
}
```

It is noteworthy that the ship itself and each component are implemented as a Struct product typ enabling the entire system simulation's creation to be as simple as initializing an array of variables and iterating over it, as it's possible to state looking at the program's main function

```
fn main() {
    let policy = PolicyHandler::new(
        String::from("./policies/policy_0.toml")
        );
    let mut ship_components: Vec<ShipComponent> = Vec::new();
    let policy = policy.get_policy();
    policy.iter().for_each(|component| {
        ship_components.push(ShipComponent::new(
            component.name.clone(),
            component.iface.clone(),
            component.mac.clone(),
            component.ip.clone(),
            component.sends.clone(),
            component.receives.clone(),
        ));
    });

    let mut ship = Ship::new(ship_components);
    ship.monitor_network();
}
```

Ship components contain essential values required for networking in addition to static data, such as the MAC and IP addresses. There exist also variables that oversee the AF_XDP socket bound to the component's NIC. This approach enables a high degree of flexibility by leveraging the **impl** pattern, which facilitates the definition of methods tailored to the component's specific functionality. A prime illustration of this methodology is evident in the manner in which each component administers the application of the network policy.

```rust
impl ShipComponent<'_> {
    fn apply_policy(&self, message: String) -> (bool, bool,
       String) {
        let mut nmea = Nmea::new();
        let message_ok = nmea.parse(message.clone());

        match message_ok {
            Ok(()) => {
                nmea.show();
                let prefix = format!("${}{}", nmea.
                   str_talker_id(), nmea.str_sentence_type());
                let is_allowed = self
                    .sends
                    .iter()
                    .any(|allowed_message| prefix == *
                       allowed_message);
                (is_allowed, true, prefix)
            }

            Err(_) => (true, false, String::from("NONMEA")),
        }
    }
}
```

The policy is applied to the packets sent by the component with a level of granularity specific to itself. In the present implementation, the UDP packet undergoes parsing and it is identified as a NMEA message if it conforms to the syntax rules of the protocol otherwise it is handled as a standard UDP message. Subsequent to this, a basic control mechanism oversees the messages that can be dispatched by the component, as delineated in the policy. This mechanism determines whether or not the packet under scrutiny should be appended to the ship's traffic.

The analysis performed is relatively elementary, as it does not examine additional packet information beyond the aforementioned parameters. Utilizing this paradigm, it is feasible to implement more stringent rules, such as those that verify packet weight, payload characteristics, or unusual attributes to the packets' mean. The capacity for data manipulation is boundless, enabling high-precision statistical analysis, the establishment of privilege systems among components, and more precise visualization of network status.

4.2.4 Policy Implementation

Another non-trivial feature to implement has been the policy handling. The policy software market is saturated with infrastructure and application-dependent programs, the adoption of which in legacy systems might imply changes to currently adopted standards. The objective was to engineer a policy system that is autonomous and environment-agnostic, thereby formalizing network communications without incurring computational overhead. The most rational choice was to select the most basic configuration language, allowing the switch software to integrate it into its process.

TOML files, renowned for their simplicity, readability, and flexibility, emerged as the preferred choice for this purpose. To that end, the decision was made to leverage these advantageous features, resulting in the creation of a components' configuration file that is both immediately comprehensible and retains sufficient expressiveness and modularity.

```toml
[policy]
c0 = { name = "girobussola", iface = "test1", mac = "54:00:00:00:00:10", ip = "10.42.0.10", sends = [
"$IIHDT",
], receives = [] }
c1 = { name = "ais", iface = "test2", mac = "54:00:00:00:00:20", ip = "10.42.0.20", sends = [
  "!AIVDM",
  "!AIVDO",
], receives = [] }
c2 = { name = "gps", iface = "test3", mac = "54:00:00:00:00:30", ip = "10.42.0.30", sends = [
  "$GPGGA",
  "$GPGLL",
  "$GPRMC",
], receives = [] }
```

```
c3 = { name = "ecoscandaglio", iface = "test4", mac = "
    54:00:00:00:00:40", ip = "10.42.0.40", sends = [
   "$IIDPT",
], receives = [] }
c4 = { name = "velocita", iface = "test5", mac = "
    54:00:00:00:00:50", ip = "10.42.0.50", sends = [
   "$IIVHW",
], receives = [] }
c5 = { name = "radar", iface = "test6", mac = "
    54:00:00:00:00:60", ip = "10.42.0.60", sends = [
   "$RATTM",
   "$RATLL",
   "$RAZDA",
], receives = [
   "$IIHDT",
   "!AIVDM",
   "$GPGGA",
   "$GPGLL",
   "$GPRMC",
   "$IIDPT",
   "$IIVHW",
] }
c6 = { name = "ecdis", iface = "test7", mac = "
    54:00:00:00:00:70", ip = "10.42.0.70", sends = [
], receives = [
   "$IIHDT",
   "!AIVDM",
   "!AIVDO",
   "$GPGGA",
   "$GPGLL",
   "$GPRMC",
   "$IIDPT",
   "$IIVHW",
   "$RATTM",
   "$RATLL",
   "$RAZDA",
] }
```

As demonstrated above, the TOML file is capable of defining not only policy rules through a list of permitted send and receive messages, but also the entire structure of the simulation. The incorporation of a component into our OT system is a straightforward process, requiring only the addition of a line of configuration with basic networking information. The marshaling procedure is equally streamlined, a feat facilitated by the "toml" crate in Rust, which enabled the conversion of the TOML file into an efficient hash map data structure with minimal coding.

```rust
#[derive(Debug, Deserialize)]
pub struct PolicyHandler {
    policy: HashMap<String, Component>,
}

#[derive(Debug, Deserialize, Clone)]
pub struct Component {
    pub name: String,
    pub iface: String,
    pub mac: String,
    pub ip: String,
    pub sends: Vec<String>,
    pub receives: Vec<String>,
}

impl PolicyHandler {
    pub fn new(policy_file_path: String) -> Self {
        let toml_content =
            fs::read_to_string(policy_file_path).expect("
               Failed to read config.toml");

        let policy_handler: PolicyHandler =
            toml::from_str(&toml_content).expect("Failed to
               parse config.toml");

        policy_handler
    }
}
```

4.2.5 Network Simulation

The components' networking capabilities, which leverage the AF_XDP technology, were developed using the *xdrippi* crate. This library provides low-level accessibility to the socket's main components through fairly simple APIs. The socket's initialization and setup is accomplished in a concise number of lines of code, as is the management of the ring buffers and UMEM. Furthermore it is necessary to highlight that the crate's implementation utilizes *glibc* primitives for socket creation and management, which may potentially lead to unsafe memory management. However, as previously mentioned, the use of unsafe blocks ensures the security and reliability of this implementation choice.

```rust
impl ShipComponent<'_> {
    pub fn new( name: String, ifname: String, mac: String,
        ip: String, sends: Vec<String>, receives: Vec<String>,
    ) -> Self {
        let ifindex = interface_name_to_index(ifname.as_str()).unwrap();

        let umem = Umem::new_2k(16384).unwrap();
        let umem = Arc::new(umem);
        let mut sock = XDPSocket::new(ifindex, 0, umem.clone(), 4096).unwrap();
        let mut bpf_manager = BPFRedirectManager::attach(ifindex);
        bpf_manager.add_redirect(0, sock.as_raw_fd());
        let umem_allocator = UmemAllocator::for_umem(umem.clone());

        while let Some(chunk_index) = umem_allocator.try_allocate() {
            if sock.fill_ring.can_produce() {
                sock.fill_ring
                    .produce_umem_offset(sock
                    .umem
                    .chunk_start_offset_for_index(chunk_index));
            } else {
                umem_allocator.release(chunk_index);
```

```
                    break;
                }
            }

            let poll_fd = libc::pollfd {
                fd: sock.as_raw_fd(),
                events: libc::POLLIN,
                revents: 0,
            };

            ShipComponent {name,ifname,mac,ip,ifindex,bpf_manager,
               sock,
                umem_allocator,poll_fd,sends,receives,}
        }
    }
```

The socket is created during the component's initialization, allocating UMEM memory frames for the Fill Buffer ring at startup. This ensures that the switch is ready to receive packets as soon as the device is added to the system. Rust's strong pattern matching capabilities were leveraged to facilitate the parsing of network packets, thereby ensuring the efficiency of the process while retaining the benefits of low-level checking through zero-cost abstractions. Constant monitoring of packet layers ensures precise identification of the data being processed, facilitating the creation of basic and filtered data flows with minimal effort.

