CyberShip-IoT: A Dynamic and Adaptive SDN-Based Security Policy Enforcement Framework for Ships

Sahay, Rishikesh; Meng, Weizhi; Sepúlveda Estay, Daniel Alberto; Jensen, Christian D.; Barfod, Michael Bruhn

Published in:
Future Generation Computer Systems

Link to article, DOI:
10.1016/j.future.2019.05.049

Publication date:
2019

Document Version
Peer reviewed version

CyberShip-IoT: A Dynamic and Adaptive SDN-based Security Policy Enforcement framework for Ships

Weizhi Meng[a], Christian D. Jensen[a], D.A.Sepulveda[b], Michael Bruhn Barfod[b], Rishikesh Sahay[a]

[a]*Dept. of Applied mathematics & Computer Science,Technical University of Denmark, DK-2800 Kgs., Lyngby, Denmark*
[b]*Dept. of Technology, Management and Economics, Technical University of Denmark, DK-2800 Kgs., Lyngby, Denmark*

Abstract

With the wide adoption of Information and Communication Technology (ICT) in the marine environment, ship systems are increasingly similar to other networked computing systems. The integration of positioning systems with navigational and propulsion control systems and the increasing reliance on Supervisory Control And Data Acquisition (SCADA) systems for monitoring the ship's performance makes modern ships vulnerable to a wide range of cyber security issues. Moreover, frequent or permanent onshore connection makes the ship's communication network a potential target for cyber-criminals. Such attacks can incapacitate the vessel, i.e., through a ransomware attack, or greatly degrade the performance of the ship systems, i.e., causing delays in the propagation of control messages between critical components within the ship. Furthermore, crew members and marine engineers are challenged with the task of configuring security policies for networked devices , using low-level device specific syntax, which is an error prone and time consuming process. In addition to this, crew members must also be familiar with the specific syntax for low-level network management task, which exacerbates the problem. The emergence of Software-Defined Networking (SDN) helps reduce the com-

Email addresses: weme@dtu.dk (Weizhi Meng), cdje@dtu.dk (Christian D. Jensen), dasep@dtu.dk (D.A.Sepulveda), mbba@dtu.dk (Michael Bruhn Barfod), risa@dtu.dk (Rishikesh Sahay)

plexity of the network management tasks and we believe that a similar approach may be used to address the larger problem. We therefore propose the CyberShip-IoT framework to provide a network level defense for the communication network component of ship systems. CyberShip-IoT offers a high-level policy language and a translation mechanism for automated policy enforcement in the ship's communication network. The modular design of the framework provides flexibility to deploy detection mechanism according to their requirements. To evaluate the feasibility and effectiveness of this framework, we develop a prototype for a scenario involving the communication network of a typical ship. The experimental results demonstrate that our framework can effectively translate high-level security policies into OpenFlow rules of the switches without incurring much latency, ultimately leading to efficient attack mitigation and reduced collateral damage.

Keywords: SDN, Shipping system, Policy language

---

1. Introduction

The Internet of Things (IoT) generally refers to connecting devices over the Internet. IoT has many applications in a varied number of fields ranging from smart cities and the power grid, to the shipping industry. It has given new opportunities to the ICT to leverage the interconnection between the physical and cyber layer to effectively manage the network. Development in the IoT along with the ICT have also revolutionized the ship technology. It has brought a new era of cyber ships. All the ships' components such as global navigation satellite system (GNSS), Automatic Identification Systems (AIS), Electronic Chart Display Systems (ECDIS) are integrated with the cyber systems. With the use of ICT technologies, the control systems on the ship have evolved similar to industrial control systems. They have the controller that manages the different IoT devices on the ship such as sensors, actuators and different bridge devices (e.g., AIS, GNSS, ECDIS). These components of a ship are interconnected with the on-board communication network to control and manage the ship. This

advancement in the shipping technology with smart devices bring intelligence to the cyber ships. It enhances the monitoring and communication capabilities to control and manage the ship. However, the use of ICT and IoT systems in the shipping technology also introduces a lot of risks related to cyber attacks, faults and failure. The shipping industry has become more vulnerable to cyber attacks because of its dependence on the ICT technology [1].

Devices on board such as GNSS, AIS, ECDIS are vulnerable to Denial of Service, jamming, spoofing and malware attack respectively [2, 3]. In addition to these devices, network switches that are used to propagate messages in the ship are also vulnerable to attacks such as Distributed Denial of Service (DDoS) attack. For instance, a DDoS attack on the network could result in the inability to control the engine system, bridge system, and alarm system, endangering the ship. Such DDoS attacks are frequently used against other types of network [4].

However, mitigation and detection of these network attacks in the traditional communication network of ship require the deployment of special purpose software and hardware devices, as well, the modification of legacy routers and switches, incurring necessary, more complex human interventions. Additionally, it requires crew members to be an expert in network and security administration to perform low-level device specific configuration. While the growing cyber attacks on the ship require frequent changes in the configuration of network and security devices. Manual configuration of these devices is tedious, complex and error prone. Clearly, the lack of dynamic and automated configuration leads to network downtime and degradation in the performance of the ship control systems.

This motivates us to design a framework that will be capable of mitigating cyber attacks within ship environment in an automated fashion. To this end, we propose utilizing the capabilities of Software-Defined Networking (SDN) architecture to design a framework for mitigating these attacks in an automatic way. Since the decoupling of control and data plane in SDN, and configuration of data plane devices from a centralized controller through programmable

interface allows us to achieve automation in mitigating attacks.

In particular, SDN is a promising technology that can assist in the modernization of the ship communication networks. Recent developments in SDN have led to a proliferation of studies on the automation and simplification of network management tasks [5, 6]. Decoupling of the control and data plane in the SDN provides the flexibility to simplify the network operation compared to traditional network management techniques, since it facilitates us to express the policies at the controller, which can be enforced into network devices depending on the status of the network [7]. In SDN, network switches behave as a simple forwarding devices, whose forwarding rules can be dynamically configured by a centralized controller. Several studies advocate employing SDN to simplify the network management tasks for improving the resilience and security in the smart grid and enterprise environment [8, 9, 10, 11, 12, 13].

Therefore, in this paper, we attempt to design a framework based on SDN to defend the ship's communication infrastructure against cyber attacks, with an attempt to improve the resilience against the attacks. The traditional communication network of ship can be changed to SDN environment incrementally. The focus of this paper is to mitigate the attacks. Our framework offers a high-level policy language and a translation mechanism to translate the high-level policies expressed at the controller into low-level rules for the enforcement in an automatic way in the ship's communication network. To experiment with the proposed framework, we have developed a use case and evaluated different performance metrics. The experimental results show that our framework performs well and incurs low computation overhead.

The rest of the paper is organized as follows. Background is described in Section 2. Section 3 presents challenges and motivations for our framework. Section 4 introduces our cyber ship framework and its different components to mitigate cyber attacks on the shipboard control system. It also presents features of SDN which can be helpful in achieving the objectives of our framework. Our policy language and translation mechanism is introduced in Section 5. The mechanism for steering the flow in our framework is presented in Section 6.

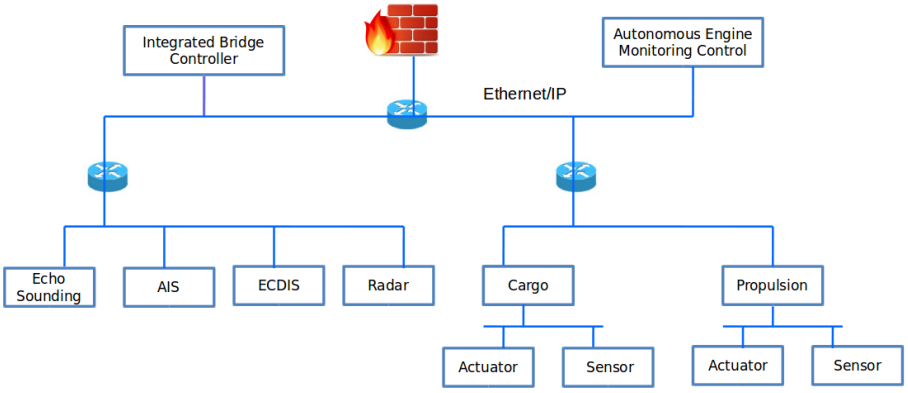

Figure 1: Ship Systems

Section 7 presents a use case showing the applicability of the framework. Experimental results are discussed in Section 8. Section 9 provide limitations and discussion about the framework. Finally, Section 10 concludes the paper.

2. Background

In this section, first, we present the critical components of the ship and discuss some works on building communication network on the vessels to connect the critical components. Second, we survey some work on protecting the communication infrastructure of the ships. Then, we survey work on policy language for comparative analysis to show how our policy language and translation mechanism can be effective in automating the attack mitigation in the ship's communication infrastructure and reducing the burden on crew members.

2.1. Ship Systems

This Section of the paper presents a description about the general ship systems and the communication infrastructure on ships. As shown in Fig. 1, the Integrated bridge controller (IBC) and the Autonomous Engine Monitoring

controller (AEMC) manage and control the different bridge and engine components, respectively, on the ship. Below, we describe the different components and then we present how these components are connected.

- **Autonomous Engine Monitoring Control (AEMC):** It manages the propulsion, propeller, cargo, etc. of the ship [14]. Depending on the scenario, it issues the control command to start, stop, increase, or decrease the speed, and to reroute the ship. Moreover, it periodically analyses the data received from the sensors of the propulsion, cargo and other components managed by the engine to check the status of the devices i.e. whether they are working properly or not.

- **Integrated Bridge Controller (IBC):** It supervises the functioning of the different bridge components of the ship such as a electronic chart display system, radar, echo sounding device, and automatic identification system [3]. It receives the data from the sensors of these devices and provide a centralized display to the crew on-board to access the data. It also issues control commands to the `AEMC` to start/stop the propulsion control system, and to reroute the ship to different routes depending on the information from the bridge devices.

- **Automatic Identification System (AIS):** The AIS devices broadcast and receive ship related information such as ship's name, type, size, IMO and MMSI number in the deep ocean to avoid collision with other ships [15].

- **Electronic Chart Display System (ECDIS):** It contains the pre-planned traffic routes, prohibited areas, next port of call, etc. Generally, it is maintained in a computer which is managed by captain.

- **Radar:** It helps to detect and monitor objects in the deep ocean to avoid collision.

- **Echo Sounding:** It provides information related to the level of water under the ship to help in navigation and maintain the balance of the

ship. Depending on the information about the water level from the `Echo sounding` device, `AEMC` either reduces the water level in the ballast or fills the ballast tank.

- **Cargo:** It contains the shipment tracking details of the containers in the cargo [3]. Depending on the design of the ship, it can be under the control of `AEMC` or can be separate [14].
- **Propulsion:** It is controlled by the `AEMC` and is coupled with the propeller [16]. It is responsible for start/stop and increasing/decreasing the speed of the ship.

The critical components on the ship are interconnected through a ship area network. The traditional ship area network is mainly based on wired networks, which use the National Marine Electronics Association (NMEA) 2000 and shipboard control network (Ethernet based MiTS) [17]. The lowest level devices (AIS, radar, etc.) are connected to a switch or gateway through NMEA protocol. Integrated bridge and engine controllers are connected through an ethernet based connection.

There have been some efforts to connect the critical systems on the ship in an efficient way for effective data transfer. Chou et.al [18] proposed a shipboard LAN to connect the critical systems on the ship, so that the captain can control and manage the ship from anywhere on the vessel. The proposed architecture is composed of a star network and all the critical systems are connected by point-to-point links to a central controller. Kim et.al [19] introduced a network design for the ship. The authors proposed to use ship area network (SAN) for connecting different networks (wired and wireless) on the ship. The experimental results show that, the SAN LAN deployed with the STAR topology can improve the communication services on the ship.

There have also been some works to provide wireless connection between different network on the ship. Jeon et.al [20] proposed a wireless communication technique between the shipboard control network and the low-level device network to achieve high data throughput. Chang et.al [21] designed a

software-defined wireless networking architecture for high performance in the ship area network. The authors combined self-organizing time division multiple access (SOTDMA) and software defined wireless networking (SDWN) to reduce end-to-end transmission delay to get high performance in ship area networking.

The focus of these works was mainly to build networks to connect critical components on the ship for effective data transfer. Our aim is to design a framework using SDN technology to achieve automated mitigation. Specifically, we connect the IBC and AEMC through SDN environment to achieve a dynamic and automated configuration in the network. In the next section, we survey some works on protecting the critical components and communication network of the ships.

2.2. Protection Mechanisms for Ship Systems

The widespread adoption of ICT in shipping, has led researchers to focus on the security properties of these systems, to understand aspects such as how security breaches within these technologies can result in a variety of harmful impacts to the ship's operation and its crew members. However, the research into ship security is in an early stage, whereby much research has focused on identifying potential threats and vulnerabilities [22, 3, 23], highlighting the risks that result from the application of ICT to critical systems on the ship. In particular, the guidelines of BIMCO [3], which is an association representing shipowners, draw special attention to the different types of cyber attacks affecting the ships and to the exploitation of vulnerabilities in the critical components [3].

These are management guidelines on how to approach the cybersecurity issue in the context of shipping, and can be used as an input for the cyber risk assessment. Bensing [24] performed an assessment of the vulnerabilities in the control system of the ship, examining the importance of critical infrastructure on a shipboard system. In a similar way, Hayes [25] identifies recent cyber attacks and incidents, and as a result generates recommendations to protect ships from these attacks. These approaches establish threats and vulnerabilities

with the aim of developing countermeasures to protect the system.

To the best of our knowledge, only limited research has dealt with the defense of the communication infrastructure of the ship from cyber attacks. Babineau et.al. [26] proposed to periodically divert the traffic between different switches in the network to protect the critical components of the ship from cyber attacks. It relies on the redundancy in the design of the ship's communication network to divert the traffic through different paths. ABB, a leading company in industrial automation, proposed placing the critical components of the ship in the core of the network, a location typically requires overcoming firewalls and Intrusion Detection System (IDS) to enter from outside [27].

Yunfei et.al. [28] and Chen et.al [29] proposed architectural solutions to protect a warship system from cyber attacks. The proposed mechanisms rely on statically deployed access controls, firewall and intrusion detection system (IDS) in the network to mitigate the attacks. Moreover, Penera et.al. [30] identifies the packet scheduling attack on the shipboard network controlled system for mitigation. A network intrusion detection mechanism based on hidden markov model is presented for communication network of ship by Wu [31]. Xing et.al [32] proposed an intrusion detection system based on collaborative control structure of ship information system. The authors focused on detecting signal attack between different components of the ship.

All these mechanisms focus on detecting and preventing cyber attacks on ships. However, our work aims at proposing a framework to mitigate the attacks in an automated way to improve the resilience of the ship control system and reduce the burden on network operator and crew member of configuring policies in the network devices manually. In this regard, we use SDN technology to design our framework to mitigate attacks on ships in an automated way. The traditional communication network of ship does not offer the flexibility of dynamic and automated configuration to mitigate attacks. Crew members have to configure the network devices manually to deploy rules for mitigation of attack. However, the global visibility and programmability of SDN make it suitable to detect and mitigate cyber attacks in an automated way. To achieve

automated mitigation, our framework proposes a policy language and a translation mechanism to define high-level policy and translate them automatically for enforcement into network devices. In Section 2.3, we survey some policy languages proposed for SDN networks.

2.3. Policy Language

Our policy language and translation mechanism leverages the benefits of SDN [33], i.e. a centralized control point and an interface between the control and the data plane to deploy the rules in the network devices. It helps the crew members and network administrator to express the policies in human understandable language for managing the communication network of ship. Crew members can define the high-level policies in simple to understand syntax without delving into low-level details.

There are a few high-level policy languages proposed for SDN networks. Table 1 compares our policy language with Procera [34] and Merlin [35]. Procera and Merlin are both high-level policy languages developed specifically to express policies in the SDN network.

Procera uses a complex syntax which makes it difficult to understand. In particular, it does not make use of controlled natural languages (CNL) [36]. CNL allows us to express the high-level policies in a human understandable format, so that network administrators need not learn device specific syntax to specify high-level policy. Procera is an event-based language, but to define an event, it needs to be registered with its global data structure. Moreover, Procera does not provide an unique policy ID to retrieve the policy. A unique index such as policy ID helps to retrieve the policies efficiently in a large database containing policies [37].

Merlin [35] enables the definition of high-level policies using programs in a declarative language. It uses the packet header fields to classify and express the policies for a set of packets. An unique identifier is not specified for policy retrieval. Merlin policies are based on the condition-action paradigm. Conditions are continuously checked to trigger policy in condition-action paradigm.

The main focus of Merlin is resource provisioning in the network to specific flows.

However, our policy language uses CNL to define the high-level policy in a human-readable format which makes it easy to understand and express. Crew members do not need to be familiar with device specific syntax or OpenFlow concepts to define policies using our high-level language. Interestingly, makes use of the Event-Condition-Action format. Moreover, it uses a unique ID to efficiently retrieve the policies from the policy repository.

Table 1: Comparison with existing policy language

	Procera	Merlin	Proposed Language
Event driven	Yes	No	Yes
Controlled Natural Language	CNL is not used	CNL is not used	CNL is used to define policy
Index	No unique ID is used	No unique ID is used	Unique ID is used to identify the policy in database
Syntax	Proc world return A: policy req -> if flow_class=legitimate and bandwidth_class=gold and event already exist then allow	(ip.src=10.0.0.1 and ip.dst=10.0.0.3) ->.*	Event=QoS_message Conditions: Flow class= legitimate Bandwidth class = Gold Action:Redirect

3. Challenges and Motivations

The aim of our framework is to enable dynamic and automated attack mitigation in the ship to protect its critical components from unavailability. To achieve this goal, the following design requirements must be satisfied.

- **Simple and Expressive:** The framework should provide a high-level language to express the policies in a controlled natural language [36] without getting into details of how these policies will be implemented. It makes the high-level policy simple to write in a human understandable

format. Moreover, a policy language should be expressive enough so that the network operators or the crew members (in case of communication infrastructure of the ship) can define their diverse intents i.e. it allows them to specify their network and security requirements and contexts depending on the varying network conditions. For example, when the context of a flow changes from legitimate to malicious in nature, the administrator would like to block the flow. On the other hand, the network administrator may want to rate-limit suspicious flows with a higher impact severity.

- **Dynamic and Automated:** The manual process of policy enforcement causes delay in adapting to varying network conditions. Moreover, manual policy enforcement requires the availability of the expert network administrator. In the case of mitigating attack in the communication infrastructure of the ship it can cause a problem, as many a times crew members are not well versed with the network configuration. Therefore, the framework must be dynamic and highly automated to free the network operators from the manual and error-prone process of policy enforcement, in particular through the configuration of data forwarding equipments.

- **Service Chaining:** The framework should allow the network administrator to specify service function chains in its network, i.e. the processing order of flows through a series of service functions or middleboxes. It helps to deploy defense in depth and specify the processing order of network traffic to protect critical components on the ship. For instance, a network administrator may want to specify that a packet coming from the offshore control center should be first processed through a firewall before traversing the core network, or that a suspicious traffic should traverse a set of specific switches or middleboxes such as firewall and DPI (Deep packet inspection) in the network.

- **Resilient:** Legitimate traffic between different components should be able to reach each other even during the attack or congestion to make sure that the ship is sailing safely. The framework should be able to recover from the attack quickly so that the ship can operate properly and should not cause any delay.

- **Easy Deployment:** The framework should not depend on the deployment of too many security devices to protect the communication infrastructure and critical components on the ship. Requirement of too many hardware and software devices will make crew members and shipping companies reluctant to deploy the defense mechanism. Moreover, network management tasks become difficult with too many hardware and software devices in the network.

4. SDN Enabled CyberShip Architecture

To achieve the objective identified in Section 3, we propose a cyber ship framework using SDN to mitigate the attacks against the ship's communication infrastructure. The focus of the paper is not on securing the SDN. There has been a lot of work to secure the SDN from the attacks [38, 39]. Our aim is to realize the dynamic and automated mitigation of cyber attacks using SDN.

The features of SDN which can be advantageous for the cyber ships are as follows:

1. **Centralized Controller:** In SDN control plane is decoupled from the data plane and it is centralized as an entity called controller. It communicates and configure the data plane devices depending on the policies. Security and network policies can be expressed at the controller which can be deployed automatically by the controller depending on the need. Moreover, centralized controller offers a global visibility of the network. In the cyber ship environment, it can assist in monitoring and configuring the network to divert or block the traffic if the needs arises.

2. **Flexibility:** SDN can enable us to easily upgrade network application in the ship system.

3. **Standard API:** The OpenFlow protocol provides a standard API that allows to manage the distributed network equipment. It facilitates us to issue the command to configure the network device according to the requirements from the centralized location.

4. **Programmability:** SDN allows creating a range of customized applications. In the cyber ships this feature can be useful to define the high-level policy language which can be translated and deployed from the centralized controller. Additionally, it facilitates us to deploy the security rules dynamically in the network where it is required.

5. **Global visibility:** SDN makes it possible to have the global visibility of the network. This can simplify the network management tasks, offering network administrators global view of the network and allowing to control data plane devices from a centralized controller [40]. This can help to monitor the cyber ship network effectively and configure the network devices to divert the traffic depending on the requirements such as in case of congestion or if a device is not working properly.

4.1. Components of the Framework

In this section, we describe the components of our framework. It consists of the different cyber physical components. Firstly, we describe the physical and data plane components and then we discuss the control plane components.