```
    pub fn parse_traffic(&self) -> Result<String, i32> {
        if let Some(eth_packet) = EthernetPacket::new(self.
           packet) {
            match eth_packet.get_ethertype() {
                EtherTypes::Ipv4 => {
                    if let Some(ipv4packet) = Ipv4Packet::new(
                       eth_packet.payload()) {
                        return self.parse_protocol_ipv4(
                           ipv4packet);
                    } else {
                        return Err(-1);
                    }
                }

                _ => return Err(-2),
            }
        }

        Err(-1)
    }
```

Chapter 5

Results

In this chapter, an analysis is conducted of the results obtained from the execution of multiple tests on the software under investigation. Since tests over the implementation's correctness are trivial by design the focus is on network speed. The results are then compared with the performance of OT systems traditional networking techniques. Furthermore, an assessment is made of the advantages and disadvantages of the L7 switch.

5.1 Rust/AF_XDP filtered-unicast traffic

In order to ascertain the viability of our implementation as a substitute for current standards, it was necessary to establish a baseline, defined as the average speed at which UDP packets travel in a maritime network. Due to the unavailability of a real OT environment for testing our switch, all tests were conducted using the previously analyzed Network Namespace technology.

Given the information flow characteristics of naval systems, as outlined in the background section, the initial tests centered on the normal unfiltered data exchange. As the data payload values were not a factor in this testing phase and the generation of a plausible amount of data was necessary, the iperf3 tool was selected. This software enabled easily the recreation of a representative environment, in which data flows at approximately 1 Gbit/s with a default UDP packet size of 1460 KB between two components.

Initially, it was necessary to ascertain whether the implementation of system emulation utilizing network namespaces would result in the introduction of overhead, which could potentially compromise the reliability of the tests from the outset. Consequently, experiments were executed on the data exchange between emulated components, leveraging the Linux Network Stack with base parameters. Subsequent to the initial testing, additional trials were executed in a second iteration, employing maximum values to assess the limits of data exchange.

Iperf tool results - Linux Network Stack Unicast		
Role	Bitrate [Mbit/s]	Datagram size [B]
SENDER	1000	1460
RECEIVER	1000	1460
SENDER	4800	8972
RECEIVER	4800	8972

As evidenced by the iperf tool's output, the namespace abstraction did not result in any additional overhead maintaining a bitrate of 1 Gbit/s between the sender and receiver. Having established a baseline that should emulate a real-life scenario, we proceeded to test a filtered-unicast information flow, leveraging our switch implementation with the same iperf parameters.

Iperf tool results - Rust/AF_XDP Unicast		
Role	Bitrate [Mbit/s]	Datagram size [B]
SENDER	1000	1460
RECEIVER	1000	1460
SENDER	4800	8972
RECEIVER	4800	8972

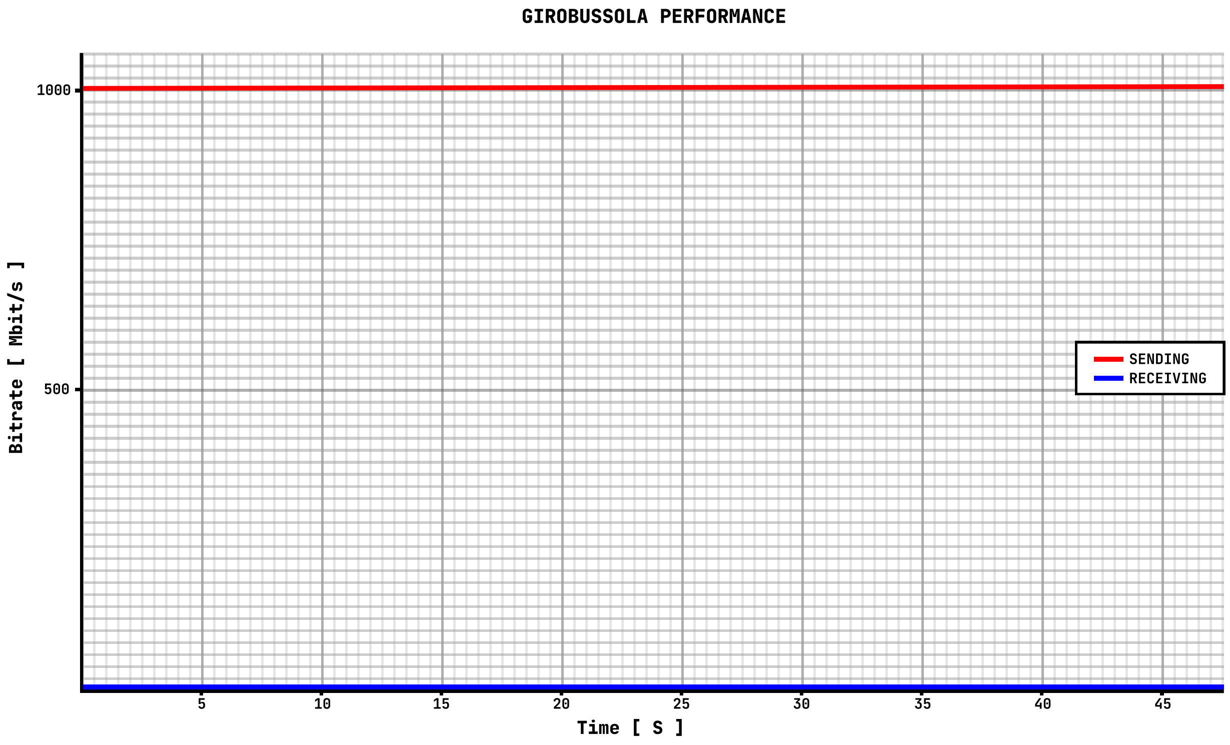

Figure 5.1: Bitrate over time - Sender AF_XDP- Unicast flow - 1460B Dataframe size

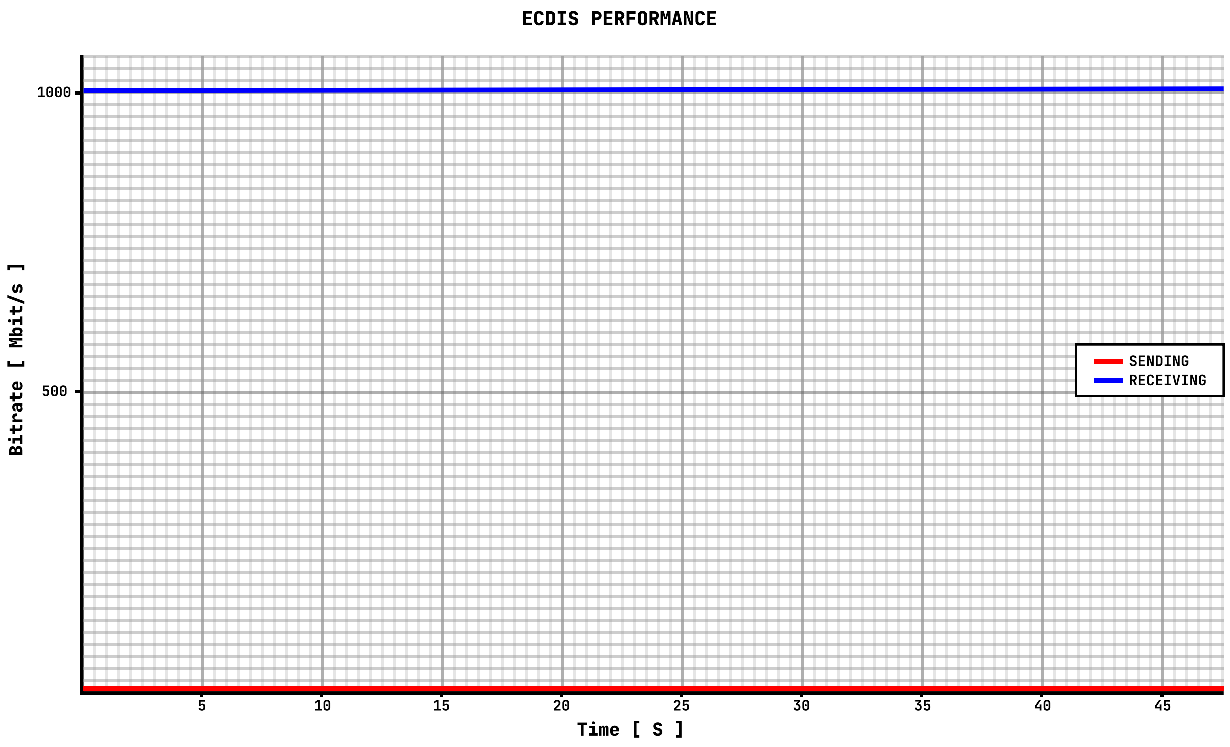

Figure 5.2: Bitrate over time - Receiver Rust/AF_XDP- Unicast flow - 1460B Dataframe size

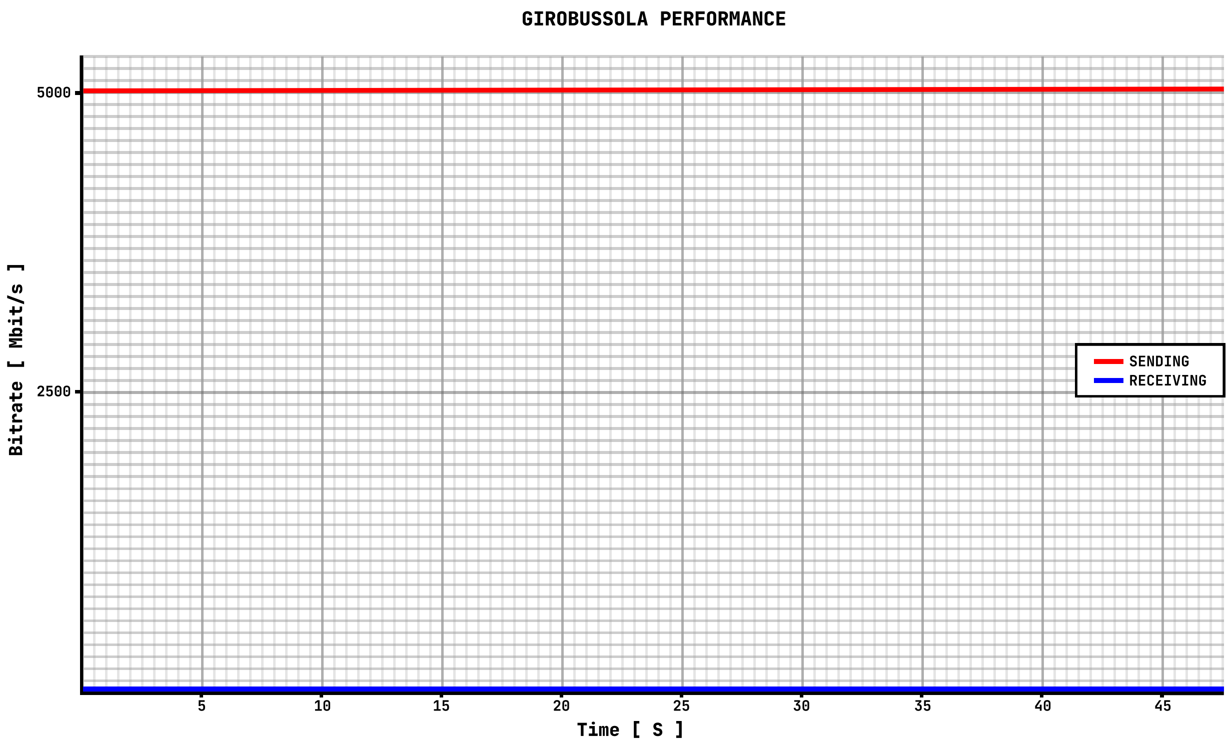

Figure 5.3: Bitrate over time - Sender Rust/AF_XDP- Unicast flow - 8972B Dataframe size

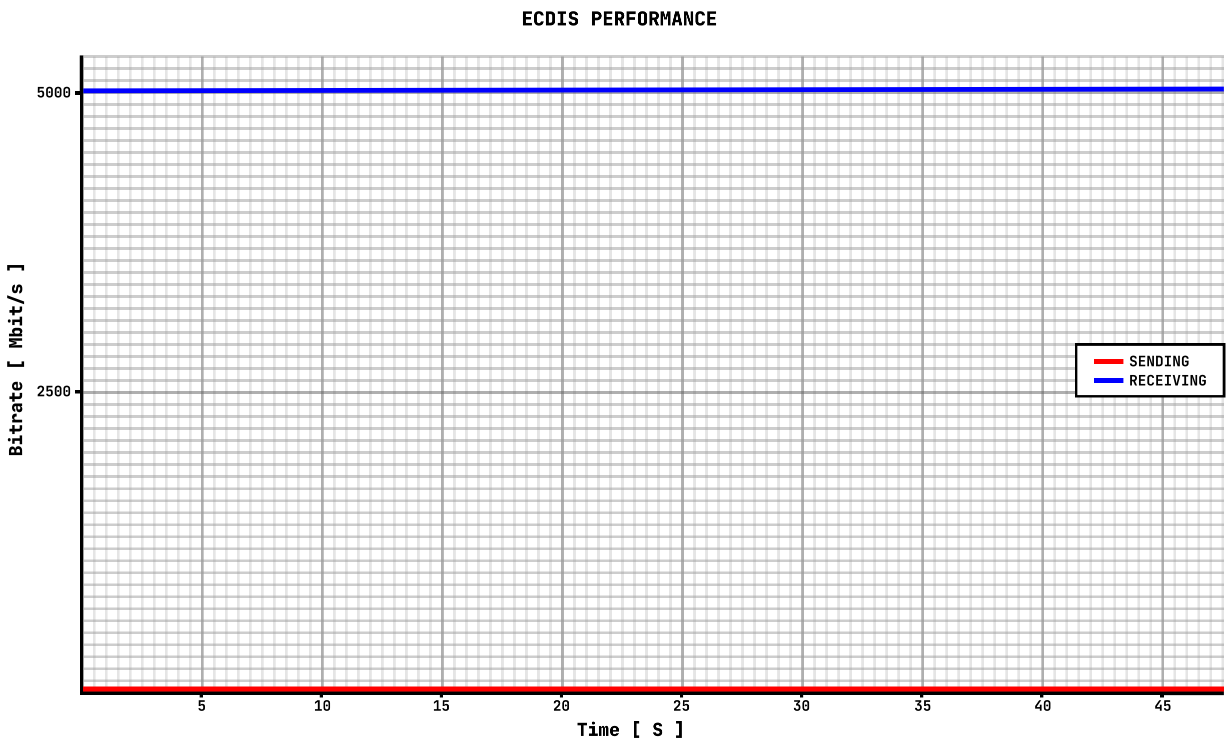

Figure 5.4: Bitrate over time - Receiver AF_XDP- Unicast flow - 8972B Dataframe size

An examination of the iperf results and the graphs obtained from analyzing the network traffic within the switch on a twenty seconds simulation revealed that our implementation did not modify the data exchange speed over time. This observation indicates that the Linux Network stack baseline was maintained without introducing additional overhead in the filtered-unicast scenario with a base traffic bitrate of 1Gbit/s with a dataframe size of 1460B and a maximum traffic bitrate of 4.8Gbit/s with a dataframe size of 8972B.

5.2 Rust/AF_XDP filtered-multicast traffic

In pursuit of the implementation's objective to engineer a tool that would demonstrate seamless integration with an existing network while maintaining complete architecture-agnosticism, the multicast traffic feature was implemented through a process of packet multiplication operating at the application layer, thereby circumventing reliance on addresses. This strategy aligns with the principles of Software Defined Networking, fostering modularity within the network itself by defining system components' relationships dynamically through a policy file, eliminating the need for modifications to the network architecture.