The physical and data plane components of the framework is mainly comprised of SDN switches and the physical layer devices such bridge and engine devices. The major components are shown in Fig. 2, while the details are given below: The components are as follows:

1. **Sensors and Actuators:** Sensors and actuators are attached to the different physical components of the ship related to the bridge, engine and

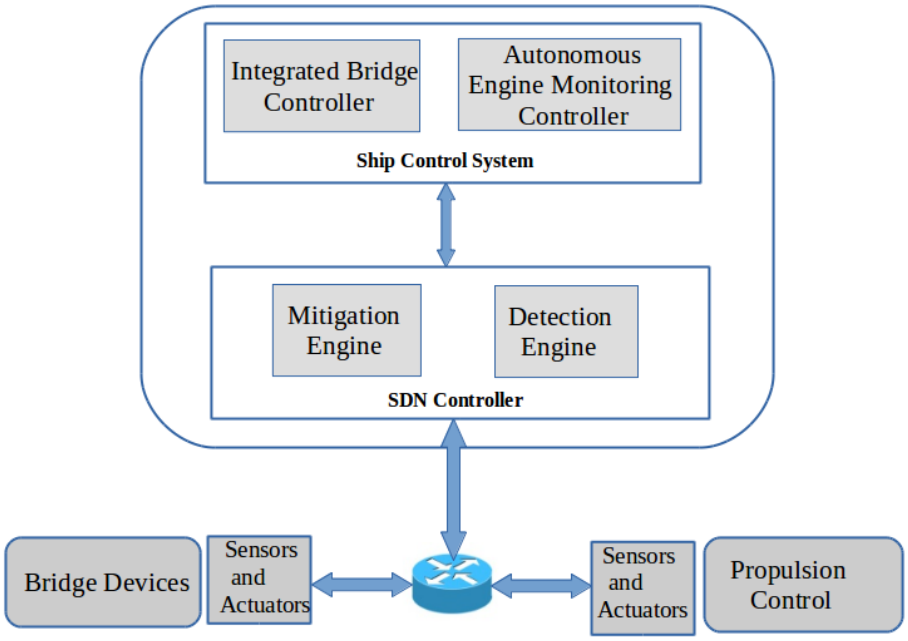

Figure 2: Framework of the CyberShip

propulsion control devices. These sensors forward the data related to these physical devices to `Integrated Bridge Controller` and the `Autonomous Engine Monitoring Controller` for analysis.

2. **SDN switches:** These are the programmable switches deployed at the data plane of the framework. Through southbound API of the controller these switches can be dynamically configured by the applications deployed at the controller.

Next, we describe the components in the control plane of the framework.

1. **SDN controller:** It is a software platform deployed as a remote entity able to provide the network abstractions needed to manage the network [41]. It provides centralized intelligence and global network visibility to control and manage the network. It offers an interface with the switches which makes it possible to deploy the rules in the switches through a

centralized location depending on the network status.

2. **Detection Engine:** The framework employs a detection engine to examine the network traffic to identify suspicious and malicious activities. Because of the modular design of the framework, network administrators have the ability to deploy the detection mechanism according to their requirements. Network operators can deploy some mechanisms to classify the suspicious and malicious flows [42, 43]. Upon detection of the suspicious or malicious traffic, the detection engine in the framework reports a security alert to the mitigation engine.

3. **Mitigation Engine:** It is responsible to take appropriate countermeasures to mitigate the attacks in the cyber ship. It contains a repository consisting of security and network policies defined in high-level language to mitigate the attacks. Depending on the security alert, countermeasure policy is instantiated to mitigate the suspicious or malicious traffic. Details about the high-level policy is given in the Section 5.

   `Mitigation Engine` also maintains a table containing the location of middleboxes (firewall, Intrusion Detection System (IDS)) in the network i.e. switch and port through which the devices are connected (Details are given in Section 7). Furthermore, it maintains a list of network paths with switch ID and output port along with its label to reach the different middleboxes (firewalls, IDS, etc.) or to reroute the traffic through different path. The path is defined as a list of switch ID and the output port, and label is associated with an end-to-end path. Labels can be inserted in the VLAN ID field or as an MPLS header in the flow. Labels help in fast forwarding the traffic in the network as the core switches only check the label for forwarding. Moreover, core switches do not need to maintain too many flow entries in the flow table as the forwarding is performed based on the labels.

4. **Autonomous Engine Monitoring Controller (AEMC):** It manages and

controls the different components of the engine. In the architecture, it is positioned at the SDN controller as shown in Fig. 2; however, it can be deployed on a separate SDN controller. Upon detecting attack or fault in the engine component it informs the `Mitigation Engine` to redirect the traffic through another route. Section 2.1 describes about the `AEMC`.

5. **Integrated Bridge Controller:** It offers a centralized platform to show the information from different bridge devices. Upon detecting the fault, failure or attack on the bridge devices, it notifies the `Mitigation Engine` to divert the network traffic through another route or to the middleboxes for further processing. On a similar pattern of the `AEMC`, it can be deployed on a separate SDN controller. IBC has been described in the Section 2.1.

5. Security Policy Specification

In this section, we describe how the high-level policies are expressed in the mitigation engine module of the framework. These high-level policies are translated into low-level OpenFlow rules for the enforcement in the SDN switches when the need arises. The automated policy deployment eliminates the need of manual configuration for policy enforcement. One of the main objectives of our policy language is to allow crew member with little ICT skills to write simple network and security policies to manage the ship network.

5.1. Grammar of High-level Policy

Listing 1: Grammar for the High-level policy language

```
1 <Policy>=<PolicyID><Target><Rules>
2 <Target>=<DeviceID><IP_Address>
3 <Rules>=Set(<Rule>)
4 <Rule>=<Event><Conditions><Action>
5 <Event>=<Attack_Type>|<Fault_Type>
6 <Conditions>=<Condition>[<Connective><Conditions>]
7 <Condition>=<Field_Name><Comparison_Operator><Value>
8 <Connective>=And|Or
```

```
9  <Comparison_Operator>=less than|equal to|greater than|Not
10 <Action>=Drop|Redirect|Forward
```

The high-level policy syntax provides the guidelines to the security administrators to define the abstract policy. To do that, we need to define a grammar and formats. It enables the network operator or the crew member with little network and security expertise to express the security policies into an easy to understand language without getting into low level implementation details.

In the framework, we use Event-Condition-Action (ECA) paradigm for policy specification [44]. The reasons for choosing ECA is two-fold: (1) it allows to specify a wide variety of events which can trigger conditioned actions; (2) it enables to define an asynchronous notification mechanism to react on these events. Our policy grammar shown in Listing 1 provides the syntax to define the security and network policies in a human readable format. The high-level policies are expressed by the network operator in a simple way through the northbound API of SDN controller.

Table 2: Syntax to Specify Policies

	Keyword	Match
Events	UDP_Flood, TCP_Flood, Any_Flow	Source IP, Destination IP, DeviceID
Conditions	Flow Class, Impact Severity, Traffic Type, Component Status	Destination IP, IP Protocol, Input Port, VLAN ID, Available, Down
Action	Drop, Redirect, Forward Fwd_Middlebox	Action={Output Port},Drop=No Forward

In particular, our policy is composed of a `PolicyID`, `Target` and a set of `rules`. The `PolicyID` helps in uniquely identifying a policy, as there are many different policies in the mitigation engine module. The `Target` specifies the device for which policy should be enforced. Each `rule` is composed of an event, some conditions and an action. Event represents what triggers the rule. In Listing 1, we exemplify events, such as attack and fault type. However, it is not limited to these events only, other types of events can also be defined using our policy language. A network operator only needs to specify the corresponding conditions to define new events. Table 2 shows the list of events and match

at the low level OpenFlow rule for the deployment in the SDN/OpenFlow switches. For instance UDP and TCP flood represent the events. These events at the low level rule specification are specified by `Source` and `Destination IP` addresses.

When an event is triggered, the associated parameters are checked against the condition specified in the policy. Condition is generally a boolean expression that can be evaluated as true, false or not applicable. Not applicable represents that no condition is specified for the event. In our grammar shown in Listing 1, Condition is specified with the field name and a value. Field name contains the name of the condition and value represents the parameters specified in the condition. Table 2 shows the list of conditions and match for the conditions at the low-level OpenFlow rule. However, our policy language is not limited to only these conditions. Network operators can include other information for the condition and concrete information. At the low-level rule specification conditions are represented by `Destination IP`, `Input Port`, `VLAN ID`, and `IP protocol`.

Flow class, impact severity, traffic types, component status shown in Table 2 are the part of condition in our policy language. Specifically, Flow class categorizes the flow into three different classes known as: suspicious, malicious and legitimate. Suspicious indicates that the flow is a mix of legitimate and malicious flow. The flow which is confirmed as an attack traffic is specified as malicious flow. The benign traffic is defined as legitimate flow. Traffic type specifies the different types of flows based on their protocols such as UDP traffic, HTTP traffic, TCP, etc. Impact severity specifies the impact of attack traffic or traffic causing the congestion in the network, and the values are set as low, medium and high. Component status indicates whether a critical component of the ship is working or is down.

Action specifies the high-level decision which should be enforced when the conditions are met for the event. In Listing 1 three actions have been specified. Generally, OpenFlow specifies two high level actions:Forward and Drop [45]. In Table 2, we have listed a few high-level OpenFlow actions for

our framework. High-level action `Drop` is enforced when it is confirmed by the `Detection Engine` or controllers managing the critical components of the ship that traffic is malicious in nature. `Redirect` action is enforced when there is suspicious flow confirmed by the `Detection engine` or a component is not working and traffic needs to be routed to secondary component. Suspicious traffic is not directly blocked rather it is diverted through different path with low-bandwidth to reduce the collateral damage caused to the legitimate traffic. Moreover, directly blocking the suspicious traffic would also harm the legitimate traffic as, suspicious flow is mixed of legitimate and malicious flow. Forward action is enforced for the legitimate traffic. `Fwd_Middlebox` specifies to route the traffic towards middleboxes such as firewall and IDS.

5.2. Examples

To demonstrate the expressiveness of the policy language, we present some examples based on network and security policies.

Access Control: The access control policies enable us to express whether flows are allowed in the network or not. For example, to force traffic coming from the shore control center through firewall, a simple high-level policy can be defined as follows:

`Flow:` 175.100.1.1; `Traffic_Type:` Any; `Action:` Fwd_Firewall.

The IP address in the above policy indicates that the traffic from a shore control center is subject to pass through a firewall.

Resource Isolation: It is a good practice to completely separate the different resources in the network. For example, it is required that the passenger's entertainment network on the ship is separate from the network of critical components such as the engine, since it can affect the safe and secure of the ship. The policy below indicates that the traffic originating from a source and going to a certain destination network should be dropped.

`Flow:` 192.158.1.1; `Destination_IP:` 172.157.1.1; `Action:` Drop.

The policy above ensures that the traffic originating from a source and going to a certain destination network should be dropped.

Quality of Service (QoS): QoS can be defined, depending on the flow class and source of the flow. For instance, legitimate flow originating from integrated bridge controller is given high bandwidth path.

`Flow:` 154.70.56.1; `Flow Class:` Legitimate;

`Action:` high_bandwidth_path

The above policy statement states that the legitimate flow coming from bridge controller should be forwarded through a high bandwidth path.

Service Chaining: It protects a system by chaining the different security middleboxes such as firewall, Intrusion Detection System (IDS), etc. The following policy specifies that the traffic going towards engine controller should traverse through a set of middleboxes. The IP address in the below policy statement denotes the engine controller.

`Source:` Any; `Destination:` 197.214.100.1; `Action:` Fwd_Middleboxes

Depending on the high-level action and condition (e.g., destination IP address) mentioned in the above policy statement, `Mitigation engine` module checks its database to redirect the flow through a path containing the list of middleboxes to be traversed.

In Summary, our policy language allows crew members to specify different set of network and security policies. The high-level policy is translated into low-level rule for the enforcement in the network devices. In Section 5.3, we describe the format of the low-level rule depending on the high-level policy.

5.3. Semantics of the Low-level Rules

A policy that has been defined with the high-level language must be translated into low-level rules in order to be enforced in the network. Translation uses an intermediate format. High-level policies are translated into common format rules, and then the common format rules are translated into device specific configuration. It is important that these steps do not introduce ambiguous rules. In this section, we present a formal way to translate the high-level policy into low-level rules for enforcement.

In our framework, a packet-type consists of a list of field names of the packet along with their types. For example, the packet-type is given by (srcip : IPADDR, dstip : IPADDR, protocol : proto). A concrete packet specifies a value for each field of type corresponding to that field of packet. A representative value of a type 'T' is a concrete value of 'T' and '*' represents a wildcard value for the packet.

An event is represented as pair (pf,a), where `pf` is a symbolic packet field and `a` is an action. With this notation, rule to drop a malicious flow is specified as: (srcip=ip1, dstip=ip2, protocol=udp, action=drop).

Similarly, using this notation, a rule to forward a flow can be specified as: (srcip=ip1, dstip=ip2, protocol=udp, action=output:port). This rule represents a scenario where a UDP flow from `ip1` to `ip2` can be forwarded through an output port.

A rule is called concrete if all the symbolic values in packets have only concrete values. It is obtained by replacing each variable and wildcard by possible value of the corresponding type.

To evaluate a condition for a flow, a test on a flow is performed which is of type CT(f1,...,fn)=c, where fn is a packet field and all fn's are different. In our example, CT(srcip)=x and CT(dstip=y) is a test with the field srcip and dstip respectively.

Listing 2 shows the format of the low-level rules.

Listing 2: Format of the low-level rules

```
1 <Rules>=Set(<Rule>)
2 <Rule>=<Packet_header> "->" <Action>
3 <Packet_header>=<srcip>":"<dstip>":"<port>":"<protocol>":"<vlan_id>
4 <Action>=<Output_Port>|Drop
```

A packet consists of header field. Depending on the high-level action the low-level rule is enforced in the switches by specifying action on the packet header. In the low-level rule different packet header fields can be specified depending on the conditions in the high-level policy and the location of the

switch where the rule is to be deployed.

5.4. OpenFlow Rule Templates

OpenFlow rule templates provide a way which allows to instantiate the rules for different tasks in the network which can be enforced by different devices. It provides a view on what parameters the low level rules should be specified in the data plane devices for different types of actions. Moreover, it provides an additional level of modularity to network operator to specify the parameters for the translation of high-level action into low-level OpenFlow rules. OpenFlow rule templates are maintained in the *mitigation engine* module.

5.4.1. Template for specifying the path

In our framework, a path is specified as a set of switch IDs and output ports. The Listing 3 shows a template to define a path with the switch ID and the output port to route the flow through a specific path. Each path is associated with a unique label which helps in fast forwarding the flow through that path. The template to specifying the path is maintained in the *mitigation engine*.

Listing 3: A template for specifying the path

```
1 Path={Switch:'SwitchID','Port':Output_Port,...,Switch:'SwitchID','Port'
      :Output_Port}
2 Label={PathID:VLAN_ID}
```

5.4.2. Template for action `forward`

The high-level action `forward` is used for routing the traffic through a path. The Listing 4 describes the template for action `forward`, which shows that at the ingress switch, forwarding rules are specified with: source IP, destination IP address, setting a label in the Vlan ID field, and `forward` the flow through output port. Clearly, it can be used for forwarding the flows originating at a specific source and going to a particular destination. In the core switches, the forwarding rules are specified only with matching a label and forwarding the flow through an output port. Moreover, at the egress switches rules are

specified with: destination IP address, with deleting the label and forwarding the flow through an output port.

Listing 4: OpenFlow rules templates to forward the flow

```
1 Ingress Switch: ['Source_IP': addr, 'Destination_IP': addr, '
      SetLabel':Vlan_ID,'Action':Out_Port]
2 Core Switch: ['Action':Out_Port,'Label':Vlan_ID]
3 Egress Switch:['Destination_IP': addr, 'Label':pop,'Action':Out_Port
      ]
4 All the flows originating at a source: ['Source_IP': addr, 'SetLabel
      ':Vlan_ID,'Action':Out_Port]
5 All the flows traversing to a destination: ['Destination_IP': addr,
      'SetLabel':Vlan_ID,'Action':Out_Port]
6 Specific flows originating at a source: ['Source_IP': addr, '
      Protocol': proto, 'SetLabel':Vlan_ID,'Action':Out_Port]
```

5.4.3. Template for action `drop`

The action `drop` is used to block the traffic. Listing 5 shows an abstract template to specify the action `drop` depending on the condition in the network. For instance, to drop a specific flow, we specify its source IP, destination IP and protocol. Moreover, dropping all the flows originating from a particular source requires to specify only source IP and action drop. Similarly, to filter all the flows traversing to a particular destination, requires only destination IP address and action drop. Blocking rule is generally specified at the ingress switch of the network, so that it won't cause collateral damage to legitimate traffic in the core network.

Listing 5: OpenFlow rule template to Drop the Flows

```
1 Drop specific flows: ['Source_IP': addr, 'Destination_IP': addr, '
      Protocol':proto,'Action':Drop]
2 All the flows originating at a particular source: ['Source_IP': addr
      , 'Action':Drop]
3 All the flows going to particular destination:['Destination_IP':
      addr, 'Action':Drop]
```

5.4.4. Template for action `redirect`

The `redirect` action is used to divert the traffic from its previous path to a new path because of link failure or congestion in the current path of a flow. Listing 6 shows a template to specify action `redirect` at the switches in the network. For instance, to redirect specific flows, OpenFlow rules are specified with: source IP, destination IP address, modifying a label and forwarding the flow through an output port of ingress switch. Moreover, to redirect all the flows originating from a particular source, we require only to specify the source IP address of the flow and setting a new label in the `VLAN ID` field at the ingress switch. Similarly, to redirect all the flows traversing towards a particular destination, it only requires the destination IP address and setting a new label on the flow and specifying the output port of the ingress switch to redirect the flow.

Listing 6: OpenFlow rule template to redirect the flow

```
1 Redirect specific flows: ['Source_IP': addr, 'Destination_IP': addr,
      'SetLabel':Vlan_ID,'Action':Output_Port]
2 Redirect the flows originating at a specific source: ['Source_IP':
     addr, 'SetLabel':Vlan_ID,'Action':Output_Port]
3 All the flows going to particular destination:['Destination_IP':
     addr, 'SetLabel':Vlan_ID,'Action':Output_Port]
```

The deployment of the policies using the templates is based on Algorithm 1. It takes three inputs:

1. The high-level action which can be either one of the three: `forward`, `redirect`, and `drop`.
2. The path details containing the set of switchIDs and output port.
3. Flow information consists of source, destination IP, and protocol.
4. Network Service Header (NSH) contains the VlanID which identifies the path.

Based on these inputs, Algorithm 1 composes the rules based on the Open-

Flow rule templates described above. Finally, the algorithm deploys the rules in the switches for processing the traffic. The rules are deployed in the core switches then at the entry switches, leading to the reduction of packet loss, especially when the flow is redirected through another path. The detailed information about the NSH are describe in Section 6.

Algorithm 1 *OpenFlow rule deployment*

procedure POLICY_DEPLOYMENT(Action,path,flow,NSH)
  $path \leftarrow [SwitchID : outpput\_port, ..., SwitchID : output\_port]$
  $Action \leftarrow (forward, redirect, drop)$
  $A \leftarrow Action$
  $NSH \leftarrow VlanID$
  $flow \leftarrow (sourceIP, destinationIP, protocol)$
  $forward \leftarrow template\_forward[forward, path, flow, NSH]$
  $drop \leftarrow template\_drop[flow, drop]$
  $redirect \leftarrow template\_redirect[redirect, path, flow, NSH]$
  for Action A in concrete rules **do**
    $rule \leftarrow (A.flow, A.NSH, A.output\_port)$ // Compose the rule with the information
  end for
  $Deploy\_rule(rule)$
end procedure

6. Network Service Header

Our framework uses Network Service Header (NSH) [46] to steer the traffic. NSH contains an end-to-end path identifier (global segment) and a middlebox or switch identifier (local segment). It helps in specifying the processing order of the traffic through a set of middleboxes or switches in the network to perform service chaining. Specifically, local segment avoids the unnecessary processing of the traffic through middleboxes if it is not needed. It reduces the processing overhead on the middleboxes as well as the delay caused to the traffic because of middlebox processing. In particular, we use VLAN ID (MPLS and VXLAN header can be used as well) to provide NSH in the packets. Both local and global segments are stacked as labels in the NSH for forwarding, and these two labels are described as follows,

1. **Global Segment** helps to steer the flows from the ingress switch to the destination in the ship's communication network. Thanks to this label, core switches in the path can easily identify the output port through which a packet is to be forwarded. Flow states containing NSH is maintained only at ingress switch. It significantly reduces the flow table entries in the core switches [47].

2. **Local Segment** is used to identify the middlebox through which the traffic of interest needs to be processed. It avoids broadcasting the traffic towards the middleboxes for processing [48]. Moreover, middleboxes are not required to be deployed outside the main datapath in the network. It reduces the processing overhead on the middleboxes by avoiding the processing of all the traffic in the path through the middleboxes, if it is not needed.

It is worth noting that although inserting extra header may increase the size of packets, the processing time for switches and middleboxes can be significantly saved. Moreover, NSH can reduce the number of flow table entries in the core switches, because its operations only occur at the ingress switch of the network. Moreover, software switches can be used at the ingress point to overcome the limitations of flow table entries. In the paper, we have used the term label more often instead of NSH.