In accordance with the established protocol, packets are multicasted to the policy-defined components subsequent to the validation of their payload. To this end, a dedicated tool was developed to enable the transmission of custom dataframes that adhere to the protocol's structure, thereby circumventing the utilization of randomly generated packets by iperf.To emulate a realistic scenario, three components were selected: Girobussola, ECDIS, and Radar. According to NMEA protocol's rules whenever the "girobussola" sends a message, containing the talker ID value "II" and the sentence type value "HDT", this should be multicasted to the ECDIS and Radar components. In this scenario, the traffic was emulated using the same parameters that had been set previously in iPerf. The results obtained were as follows:

Iperf tool results - Rust/AF_XDP Multicast		
Role	Bitrate [Mbit/s]	Datagram size [B]
SENDER_Girobussola	1120	1460
RECEIVER_ECDIS	1120	1460
RECEIVER_Radar	1120	1460
SENDER_Girobussola	5100	8972
RECEIVER_ECDIS	800	8972
RECEIVER_Radar	5100	8972

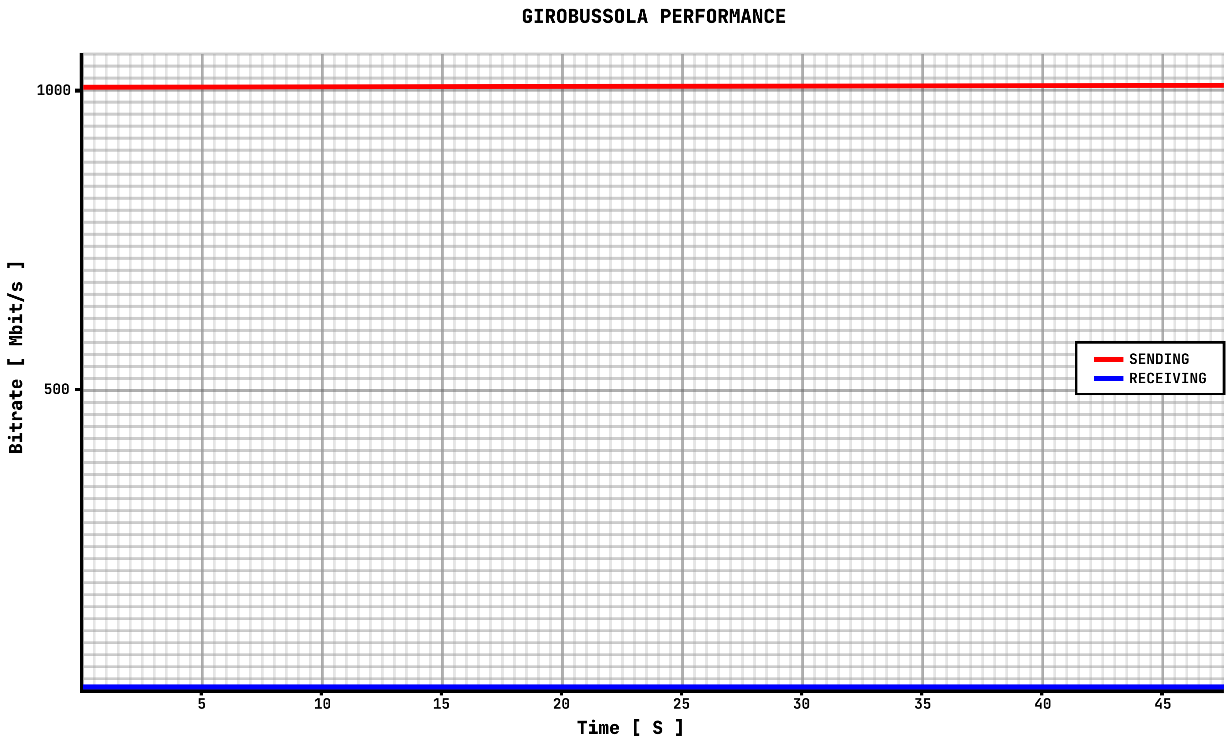

Figure 5.5: Bitrate over time - Sender_Girobussola AF_XDP - Multicast flow - 1460B Dataframe size

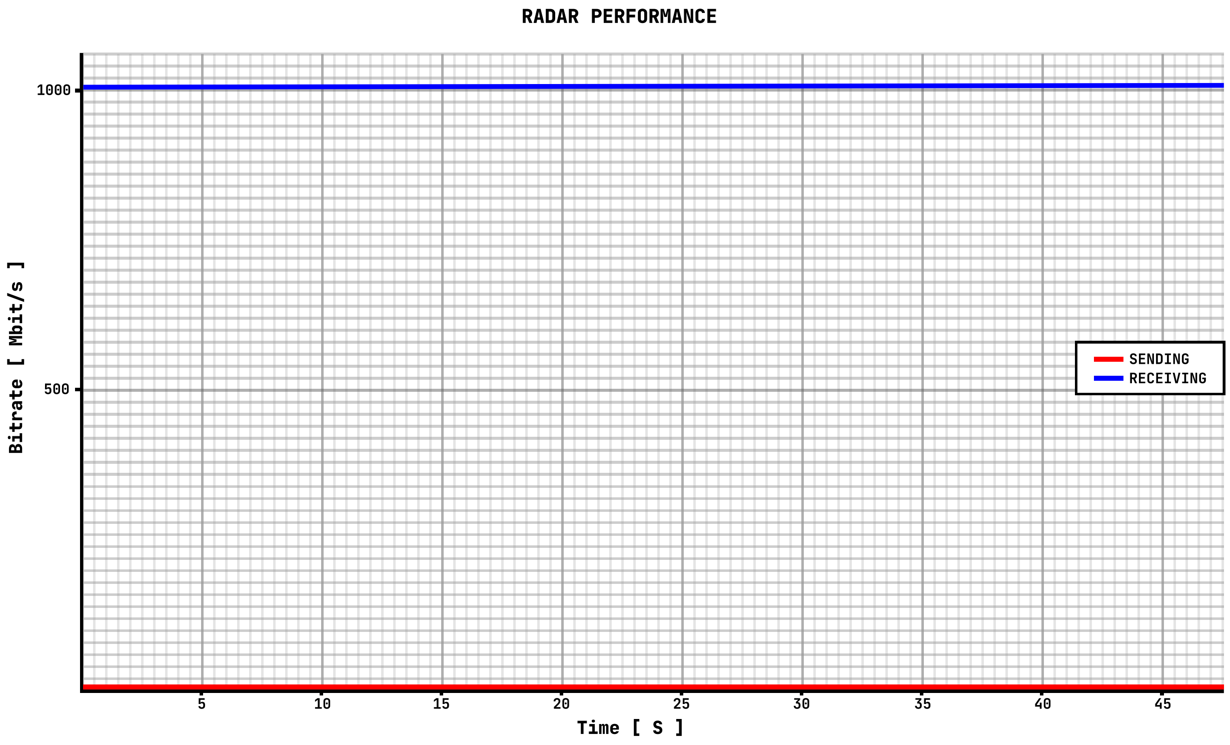

Figure 5.6: Bitrate over time - Receiver_Radar Rust/AF_XDP - Multicast flow - 1460B Dataframe size

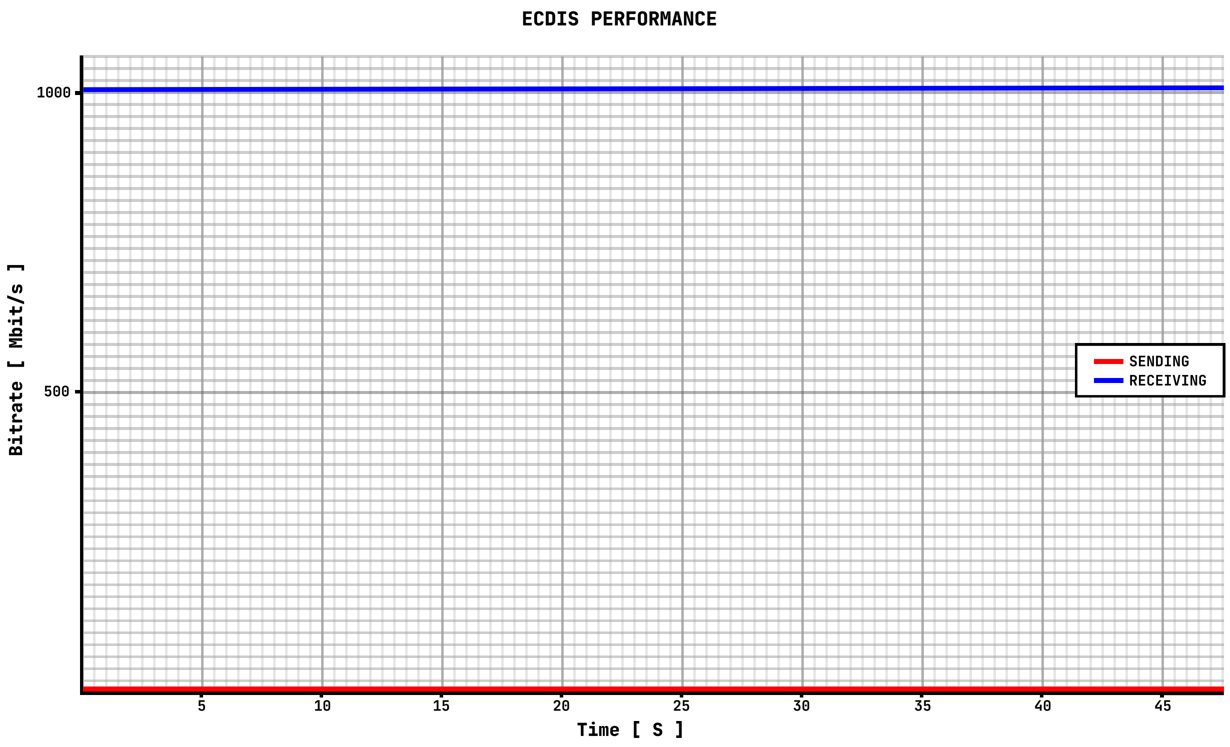

Figure 5.7: Bitrate over time - Receiver_ECDIS Rust/AF_XDP - Multicast flow - 1460B Dataframe size

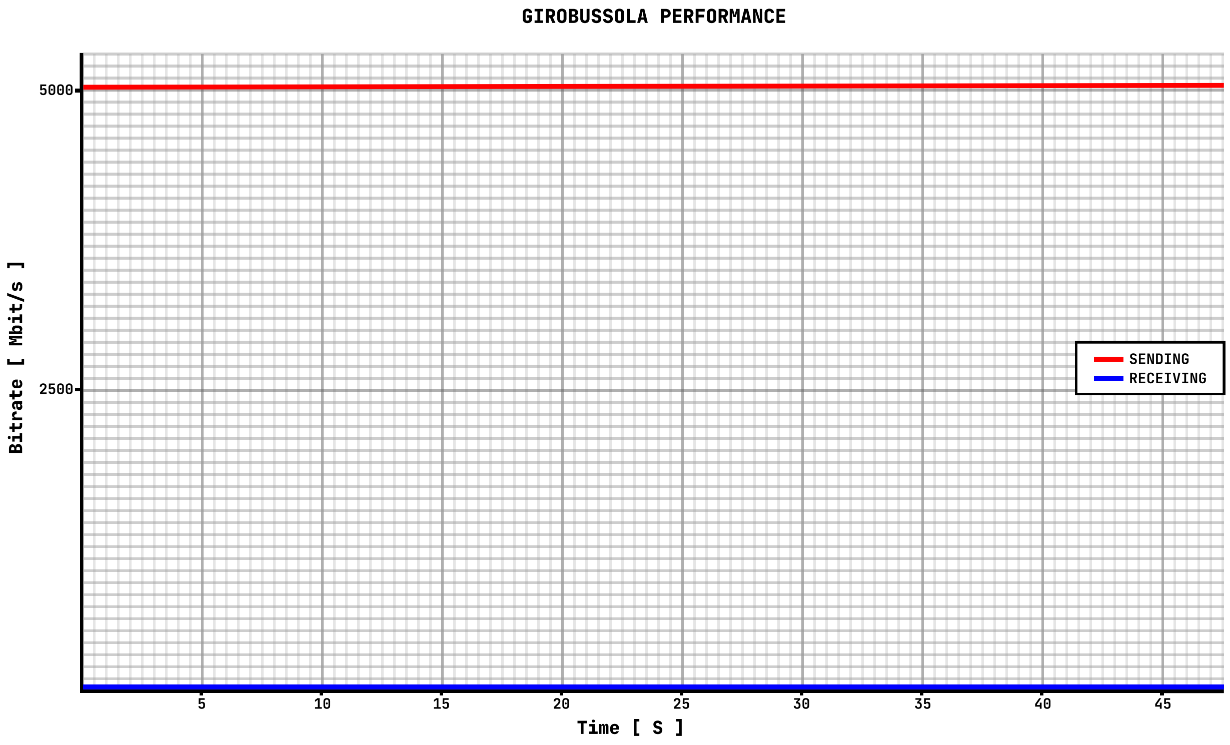

Figure 5.8: Bitrate over time - Sender_Girobussola AF_XDP - Multicast flow - 8972B Dataframe size

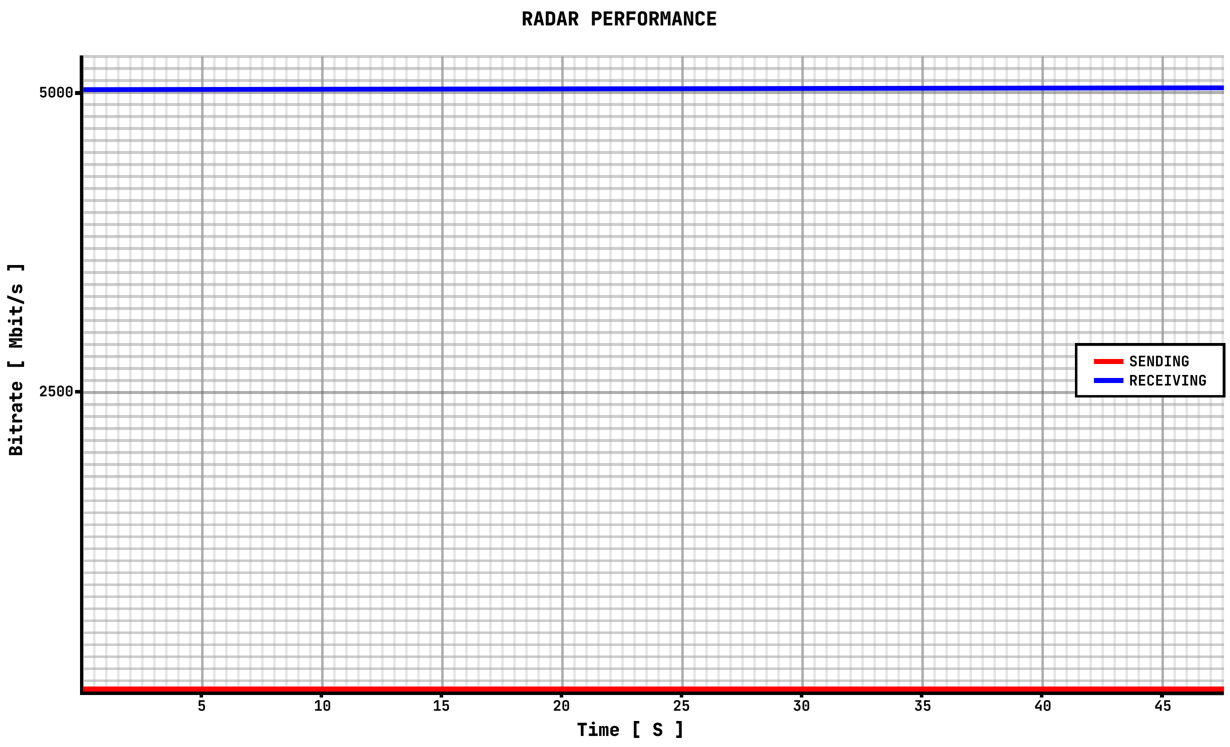

Figure 5.9: Bitrate over time - Receiver_Radar Rust/AF_XDP - Multicast flow - 8972B Dataframe size

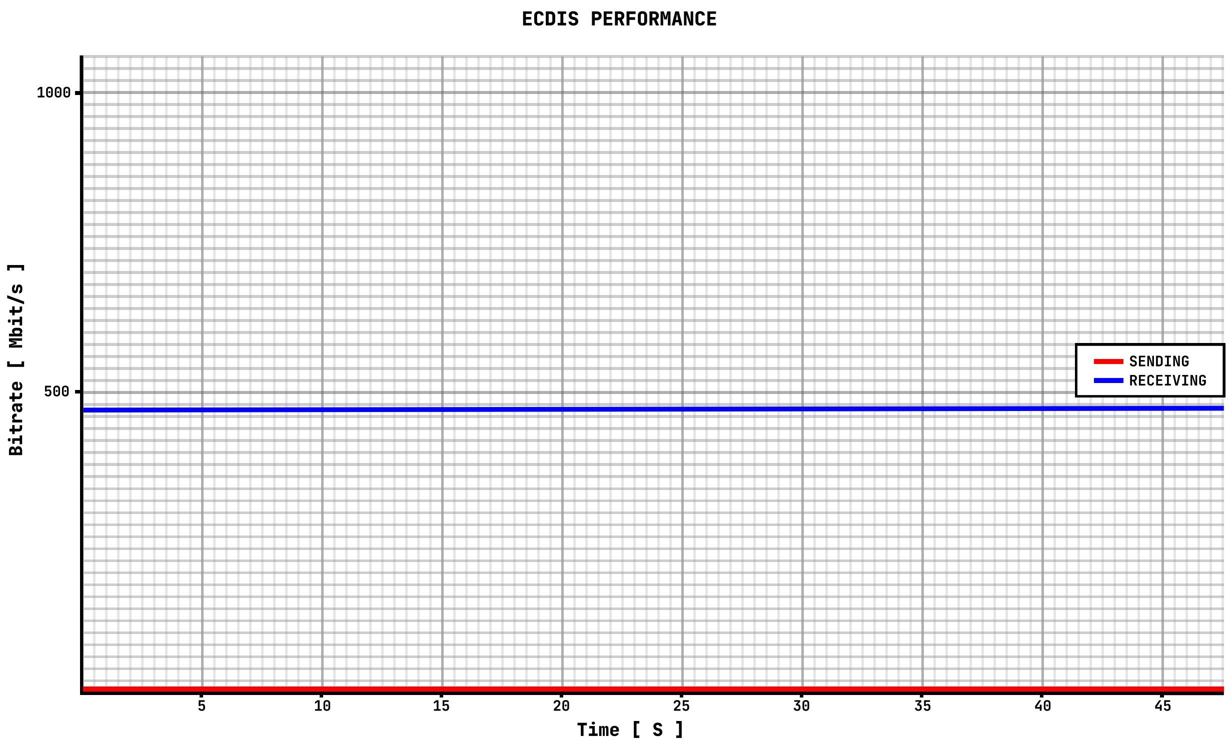

Figure 5.10: Bitrate over time - Receiver_ECDIS Rust/AF_XDP - Multicast flow - 8972B Dataframe size

The graphs above illustrate the switch performances while working with different datagram sizes, 1460B and 8972B. The results coming from the ECDIS receiver component working with heavier packets are of particular interest. While performances reached working with lighter datagrams demonstrate an identical trend to the ones in the unicast traffic simulation, an anomaly is observed for more demanding workloads. This phenomenon can be explained by examining our implementation of multicast functionality. Rather than relying on multicast addresses, packets must be copied completely and then modified to be multicasted. Doing so the performance remains unaltered when datagrams are relatively small while adding computational load to the switch results in a decline in performance as datagram size increases.

The ECDIS component received packets at a lower bitrate because it was not the primary target of the sender. Given that the primary target was the radar component, there was no need to copy and modify packets for transmission to it. Conversely, the ECDIS component was a secondary target, and the multicast process operations resulted in a decline in performance when handling more substantial packets. This phenomenon is made evident with maximum datagram lengths. This represents the sole constraint in our implementation, wherein we prioritize architectural flexibility over performance for non-standard network workloads.

Chapter 6

Related Works

6.1 OT Security overview

In recent years, there has been a concerted effort to modernize and fortify critical maritime OT infrastructures. This initiative has given rise to the development of cybersecurity campaigns and directives, which have in turn identified opportunities to enhance the management of control systems and network operations. These are comprised of a set of rules and procedures that aim to raise awareness of potential cyber risks as well as to define the behaviors that are to be adopted, identifying the human factor as the most significant vulnerability, as stated in a serie of articles and online resources such as [Kec+22], [Sin23], [HSLB24], [KB19].

Although solutions that do not necessitate additional software or hardware appear to be the prevailing preference with a notable emphasis on personnel training the necessity of a technical response to this threat has been underscored as well: [Mel+22].

The modern industry is shaped by innovative concepts and technologies that aimed towards interconnectedness facilitated by machine-to-machine and machine-to-control system communications. Keeping this in mind critical challenges in securing OT systems were born needing solutions that prioritized availability, integrity and confidenciality.

Moreover, the integration of information technology (IT) and operational technology (OT) infrastructures within contemporary architectural frameworks has unveiled a plethora of previously unacknowledged vulnerabilities and threats. This integration has led to an augmentation of complexity and interconnectedness, thereby expanding the attack surface significantly. In light of these developments, the necessity for novel risk assessment methodologies has become apparent to ensure the proper security of infrastructures.

Within the domain of OT systems, two primary methods have emerged: qualitative and quantitative. Qualitative assessment prioritizes individual risks based on the probability of their occurrence, whereas quantitative assessment analyzes risk numerically by assigning it a numerical value. Presently, the preponderance of maritime physical risk assessments is rooted in probability statistics.