7. Use Case

This Section gives a use case about DDoS attack mitigation for policy enforcement to improve the resilience of the `CyberShip`. We choose the scenario of mitigating DDoS attack because in the shipping industry it is one of the frequently used attack method, and it is still the most prevalent attack in the Internet [4, 3]. The main purpose of the use case is to exemplify the process of policy translation and enforcement in the data plane devices for improved resilience in the communication infrastructure of the ship. For brevity purpose,

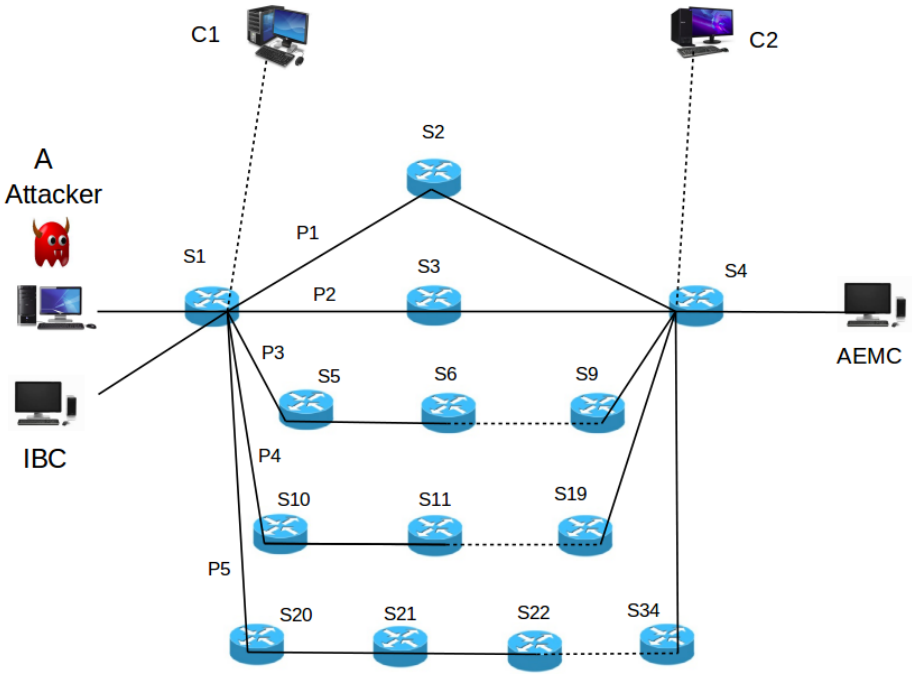

Figure 3: Configuration of the network topology in the experimentation

in Fig. 3, we have shown an example of mitigating the attack targeting only `AEMC`. However, it can be applicable to protecting other critical components of the ship as well.

The network topology is illustrated in Fig. 3, consists of an attacker denoted as A, `IBC` and `AEMC`. Moreover, `Mitigation` and `Detection` engine are deployed on a separate controller denoted as C_1 and C_2 respectively. Controller C_1 is connected through the switch S_1 and manages all the switches in the network except the switch S_4. Controller C_2 and `AEMC` are connected through the switch S_4. In the scenario, `Detection Engine` is deployed close to the `AEMC`, as the detection can be performed effectively close to the system under protection. In the use case, detection is performed based on the threshold set for packet arrival rate, average of bytes per flow, average of duration per flow [42]. If the packet arrival rate and average of bytes per flow is greater than the threshold

then `Detection engine` triggers an attack. In the scenario, we assume that the bridge controller collects and analyses the data received from the different bridge devices. It sends the messages to the `AEMC` either to increase or decrease the speed or to reroute the ship through different waypoints.

In the following, we firstly describe the settings with respect to the deployment of our translation mechanism. We will then give a step-by-step example to showcase how a high-level mitigation policy can be translated into low-level rules for enforcement.

7.1. Settings for Attack Mitigation

Alert Message sent by the AEMC: Security alerts are used to report the attack traffic. It contains the network and assessment attributes. We use the security alert in the IDMEF format, since it offers the flexibility to add additional data field [49]. However, our framework offers flexibility to use different formats as well, such as IODEF (Incident Object Description Exchange Format) [50]. Listing 7 shows the security alert forwarded by the `AEMC` to the SDN controller.

- The network attributes are comprised of the IP addresses and attack type. The security alert contains network informations such as: source IP: 10.0.0.1 (A), destination IP:10.0.0.3 ($AEMC$), attack type which is represented as an event in the alert is: UDP_Flood.
- The assessment attributes describe the effect of the attack on the network. For instance, a low impact severity, as shown in the alert represents that the impact of the attack is not very critical. Moreover, we extend the IDMEF alert with an additional data of flow class represented as suspicious.

DDoS Mitigation Policy at the SDN Controller: It is comprised of the policy to process the received security alerts. The example policy shown in Listing 8 provides high-level action `Low-suspicious-path` for the UDP flood traffic with low impact severity.

Listing 7: Security alert sent by AEMC

```
<IDMEF–Message version="1.0">
<Alert>
 <Analyzer analyzerid="AEMC"/>
 <Source>
   <Address category="ipv4–addr">
         <address>10.0.0.1</address>
   </Address>
 </Source>
 <Target>
   <Address category="ipv4–addr">
        <address>10.0.0.3</address>
   </Address>
 </Target>
 <Classification event="UDP_Flood"> </Classification>
 <Assessment>
       <Impact severity="Low"/>
  </Assessment>
 <AdditionalData>type="string"meaning="flow␣class">
 <string>Suspicious</string>
 </AdditionalData>
</Alert>
</IDMEF–Message>
```

Instantiation of the template to process suspicious and malicious traffic: It contains the information about redirecting the suspicious and malicious traffic to middleboxes or through paths provisioned for them in the network. Path with higher number of hops are provisioned for the suspicious traffic with different impact severity. For example, path with the highest number of hops and lowest bandwidth is provisioned for the traffic with high impact severity. This template is maintained in the `Mitigation` engine module. Listing 9 provides the concrete path details along with the switchID and the output port for steering the suspicious traffic. Labels shown in Listing 9 at the end of the path details are used to identify the paths. For example, a label associated with the path P_3 is "3". The label is inserted in every flow to steer it through a path. We

have used Vlan ID in our use case. However, other fields such as MPLS can also be used to insert the labels.

Listing 8: Policy to redirect suspicious traffic

```
<Policy PolicyID="Mitigation">
        <Event Type = "UDP_Flood">
    </Event>
        <Condition>
       <flow class="Suspicious"/>
       <Impact severity="Low"/>
    </Condition>
    <Actions action="Low-suspicious-path"/>
    </Actions>
</Policy>
```

Listing 9: Paths details containing switch and output port details

```
P3:{Switch:S1,output(6);Switch:S5,output(2);Switch:S6,output(3);...;
    Switch:S9,output(2),Switch:S4,output(2)}
P4:{Switch:S1,output(7);Switch:S10,output(2),Switch:S11,output(2);...;
    Switch:S19,output(2),Switch:S4,output(2)}
P5:{Switch:S1,output(8);Switch:S20,output(2),Switch:S21,output(2);
    Switch:S22,output(2);...;Switch:S34,output(2),Switch:S4,output(2)}
Path={P3:3, P4:4, P5:5}
```

Table 3: A mapping table between high-level action and the path

High-level action	Path
Low-suspicious-path	P3
Medium-suspicious-path	P4
High-suspicious-path	P5

7.2. Step-by-Step Example

In this subsection, we describe a step-by-step example of how a high-level mitigation policy is transformed into low-level rules for enforcement with an aim of mitigating the impact of attack traffic. This use case is described using the sample network shown in Fig. 3 and the high level policy presented in Listing 8.

The different steps are described below:

1. Upon detecting an attack from the machine, denoted as "A", `Detection` engine sends security alert to the `Mitigation` engine deployed at the SDN controller C_1. It sends an alert in the IDMEF format for processing the attack traffic. Attacker "A" launch the attack at the rate of 50 Mbps. For the experimentation in our topology, we used IPERF [51] to generate the traffic of higher rate to launch the attack.

2. Upon receiving the alert, `Mitigation` engine extracts the informations from the alert message. Extracted alert information are: source IP (10.0.0.1), destination IP (10.0.0.3), event type (UDP_Flood), flow class (suspicious) and impact severity (low).

3. After the extraction of the alert information, `Mitigation` engine checks its policy database to get the high-level action for the enforcement. Depending on the event type (UDP_Flood), conditions: flow class (suspicious) and impact severity (low) it gets the high-level action as a "Low-suspicious-path". The high-level policy for mitigating the UDP flood traffic is shown in Listing 8.

4. The high-level action "Low-suspicious-path" is further refined to get the concrete path details for redirecting the traffic. Based on the high-level action "Low-suspicious-path" `Mitigation` engine checks the Table 3 to get the path to reroute the traffic. In this case, it receives the path P_3 to redirect the flow. In the topology, we have provisioned the 40 Mbps of bandwidth for the traffic with low impact severity; and 30 and 10 Mbps for medium and high impact severity respectively.

5. Then, template shown in Listing 9 is used to get the concrete path details along with its label depending on the high-level path name P_3. The label associated with the path P_3 is "3". Earlier, flow from 10.0.0.1 to 10.0.0.3 was traversing through the path P_1.

6. The high-level action "Low-suspicious-path" along with the concrete path details and flow information (10.0.0.1, 10.0.0.3) are used by the mitigation engine to enforce the rules in the switches to redirect the suspicious flow.

To update the rules, `Mitigation` engine modifies the label in the VLAN ID field to "3" and the output port information to "6" at the switch S_1. Earlier, the label in the VLAN ID field of the flow from the source IP (10.0.0.1) to destination IP (10.0.0.3) was "1", since the flow was traversing through the path P_1. Once a policy has been modified in the data plane devices, traffic is redirected through another path. As in this case, we are not redirecting the suspicious flow to middleboxes so the local segment is not stacked in the label.

8. Experimental Results

This section describes the results of our experimentations to assess both the effectiveness of the response mechanism in reducing the impact of attack traffic and high-level policy translation mechanism. The scenarios are created using the Mininet emulator [52]. Experimentation was carried out to evaluate the effectiveness of the response mechanism in filtering the malicious traffic and redirecting the suspicious traffic through another path in the network to reduce the collateral damage caused to the legitimate traffic. As explained earlier, suspicious flows have been classified in three classes according to their impact severity: low, medium and high. So, we provision different paths for the suspicious traffic in the topology. The methodology to classify these flows in different classes is outside the scope of this paper. We rely on the existing mechanisms to detect and classify the flows [53].

8.1. Evaluation Metrics

The objective of our experiments is to evaluate the time to implement the high-level policies into low-level OpenFlow rules in the switches. Moreover, we also aim to evaluate how the framework can successfully reduce the collateral damage on a legitimate traffic caused by the attack traffic. The metrics

Table 4: Defined metrics to evaluate the prototype

Metric	Definition	Unit
Implementation time	It measures the time taken to translate the high-level policy and deploy them as OpenFlow rules in the switch. It increases with the increasing number of data plane devices for the deployment of the rules.	seconds
Packet loss	It evaluates the number of packets lost due to congestion or attack. Packet loss increases as long as the network is congested. Packet loss also occurs during the deployment of the rules in the switches because of some delay in the deployment.	number of packets
Throughput	Volume of traffic received in average unit of time. It decreases when the congestion increases in the network.	Mbps
Network jitter	Measures the variation in arrival time between packets and further provides the understanding of congestion or attack traffic on the system. It increases dramatically in the presence of congestion.	milliseconds

used to evaluate our prototype are specified in Table 4. They include the implementation time of the policy, packet loss, throughput and network jitter.

8.2. Implementation time of mitigation policy

From Fig. 4, we can see that the implementation time to deploy the policy to handle suspicious flows remains under 28 milliseconds, even for flows with high impact severity. It is significantly higher than the implementation time of mitigation policies with low and medium impact severity and policy to block the malicious flows. Since the path with the highest number of hops is provisioned for processing the high impact severity flows. If we consider the number of switches in the path for the deployment of the rules, it is still reasonable in between 25 to 30 millisecond. The implementation time to deploy the policy to handle low and medium impact severity flows are around 8 and 18 milliseconds respectively.

Interestingly, the deployment time to block malicious flows is significantly lower as compared to the deployment time of other policies. It is because of the fact that the block action is only enforced at the ingress switch and no further

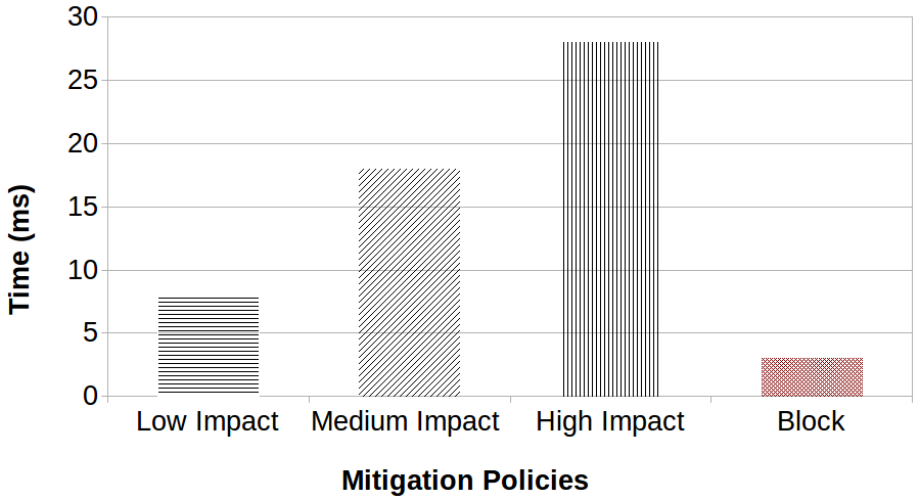

Figure 4: Deployment time of mitigation policies

deployment is required.

We conclude that the deployment time of the policies is reasonable, but it can be further reduced if the number of devices for policy deployment can be reduced. This can be done with pre-installed rules in some devices of the network. If the policies are pre-deployed in the core switches, then only at the border switches policies need to be deployed dynamically reducing the overall implementation time.

We also conducted the experimentation to evaluate the deployment time of the policy to handle suspicious flows with varying traffic rates. We wanted to evaluate whether traffic rate affects the implementation time of the policy. Fig. 5 shows that the deployment time of the low-level rules is close to constant corresponding to the high-level policy to process the suspicious flows of high impact severity. Experimentation was run with varying traffic rates from 1 Mbps to 15 Mbps. In all the scenarios, we found that the time to translate the high-level policy into low level rules for deployment are same after the security alert is received by the mitigation engine. It shows that the traffic rate does not affect our policy translation process and the deployment of corresponding low-

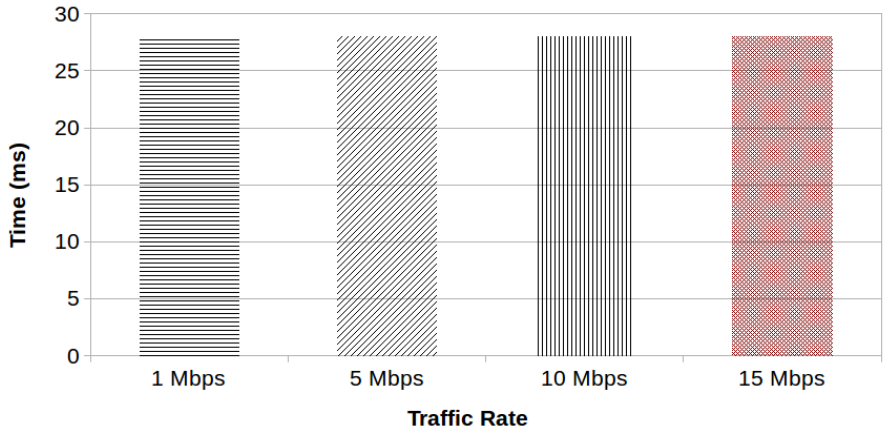

Figure 5: With varying traffic rate, time required to deploy the rules to process the high impact severity flows

level rules in the network.

8.3. Malicious traffic filtering

Filtering malicious flows at the border router of the communication network of the ship is better, as it reduces the impact on the legitimate traffic traversing through the core network. Therefore, we measured the number of packets that bypassed the ingress switch while the security alert to drop the malicious flows was being processed. The number of packets that cross the ingress switch with varied traffic rates were evaluated. As can be seen in Fig. 6, when the traffic rate is 1 Mbps then around 18 packets crossed the ingress switch and reached the `AEMC`. It is worth noting that the number of packets that crossed the ingress switch and reached the `AEMC` increases as the traffic rate increases. As shown in Fig. 6, when the traffic rate is at 5 Mbps then close to 125 packets are able to reach the `AEMC` which was under attack. There is a sharp increase in the number of packets that reached the `AEMC` when the traffic rate increases from 10 to 15 Mbps. It occurs because of the minimum delay in processing the security alert and the deployment of low-level rules to drop the malicious flows. Moreover, once the traffic is blocked, controller C_1

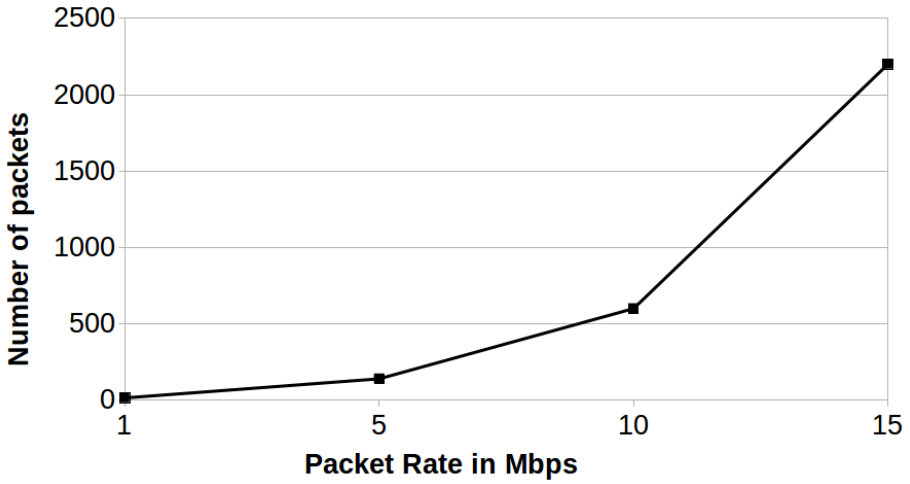

Figure 6: Number of Packets that bypass during the implementation of block action at the ingress switch

can check the flow statistics of the blocked traffic periodically to know whether the traffic is still coming or it is no longer active.

8.4. Packet loss

We measured the packet loss during the deployment of low-level rules for the suspicious traffic with low impact severity according to our mechanism. Our mechanism relies on deploying the low-level rules in the core switches first, before modifying the rules in the egress and ingress switches. Therefore, when the rules are being deployed in the core switches flow traverses the previous path and it reduces the packet loss to a significant level. However, without our mechanism low-level rules are deployed in order from ingress switch to egress switch.

As shown in Fig. 7, packet loss also follows an increasing trend. But, with our mechanism, we are able to minimize the packet loss to a large extent. It can be seen, in Fig. 7, that there is no packet loss when the traffic rate is around 1 Mbps. Even at 15 Mbps of traffic rate, packet loss is around 7 percents as compared to the 32 percents, without our mechanism. It shows that our mechanism

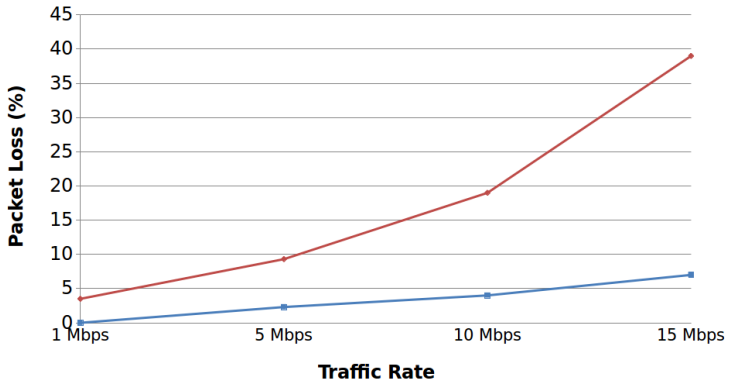

Figure 7: Packet loss during the deployment of the policy

greatly reduces the packet loss percentage while deploying the low-level rules in the OpenFlow switches.

In summary, these results show that, our strategy reduces the packet loss which relies on deploying the low-level rules in the core switches first, before modifying the rules in the egress and ingress switches. Modification of rules at the ingress and egress switches results the redirection of the flow through new path. Nevertheless, packet loss still occurs because of the delay in deploying the low-level rules from the controller to the border switches.