The development of a qualitative risk assessment framework for cyber risks in maritime environments poses significant challenges. The scarcity of data, attributable to the limitations of reporting abilities and the novelty of this emerging risk, contributes to the volatility of maritime cyber data, making reliable probability estimation challenging.

In addition to the aforementioned points, the measurement of cyber risk has been conducted on numerous occasions in a variety of sectors and their systems. However, there has been a paucity of such studies in the maritime sector, which has been estimated to be approximately two decades behind cyber-security trends. The unique systems, protocols, and the movement across physical and cyber spaces mean that traditional methods of risk assessment cannot be easily applied without modifications to existing infrastructures. The unique blend of these factors create a distinct maritime risk landscape that can result in outcomes such as loss of finance, loss of life, and environmental damage.

6.2 Hardware Solutions

In consideration of the interconnection among components, the principal vulnerability in this approach is identified in the communication channel. Consequently, it is hypothesized that an attacker can gain access to the network and identify vulnerabilities in the process controlled by a system component.

The proposed solution to this problem is the implementation of intrusion detection systems of variable complexity, which necessitate the integration of additional hardware components within the existing infrastructure whose purpose would be to analyze network traffic seeking potential attack patterns. The work published in the article [CPP19] will be taken as example for the following considerations on the reasons that lead us to believe that this is a non optimal solution, following also the topics found in [BT19]. The introduction of new hardware components invariably gives rise to a multitude of issues, largely due to their physical characteristics.

6.2.1 Hardware Security and Hardware Trust

The emergence of hardware security concerns is attributable to its inherent vulnerability to attacks across various levels. Additionally, the absence of robust hardware support for software and system security contributes to these concerns. Conversely, issues pertaining to hardware trust arise from the involvement of untrusted entities in the life cycle of a hardware component, including entities such as untrusted IP or computer-aided design tool vendors, as well as untrusted design, fabrication, test, or distribution facilities.

These entities have the potential to compromise the trustworthiness of a hardware component or system and could potentially cause deviations from the intended functional behavior, performance, or reliability of the hardware. Trust issues can also lead to other incidents, such as poor parametric behavior, or degraded reliability, or safety issues.

Attack Vectors rising from each step of hardware manifacture	
Life Cycle Step	Attack Vector
IP Vendor	H/W Trojan Insertion, Hidden backdoor placement
SOC Design House	IP Piracy, Trojan in Design
Foundry	Implant Trojan, Overproduction Cloning
Deployment	Side Channel Attacks, Reverse Engineering, Scan Based attacks, IC Counterfeiting

6.2.2 Attack surface exposure

The term "attack surface" is used to denote the totality of all potential security risk exposures. This concept can be further elucidated by defining it as the aggregate of all known, unknown, and potential vulnerabilities, along with the corresponding controls, across all hardware, software, and network components. The objective of developing countermeasures is often to minimize the attack surface. With respect to hardware security, three main attack surfaces are as follows

- **Chip Level Attacks** : Chips are susceptible to targeted attacks, including reverse engineering, cloning, malicious insertion, side-channel attacks, and piracy. In the event that an attacker is able to create a copy that closely resembles the original, counterfeit or fake chips can be sold as authentic units. Additionally, the presence of Trojan-infected chips within the supply chain can pose a threat of unauthorized access or malfunction. Side-channel attacks, in particular, present a significant threat, as they can be used to extract secret information stored within the chip.
- **PCB Level Attacks** : Printed circuit boards (PCBs) are frequently targeted by attackers due to their relative ease of reverse engineering and tampering when compared to the complexity of entire system components. The design information of most modern PCBs can be extracted through relatively simple optical inspection

and efficient signal processing. The primary objectives of these attacks are twofold: first, to reverse engineer the PCB, and second, to obtain the schematic of the board to redesign it and create fake units. Attackers may also physically tamper with a PCB to render it susceptible to leakage of sensitive information or to circumvent digital rights management (DRM) protections.