8.5. Throughput of legitimate traffic

Throughput of legitimate traffic was measured in the presence of DDoS attacks. As shown in Fig. 3, we used attacker (A) to generate DDoS attack traffic, and observed the impact on the legitimate traffic going to the `AEMC` from the `IBC`. As we can see in Fig. 8, the throughput of the legitimate traffic dropped sharply as soon as attack started. As a result, the *detection engine* sends an alert, which contains the flow information (source and destination IP), security class (suspicious) and impact severity (low) to the *mitigation engine* deployed at the controller C_1. Subsequently, the *mitigation engine* extracts the alerts and checks its policy database for the appropriate policy deployment. Finally, the

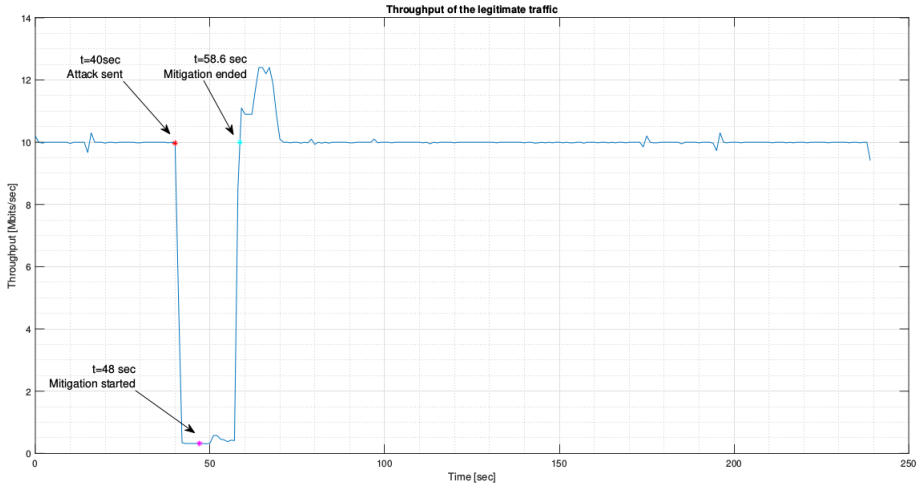

Figure 8: Throughput of legitimate traffic

corresponding OpenFlow rules are deployed in the switches to redirect the suspicious flow through the path provisioned for them. As we can see in Fig. 8, the legitimate traffic heading to the AEMC was thus able to quickly return to its normal level.

8.6. Network jitter of legitimate traffic

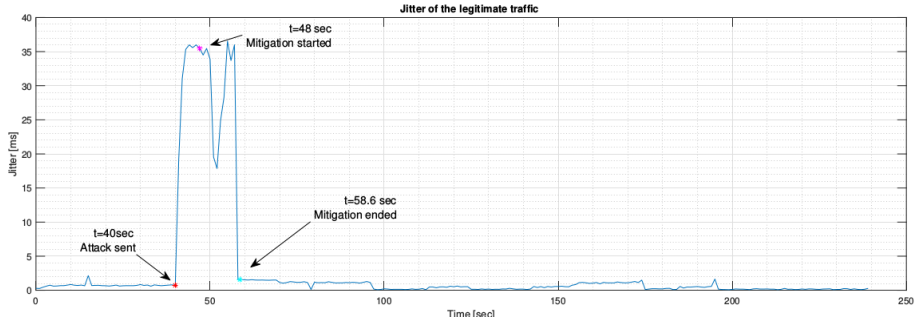

Figure 9: Network jitter of legitimate traffic

Finally, we also evaluated how the network jitter of legitimate traffic varies in the presence of DDoS traffic. As Fig. 9 shows, the network jitter of legitimate traffic going towards the AEMC started to increase when the attack traffic

from attacker machine (A) congested the network. However, all of them immediately decreased when the *mitigation engine* redirected the traffic flows upon receiving the security alert from the *detection engine*.

9. Limitation and Discussion

The design description and use case described in this paper present the following six characteristics which constitute contributions of this work.

First, the proposed architecture enables a dynamic and automated mitigation of attacks in the communication infrastructure of a ship, given that SDN controllers react on an event basis and dynamically configure the policies to handle the suspicious and malicious flows. The experimental results demonstrate that this framework is able to quickly translate high-level policies into low-level OpenFlow rules for deployment in the data plane devices while mitigating the attack traffic.

It should be noted, however, that the classification of the flows as suspicious or malicious is beyond the scope of this paper. We assume that the shipping company can deploy some detection mechanism to classify the flows as suspicious and malicious based on their criteria. In the framework, upon detection of an attack, the detection module can share the alert with the mitigation module.

Second, a multi-path routing approach provides failover in case of link failure or congestion. Thanks to the global visibility of the network achieved through the SDN controller, flow details, labels and low-level actions can be quickly modified for the concerned flow. Rules are pre-configured in the legitimate path to reduce the latency. In the path provisioned for the suspicious flow, rules are dynamically deployed in the data plane devices as and when it is required.

Third, policy language and translation mechanism introduced in the framework offers flexibility to express high-level policy without knowing low-level device specific syntax. As a result, crew members do not need to learn the

low-level syntax to express and enforce the policies manually. This reduces the burden on the crew members and of any errors from human-computer interaction.

Fourth, although the sample policy used in the experiment is mainly for attack mitigation, the framework can also be applied to express and translate varied set of network and security policies.

Fifth, the framework reduces the collateral damage caused to the legitimate traffic because of the suspicious traffic, by redirecting the suspicious traffic through lower bandwidth paths.

Sixth, the framework promotes the collaboration between the controllers managing different network devices and the critical components of the ship. For instance, `AEMC` can report the anomalous activities to the SDN controller running `Mitigation Engine` module to mitigate the attacks. `Detection Engine` deployed near the `AEMC` has better view of attacks, since it controls all the traffic reaching `AEMC`. This collaboration between the controllers helps to mitigate the attacks in the physical layer as well as in the cyber layer.

There are several limitations in our current framework, for example:

Policy conflict: It may be argued that policies can be ambiguous in cases where they shadow each other or completely deny one action to another. For example, there can be a case, where deployment of a forwarding policy violates the security policy. In this case, security policy should be given higher priority to forwarding policy.

Computing optimized path: In the framework, we have not considered computing the optimized paths at the time of redirecting suspicious traffic. Paths can be computed considering different parameters such as number of flows in the path, bandwidth of the link and middleboxes to be traversed in case it is required. Likewise, in case the topology changes in the network, then a topology discovery mechanism can be run to know the actual list of valid paths [54], which can offer flexibility and more dynamicity in the framework.

However, computing paths for every security alert can result in the processing overhead and create a bottleneck in the framework. A single instance of the

framework cannot be able to handle this overhead. Therefore, there is a need of some optimization methods to scale our framework. To solve this problem we can use some existing solutions such as running the multiple instances of the framework in a distributed manner.

Scalability of the framework: Scalability is a relevant matter for the deployment of this framework. For example, the ingress switch can cause a bottleneck in the framework because of limited TCAM entries. To address this issue scalable and efficient solutions such as DIFANE can be implemented [55]. Nowadays, software switches (OVS switches) are also widely used in the network to avoid the limitation of flow table entries of hardware switches [56]. The framework is implemented with a single SDN controller for mitigation of an attack. However, relying on a single SDN controller creates a vulnerable single point of failure in the framework. Enhancing the framework with multiple SDN controllers is therefore desirable but non-trivial, which needs to be addressed. Delay in controller-switch communication can also cause scalability issues. Yet, new SDN controllers can process millions of flows per second [56]. Open vSwitch is capable of installing tens of thousands of flow rules per second with sub milliseconds of latency [57].

Middlebox deployment: Currently, we have implemented the framework in an environment with multiple paths with varying number of switches for different types of traffic. This topology is appropriate when considering the communication topology of a ship. Furthermore, in this paper, we mainly focus on expressing the policies in a human-understandable language and translating these policies for enforcement in the network devices of the ship.

We use labels to route the traffic through the different paths and middleboxes in the network. However, chaining the middleboxes in the ship's communication network causes issues such as the order of chaining and routing conflicts for network devices. We plan to address these issues in future research.

Topological configuration: In addition to the wired topology on the ship, it is important to configure wireless topology between some devices and switches.

Wireless communication can be helpful to control the ship from anywhere on the deck. In the future, we will extend the framework considering wireless communications as well.

Security of SDN controller: Security of the SDN controller is also a major concern. As there are several works on protecting the SDN controller from different types of attacks [58, 59, 60, 61], we rely on these works for securing the SDN controller.

10. Conclusion and Future Work

In this paper, we presented a framework by leveraging the SDN paradigm to provide dynamic and automated mitigation of attack traffic in the communication network of ship. Our framework allows the network operator and crew members with little security expertise to express the network and security policies in a human readable language. The framework also offers a translation mechanism to translate the high-level policies into low-level OpenFlow rules for dynamic deployment into data plane devices. By doing so, it hides the low-level complexity of the underlying network to the crew member, who only need to focus on expressing the network and security policies. It is not required for the network operator and crew member to be present all the time to configure the network for mitigating the attacks. Another major advantage of the framework is that it allows different controllers to collaboratively manage the critical components of the ship and the underlying networking devices, which can provide efficient mitigation of threats. Moreover, our framework relies on multipath routing to increase the resilience against cyber attacks such as DDoS attacks. Our future work will be focused on resolving the conflicts of reaction policies due to simultaneous enforcements for different scenarios.

References

[1] Cyber-enabled ships:Deploying information and communications technology in shipping, Tech. rep., Lloyd Register (2016).

[2] H. R. Hayes, Maritime cybersecurity: the future of national security, Master's thesis, Naval Postgraduate School (2016).

[3] The Guidelines on Cyber Security Onboard Ships, Tech. rep., BIMCO (2017).

[4] Arbor Networks, Worldwide Infrastructure Security Report, Tech. rep., Arbor Networks (2016).

[5] S. Shin, P. A. Porras, V. Yegneswaran, M. W. Fong, G. Gu, M. Tyson, FRESCO: Modular Composable Security Services for Software-Defined Networks, in: Proceedings of the ISOC Network and Distributed System Security Symposium (NDSS), 2013.

[6] B. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, T. Turletti, A survey of software-defined networking: Past, present, and future of programmable networks, Communications Surveys Tutorials, IEEE 16 (3) (2014) 1617–1634.

[7] Y. Ben-Itzhak, K. Barabash, R. Cohen, A. Levin, E. Raichstein, Enforsdn: Network policies enforcement with sdn, in: 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), 2015, pp. 80–88. doi:10.1109/INM.2015.7140279.

[8] X. Dong, H. Lin, R. Tan, R. K. Iyer, Z. Kalbarczyk, Software-defined networking for smart grid resilience: Opportunities and challenges, in: Proceedings of the 1st ACM Workshop on Cyber-Physical System Security, CPSS '15, ACM, 2015, pp. 61–68. doi:10.1145/2732198.2732203.

[9] E. G. da Silva, L. A. D. Knob, J. A. Wickboldt, L. P. Gaspary, L. Z. Granville, A. Schaeffer-Filho, Capitalizing on sdn-based scada systems: An anti-eavesdropping case-study, in: 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), 2015, pp. 165–173. doi:10.1109/INM.2015.7140289.

[10] S. K. Fayaz, Y. Tobioka, V. Sekar, M. Bailey, Bohatei: Flexible and elastic ddos defense, in: 24th USENIX Security Symposium (USENIX Security 15), USENIX Association, Washington, D.C., 2015, pp. 817–832.

[11] R. Sahay, G. Blanc, Z. Zhang, H. Debar, Towards autonomic ddos mitigation using software defined networking, in: Proceedings of the NDSS Workshop on Security of Emerging Technologies (SENT), 2015.

[12] R. Sahay, D. A. Sepulveda, W. Meng, C. D. Jensen, M. B. Barfod, Cybership: An sdn-based autonomic attack mitigation framework for ship systems, in: F. Liu, S. Xu, M. Yung (Eds.), Science of Cyber Security, Springer International Publishing, Cham, 2018, pp. 191–198.

[13] R. Sahay, W. Meng, C. D. Jensen, The application of software defined networking on securing computer networks: A survey, Journal of Network and Computer Applications 131 (2019) 89 – 108. doi:https://doi.org/10.1016/j.jnca.2019.01.019.
URL http://www.sciencedirect.com/science/article/pii/S108480451930027X

[14] Final Report:Autonomous Engine Room, Tech. rep., MUNIN:Maritime Unmanned Navigation through Intelligence in Network (2015).

[15] Final Report:Autonomous Bridge, Tech. rep., MUNIN:Maritime Unmanned Navigation through Intelligence in Network (2015).

[16] Specification concept of the general technical system redesign, Tech. rep., MUNIN:Maritime Unmanned Navigation through Intelligence in Network (2013).

[17] K. RIJEI, S. Krile, D. Kezić, F. Dimc, Nmea communication standard for shipboard data architecture nmea komunikacijski standard za arhitekturu podataka na brodu, 2013.

[18] J. Y. J. Li-Der Chou, Network-Integrated Ship Automatic Systems and Internetworking to the Internet (1996).
URL http://jmst.ntou.edu.tw/marine/4/35-41.pdf

[19] M.-J. Kim, J.-W. Jang, Y.-s. Yu, Topology configuration for effective in-ship network construction, in: T.-h. Kim, H. Adeli, R. J. Robles, M. Balitanas (Eds.), Advanced Communication and Networking, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 380–392.

[20] D.-K. Jeon, Y. Lee, A ship area network with wimedia wireless gateway applying a cooperative transmission, 2014.

[21] S.-H. Chang, H. Mao-Sheng, A novel software-defined wireless network architecture to improve ship area network performance, The Journal of Supercomputing 73 (7) (2017) 3149–3160. doi:10.1007/s11227-016-1930-5. URL https://doi.org/10.1007/s11227-016-1930-5

[22] Guidelines on Maritime Cyber Risk Management, Tech. rep., IMO (2017).

[23] Cyber Security in the Shipping Industry, Tech. rep., Deloitte (2017).

[24] R. Bensing, An assessment of vulnerabilities for shipbased control systems, Master's thesis, Naval Postgraduate School (2009).

[25] Christopher R. Hayes, Maritime Cybersecurity: The Future of National Security, Tech. rep., Naval Postgraduate School (2016).

[26] G. L. Babineau, R. A. Jones, B. Horowitz, A system-aware cyber security method for shipboard control systems with a method described to evaluate cyber security solutions, in: 2012 IEEE Conference on Technologies for Homeland Security (HST), 2012, pp. 99–104. doi:10.1109/THS.2012.6459832.

[27] Cyber threat to ships – real but manageable, Tech. rep., ABB (2014).

[28] L. Yunfei, C. Yuanbao, W. Xuan, L. Xuan, Z. Qi, A framework of cybersecurity protection for warship systems, in: 2015 Sixth International Conference on Intelligent Systems Design and Engineering Applications (ISDEA), 2015, pp. 17–20. doi:10.1109/ISDEA.2015.14.

[29] C. Yuanbao, H. Shuang, L. Yunfei, Intrusion tolerant control for warship systems, in: 4th International Conference on Computer, Mechatronics, Control and Electronic Engineering (ICCMCEE 2015), 2015, pp. 165–170. doi:10.2991/iccmcee-15.2015.31.

[30] E. Penera, D. Chasaki, Packet scheduling attacks on shipboard networked control systems, in: 2015 Resilience Week (RWS), 2015, pp. 1–6. doi:10.1109/RWEEK.2015.7287421.

[31] W. Wu, Ship communication network intrusion signal identification based on hidden markov model, Journal of Coastal Research (2018) 868–871.

[32] B. Xing, Y. Jiang, Y. Liu, S. Cao, Risk data analysis based anomaly detection of ship information system, Energies 11 (12).

[33] Open Networking Foundation, SDN Security Considerations in the Data Center, Tech. rep., ONF (2013).

[34] A. Voellmy, H. Kim, N. Feamster, Procera: A language for high-level reactive network control, in: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, HotSDN '12, ACM, New York, NY, USA, 2012, pp. 43–48. doi:10.1145/2342441.2342451.
URL http://doi.acm.org/10.1145/2342441.2342451

[35] R. Soulé, S. Basu, R. Kleinberg, E. G. Sirer, N. Foster, Managing the network with merlin, in: Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks, HotNets-XII, ACM, New York, NY, USA, 2013, pp. 24:1–24:7. doi:10.1145/2535771.2535792.
URL http://doi.acm.org/10.1145/2535771.2535792

[36] T. Kuhn, A survey and classification of controlled natural languages, Comput. Linguist. 40 (1) (2014) 121–170.

[37] W. Han, C. Lei, A survey on policy languages in network and security management, Computer Networks 56 (1) (2012) 477 – 489.

[38] S. Scott-Hayward, G. O'Callaghan, S. Sezer, Sdn security: A survey, in: 2013 IEEE SDN for Future Networks and Services (SDN4FNS), 2013, pp. 1–7.

[39] I. Alsmadi, D. Xu, Security of software defined networks: Asurvey, Computers & Security 53 (2015) 79 – 108.

[40] S. A. Shah, J. Faiz, M. Farooq, A. Shafi, S. A. Mehdi, An architectural evaluation of sdn controllers, in: 2013 IEEE International Conference on Communications (ICC), 2013, pp. 3504–3508. doi:10.1109/ICC.2013.6655093.

[41] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, S. Uhlig, Software-defined networking: A comprehensive survey, Proceedings of the IEEE 103 (1) (2015) 14–76. doi:10.1109/JPROC.2014.2371999.

[42] R. Braga, E. Mota, A. Passito, Lightweight ddos flooding attack detection using nox/openflow, in: IEEE Local Computer Network Conference, 2010, pp. 408–415. doi:10.1109/LCN.2010.5735752.

[43] A. Mahimkar, J. Dange, V. Shmatikov, H. Vin, Y. Zhang, dfence: Transparent network-based denial of service mitigation, in: 4th USENIX Symposium on Networked Systems Design & Implementation (NSDI 07), USENIX Association, Cambridge, MA, 2007.

[44] R. Kowalski, M. Sergot, A logic-based calculus of events, New Gen. Comput. 4 (1) (1986) 67–95.
URL http://dx.doi.org/10.1007/BF03037383

[45] Open Networking Foundation, OpenFlow Switch Specification Version 1.4.0, Tech. rep., ONF (2013).

[46] P. Quinn, U. Elzur, Network Service Header, Internet-Draft draft-ietf-sfc-nsh-05, Internet Engineering Task Force, work in Progress (May 2016).
URL https://tools.ietf.org/html/draft-ietf-sfc-nsh-05

[47] S. Homma, D. Lopez, D. Dolson, A. Gorbunov, N. Leymann, K. Naito, M. Stiemerling, P. Bottorff, don.fedyk@hpe.com, Analysis on Forwarding Methods for Service Chaining, Internet-Draft draft-homma-sfc-forwarding-methods-analysis-05, Internet Engineering Task Force, work in Progress (Aug. 2016).
URL https://tools.ietf.org/html/draft-homma-sfc-forwarding-methods-analysis-05

[48] A. Mahimkar, J. Dange, V. Shmatikov, H. Vin, Y. Zhang, dFence: Transparent Network-based Denial of Service Mitigation, in: Proceedings of the 4th USENIX Conference on Networked Systems Design Implementation (NSDI), USENIX Association, Berkeley, CA, USA, 2007, pp. 24–24.

[49] B. Feinstein, D. Curry, H. Debar, The Intrusion Detection Message Exchange Format (IDMEF), RFC 4765 (Oct. 2015). doi:10.17487/rfc4765.
URL https://rfc-editor.org/rfc/rfc4765.txt

[50] T. Takahashi, K. Landfield, Y. Kadobayashi, An Incident Object Description Exchange Format (IODEF) Extension for Structured Cybersecurity Information, RFC 7203 (Apr. 2014). doi:10.17487/RFC7203.
URL https://rfc-editor.org/rfc/rfc7203.txt

[51] F. Q. Ajay Tirumala, J. Jon, Kevin, IPERF (2003).
URL https://web.archive.org/web/20081012013349/http://dast.nlanr.net/Projects/Iperf/

[52] B. Lantz, B. Heller, N. McKeown, A network in a laptop: Rapid prototyping for software-defined networks, in: Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, Hotnets-IX, ACM, New York, NY, USA, 2010, pp. 19:1–19:6. doi:10.1145/1868447.1868466.
URL http://doi.acm.org/10.1145/1868447.1868466

[53] R. Braga, E. Mota, A. Passito, Lightweight DDoS flooding attack detection using NOX/OpenFlow, in: 35th IEEE Conference on Local Computer Networks (LCN), 2010, pp. 408–415. doi:10.1109/LCN.2010.5735752.

[54] F. Pakzad, M. Portmann, W. L. Tan, J. Indulska, Efficient topology discovery in software defined networks, in: 2014 8th International Conference on Signal Processing and Communication Systems (ICSPCS), 2014, pp. 1–8. doi:10.1109/ICSPCS.2014.7021050.

[55] M. Yu, J. Rexford, M. J. Freedman, J. Wang, Scalable flow-based networking with difane, SIGCOMM Comput. Commun. Rev. 41 (4) (2010) –.
URL http://dl.acm.org/citation.cfm?id=2043164.1851224

[56] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, S. Uhlig, Software-defined networking: A comprehensive survey, Proceedings of the IEEE 103 (1) (2015) 14–76.

[57] S. H. Yeganeh, A. Tootoonchian, Y. Ganjali, On scalability of software-defined networking, IEEE Communications Magazine 51 (2) (2013) 136–141.

[58] S. Lee, C. Yoon, C. Lee, S. Shin, V. Yegneswaran, P. A. Porras, Delta: A security assessment framework for software-defined networks, in: NDSS, 2017.

[59] S. Shin, Y. Song, T. Lee, S. Lee, J. Chung, P. Porras, V. Yegneswaran, J. Noh, B. B. Kang, Rosemary: A robust, secure, and high-performance network operating system, in: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, CCS '14, ACM, New York, NY, USA, 2014, pp. 78–89. doi:10.1145/2660267.2660353.
URL http://doi.acm.org/10.1145/2660267.2660353

[60] I. Alsmadi, D. Xu, Security of software defined networks: A survey, Computers Security 53 (2015) 79 – 108. doi:https://doi.org/10.1016/j.cose.2015.05.006.
URL http://www.sciencedirect.com/science/article/pii/S016740481500070X

[61] S. K. Fayazbakhsh, V. Sekar, M. Yu, J. C. Mogul, Flowtags: Enforcing network-wide policies in the presence of dynamic middlebox actions, in:

Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking, HotSDN '13, ACM, New York, NY, USA, 2013, pp. 19–24. doi:10.1145/2491185.2491203.
URL http://doi.acm.org/10.1145/2491185.2491203