- **System-Level Attacks** : Complex attacks involving the interaction of hardware and software components can be mounted on the system. By directly targeting the most vulnerable components within a system, such as the Design-for-Test (DFT) infrastructure at PCB level and memory modules, attackers may be able to compromise the system's security, thereby gaining unauthorized control and access to sensitive data.

Furthermore, vulnerabilities related to weakness in hardware architecture, implementation, or design/test process must be taken into consideration. These vulnerabilities can be classified as either functional or nonfunctional, and their manifestation is contingent on the system's nature and its intended usage scenarios. With respect to hardware, the most prevalent causes of vulnerabilities can be identified in the following bugs :

- **Functional Bug** : The majority of vulnerabilities are attributable to functional bugs and substandard design and testing practices. These vulnerabilities encompass a range of issues, including the implementation of weak cryptographic hardware and the inadequate protection of assets within a system. Attackers may identify these vulnerabilities by meticulously analyzing the functionality of a system under various input conditions to identify any anomalous behavior.

- **Side-Channel Bug** : These bugs are indicative of implementation-level issues that result in the leakage of critical information stored within a hardware component via various forms of side-channels. Attackers may identify these vulnerabilities by analyzing the side-channel signals during the operation of a hardware component.

- **Test/Debug infrastructure**: The preponderance of hardware systems offers a satisfactory degree of testability and debuggability, thereby empowering designers and test engineers to validate the operational integrity of these systems. These systems also facilitate the study of internal operations and processes running on the hardware, which are essential for debugging the hardware. However, these infrastructures are not immune to exploitation by malicious actors. In the wrong hands, test/debug features can be used for extracting sensitive information or gaining unauthorized control of a system.

- **Access control or information-flow issues** : In certain instances, a system may exhibit a lack of distinction between authorized and unauthorized users. This vulnerability can potentially enable an attacker to gain access to sensitive assets and functionalities that may be exploited for malicious purposes. Furthermore, an intelligent adversary can monitor the information flow during system operation to decipher security-critical information, such as the control flow of a program and the memory address of a protected region from a hardware perspective.

6.2.3 Considerations

The aforementioned discourse expounds on the predominant rationales elucidating why the incorporation of hardware components does not constitute a paradigm of ideal solutions to our implementation from an empirical standpoint. A comprehensive explanation can be found in a seminal article from the Journal of Computational and Cognitive Engineering, [Joh+22], which demonstrates that the optimal availability of a system can be achieved only if it is periodically completely repaired and supporting units have been invoked.

This finding holds particular relevance to the proposed solution by the Rome University, which if implemented would require, in a system with a greater number of components, a more complex and potentially more maintenance-intensive solution. In contrast, our implementation relies exclusively on a software-based solution, circumventing the previously identified defects. Furthermore, it can be seamlessly integrated into existing devices without requiring any hardware modifications, a key advantage over cited approaches.

6.3 Protocol Solutions

In light of the maritime communication systems current state, alternative solutions have been proposed in the domain of information exchange standards. These proposals have identified the underlying issues as stemming from the utilization of outdated systems that employ data exchange methodologies devoid of any data analysis nor cyber/security considerations. There is a growing recognition of the need for a novel smart communication system, as evidenced by numerous sources that have proposed designs emphasizing network architecture, air interface, and radio spectrum, with the objective of enabling e-Navigation. This encompasses a range of autonomous and semi-autonomous navigation systems that utilize 5G, LTE, satellite, and radio communications, superseding the prevailing standards outlined in the literature, as cited in [Xia+20] [Vhf].

6.3.1 Industry Concerns

The majority of vessels and naval infrastructures currently sailing around the globe are old transports with legacy devices providing connectivity and managing internal state and network capabilities. This circumstance can be identified with ease as the primary impediment to the realization of e-Navigation, which can be formalized as a series of challenges that must be confronted. These challenges include, but are not limited to:

- **Ubiquitous Connectivity and Service Continuity** : The creation of a new maritime system is predicated on the imperative to ensure ubiquitous connectivity between vessels and shore on a global scale, especially over open oceans. Furthermore, the deployment of these services has historically been limited to a campus-style model, resulting in inconsistent and, at times, non-existent cross-region continuity of maritime service. Consequently a global network would be crucial for providing consistent and continuous maritime services worldwide, especially in remote and inaccessible regions such as the polar regions
- **Traffic Nonuniformity** : Notwithstanding its global reach, maritime traffic is characterized by significant disparities in distribution. Concentrated flows of maritime traffic are a common occurrence in ports, along coastal areas, and along waterways. For instance, coastal shipping accounts for more than 50 percent of the total ship transport to and from the ports of the coastal countries, where the cargo vessels primarily follow the routes that are set close to the shore wherever possible. In contrast, traffic on the high seas is primarily attributable to intercontinental transportation or deep sea shipping, and is comparatively sparse in density. The maritime traffic management network must therefore be equipped with an efficient solution to cope with this type of traffic pattern
- **Service Centricity** : In contrast to conventional mobile networks, which are designed around a specific network architecture, maritime MTC systems must facilitate the efficient provisioning, discovery, and execution of diverse maritime application and service components distributed across the network. This is essential for realizing the full potential of transitioning to maritime IoT. Consequently, a new maritime system should not be confined to a one-time design and deployment; it is expected to offer amorphous services that adapt to a wide variety of maritime specific needs, and match changing demands. Hence, both network configuration and communication resources must be made flexible and adaptive to the specific service offered
- **Device Heterogeneity** : To accommodate a wide array of maritime IoT applications, maritime systems must be capable of supporting a diverse range of devices, spanning from the lower-end category, which encompasses devices with constrained functionality to the higher-end category, which includes devices with comprehensive functionality. This extensive heterogeneity in communication capability, encompassing hardware, power supplies, interoperability, and protocols, necessitates the implementation of an efficient system capable of accommodating this variation under a unified framework.

- **Simplicity and Reliability** : In contrast to the majority of terrestrial systems, simplicity has historically served as the overarching criterion for the design of maritime communication systems. A simplified system is less expensive to manufacture and maintain, and it is generally more robust and reliable in complex marine environments. Indeed, reliability is of paramount importance to maritime systems, while cost is also a significant factor that cannot be disregarded. Consequently, the selection and development of technology must prioritize cost-effectiveness and the provision of complimentary loyalty services.

- **Capacity and Scalability** : Constrained by the paucity of communication resources, enhancing efficiency emerges as the pivotal strategy to optimize the system's capacity. It is imperative to optimize physical and higher layers to facilitate more spectrally-efficient communications. Concurrently, the system must be scalable, accommodating future growth in response to augmented demand for capacity and escalating bandwidth requirements.

- **Interoperability** : In the context of a maritime system, it is imperative for vessels and maritime devices to receive services from other systems or networks. The integration and cooperative utilization of data within and across network boundaries should be guaranteed. Furthermore, it should ensure the timely and seamless portability of information across the full spectrum of maritime services

- **Radio Spectrum Internationality** : The radio spectrum is indisputably the most critical component for any wireless communication system. This is particularly true for maritime systems due to its global coverage nature. To successfully deploy the system globally and ensure proper functionality, it is imperative that an international frequency band be made available and established with suitable standards and regulations. To achieve this objective, it is essential that international standards and regulatory bodies, in collaboration with the global maritime community, address the technical and regulatory challenges.

6.3.2 IoT Security

The maritime industry's reliability and sustainability are of paramount importance to the economic success of nations worldwide. Vessels are critical transportation systems that facilitate the global movement of goods. Ensuring safety and security within the maritime industry is contingent upon the implementation of robust cybersecurity measures. The advent of fully or semi-autonomous systems, which demand minimal human intervention, necessitates a comprehensive evaluation of the inherent security vulnerabilities and risks associated with the adopted technology. As articulated in [Far+23].

It is imperative to acknowledge that critical attacks that can be perpetrated against maritime OT systems that leverage IoT communication systems can assume various forms, depending on the attack vector. Examples of such vectors include:

- **Tug boat cyberattacks** : The communication systems utilized on tug boats to interface with other vessels or shore-based facilities are susceptible to cyberattacks, which can result in a loss of communication or even a complete system takeover. These cyberattacks can manifest in various forms, such as jamming, spoofing, or interception of communication signals. Jamming, for instance, can disrupt communication signals, impeding the tug boat's ability to communicate with other vessels or shore-based facilities. Spoofing involves the transmission of false signals to the tug boat's communication systems, tricking them into executing unauthorized instructions, which can result in erroneous actions or compromised safety. Finally, interception can result in the exposure of sensitive or confidential information to unauthorized third parties.
- **Harbour manoeuvres incidents** :The cybersecurity risks associated with the entire apparatus of sensors, such as laser sensors, ultrasound systems, and radar, are relatively low. This is due to the fact that they are standalone devices that are not connected to any network or system. Nevertheless, concerns regarding data interception or tampering during transmission persist, particularly in scenarios involving wireless transmission. Furthermore microwave transponders, portable pilot devices, and automatic identification systems facilitate communication between vessels and shore-based monitoring units, introducing a series of cyber vulnerabilities that can be critical for harbor manoeuvres, resulting in unauthorized access, connection jamming, and data tampering.
- **Berthing Aid System** : Unauthorized access to the BAS by an employee can result in data breaches and system damage through display of incorrect information. This could occur due to an employee accessing the system without the proper authorization or credentials. Access to the BAS should be restricted to authorized personnel only based on the principle of least privilege, since employee withoutout proper knowledge could potentially delete or corrupt data stored in the system, leading to data loss or system failure

6.3.3 Considerations

The implementation of a solution that addresses the aforementioned concerns regarding compatibility and security would necessitate a substantial allocation of resources and time, as previously indicated. Conversely, our proposed solution can be seamlessly integrated into existing infrastructures using today's standards, with a minimal time requirement for testing. The design choices that have been implemented result in a high degree of modularity, which in turn allows for extreme adaptability to legacy systems and non-optimal infrastructures.

6.4 Application solutions

In addressing the imperative to bolster cybersecurity measures within the OT landscape, alternative solutions have been proposed. These solutions prioritize enhancing network security through the utilization of user space applications. This approach bears a certain resemblance to our implementation previously outlined, albeit with certain caveats that introduce a degree of uncertainty in its reliability.

System hardening entails the implementation of security controls that serve to reduce the attack surface and enhance the security posture of the systems. This involves the implementation of security controls such as firewalls, intrusion detection and prevention systems, access controls, and other security measures that are intended to limit the potential for successful cyberattacks. This approach, when implemented, can effectively mitigate the risk of successful cyberattacks and enhance the overall security of critical infrastructure. However, from an applicative point of view, the hardening of legacy systems poses significant obstacles due to their intrinsic nature.

This path is fraught with challenges related mainly to compatibility , management and security, such as

- **Dependecy Risk** : In the context of conventional IT systems, the repercussions are commensurate with specific human activities, and the user has the capacity to ameliorate the impact through manual interventions. Conversely, in the context of OT, enterprises must exercise caution in the management of dependencies on IT components, with the objective of averting the potential for adverse impact on physical processes in instances where human intervention is not feasible. Given the inherent limitations of mitigation in OT, the most effective approach is often to exercise control over dependencies to avert potential impacts.
- **Data Management** : The data produced by OT devices can be voluminous, diverse in content, time sensitive for consumption, and geographically distributed. Conversely, the majority of IT systems exhibit a certain degree of tolerance for time delays, are constrained in size and content, and are reliably connected to company networks, facilitating accessibility to IT staff for data management and support. Conversely, OT systems necessitate a determination by the company regarding the

integration of the data into the enterprise data landscape or the potential for the data created by the OT system to remain self-contained and local.

- **Application Security** : The integration of OT systems with IT has been shown to create additional vulnerability targets, caused by an enlargement of the attack surface to the user space specific interactions, potentially impacting not just people and data, but also physical processes.

The implementation's primary function is to operate at the kernel level, leveraging the user space as a data management platform without requiring user interaction. This approach has enabled the creation of a tool that is free from dependencies and can be programmed to process data in real-time if needed. Furthermore user interaction is not needed and the potential for user-interaction specific threats is completely eradicated.

Chapter 7

Conclusion and Future Work

In this study, we presented a simulation of a Layer 7 Switch that was based on the AF_XDP technology applied to a maritime OT system. This simulation was successful in achieving the initial objectives that were established. In fact, our software is entirely agnostic to the underlying software and hardware. The only software dependency of our software is the Linux kernel, and it does not require additional physical components. Additionally, our implementation can be seamlessly integrated into existing systems without altering standard communication protocols or introducing new ones preserving the transparency feature. Finally, it does not alter the normal communication flow and is able to handle, filter, and process NMEA traffic without adding any performance overhead, as evidenced by the obtained results.

It is possible that enhancements may be implemented, including software refinements. The main one being the implementation of a crate/library that is independent from the C libbpf library. This would ensure complete system fault-proofing, a characteristic facilitated by the Rust programming language features. Prior to contemplating additional any other enhancement excluding the aforementioned, it is essential that the software undergoes testing on real industrial systems. This evaluation may identify other software features that could prove beneficial.

Bibliography

[BT19] Swarup Bhunia and Mark Tehranipoor. *Hardware Security: A Hands-on Learning Approach*. Morgan Kaufmann Publishers, 2019.

[CPP19] Riccardo Colelli, Stefano Panzieri, and Federica Pascucci. “Securing Connection Between IT and OT: The Fog Intrusion Detection System Prospective”. In: *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)* 5 (2019), pp. 19–28. DOI: https://doi.org/10.1109/METROI4.2019.8792884.

[KB19] N. Kala and Mahesh Balakrishnan. “Cyber Preparedness in Maritime Industry”. In: *International Journal of Scientific and Technical Advancements* 5 (2019), pp. 19–28.

[Xia+20] Tingting Xia et al. “Maritime Internet of Things: Challenges and Solutions”. In: *IEEE Wireless Communications* (2020), pp. 188–196. DOI: https://doi.org/10.1109/MWC.001.1900322.

[KTT21] Erik Fretheim Ky Tran Sid Keene and Michail Tsikerdekis. “Marine Network Protocols and Security Risks”. In: *Journal of Cybersecurity and Privacy* (2021). DOI: https://doi.org/10.3390/jcp1020013.

[Joh+22] Yakubu Mandafiya John et al. “Reliability Analysis of Multi-Hardware–Software System with Failure Interaction”. In: *Journal of Computational and Cognitive Engineering* 2 (2022), pp. 38–46. DOI: https://doi.org/10.47852/bonviewJCCE2202216.

[Kec+22] Evripidis P. Kechagias et al. “Digital Transformation of the Maritime Industry: A Cybersecurity Systemic Approach”. In: *International Journal of Critical Infrastructure Protection* 37.100526 (2022), pp. 135–148. DOI: https://doi.org/10.1016/j.ijcip.2022.100526.

[Mel+22] Oleksiy Melnyk et al. “Integrated Ship Cybersecurity Management as a Part of Maritime Safety and Security System”. In: *International Journal of Computer Science and Network Security* 22 (2022), pp. 1721–1739. DOI: https://doi.org/10.22937/IJCSNS.2022.22.3.18.

[Far+23] Mohamed Ben Farah et al. "Cyber Incident Scenarios in the Maritime Industry: Risk Assessment and Mitigation Strategies". In: *International Conference on Cyber Security and Resilience (CSR)* (2023). DOI: https://doi.org/10.1109/CSR57506.2023.10224972.

[Sin23] Navneet Singh. *From Ports to Protocols: Securing Maritime with Palo Alto Networks*. 2023. URL: https://www.paloaltonetworks.com/blog/network-security/netsec-maritime-ot-security/.

[Vhf] *A New Link Adaptation Technique for Very High Frequency Data Exchange System in Future Maritime Communication*. Tech. rep. MDPI, 2024.

[HSLB24] Marie Haugli-Sandvik, Mass Soldal Lund, and Frøy Birte Bjørneseth. "Maritime Decision-Makers and Cyber Security: Deck Officers' Perception of Cyber Risks Towards IT and OT Systems". In: *International Journal of Information Security* 23 (2024), pp. 1721–1739. DOI: https://doi.org/10.1007/s10207-023-00810-y.