Future Generation Computer Systems 100 (2019) 736–750

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

CyberShip-IoT: A dynamic and adaptive SDN-based security policy enforcement framework for ships

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

[a] Department of Applied mathematics and Computer Science, Technical University of Denmark, DK-2800 kgs., Lyngby, Denmark
[b] Department of Management Engineering, Technical University of Denmark, DK-2800 kgs., Lyngby, Denmark

HIGHLIGHTS

- We design an SDN-based framework to protect ship's communication infrastructure.
- Our framework offers a high-level policy language and a translation mechanism.
- We developed a use case and evaluated different performance metrics.

ARTICLE INFO

Article history:
Received 8 February 2019
Received in revised form 20 April 2019
Accepted 15 May 2019
Available online 22 May 2019

Keywords:
Internet-of-Things
OpenFlow
SDN
Ship system
Policy language and enforcement
SCADA system

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 complexity 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.

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 managing smart cities, e.g., traffic management and congestion controls, critical infrastructure, such as the power grid, and more recently the transportation sector, such as the shipping industry. IoT provides new opportunities to leverage the interconnections between the physical and cyber layers in ship technology, where all the ships' components, such as global navigation satellite system (GNSS), Automatic Identification Systems (AIS), Electronic Chart Display Systems (ECDIS) are

* Corresponding author.
E-mail addresses: risa@dtu.dk (R. Sahay), weme@dtu.dk (W. Meng), dasep@dtu.dk (D.A.S. Estay), cdje@dtu.dk (C.D. Jensen), mbba@dtu.dk (M.B. Barfod).

https://doi.org/10.1016/j.future.2019.05.049
0167-739X/© 2019 Elsevier B.V. All rights reserved.

integrated with the cyber systems. With the use of operational technology (OT), control systems on the ship have evolved similarly 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 through the onboard communication network to control and manage the ship, but the ship's communication network is also permanently or frequently connected to onshore communication infrastructures, e.g. through satellite links. This advance in ship technology enhances the monitoring and control capabilities, both on board and from shore. However, the use of ICT and IoT systems as part of the ship's technology also introduces new risks related to cyber attacks, faults and failures. The shipping industry has become more vulnerable to cyber attacks because of its dependence on the ICT technology [1].

Cyber attacks may either target the ship systems, such as GNSS, AIS and ECDIS that are vulnerable to Denial of Service, jamming, spoofing and malware attack respectively [2,3], but they may also target the ship's communication infrastructure, e.g., the ship's network switches are also vulnerable to Distributed Denial of Service (DDoS) attacks. A DDoS attack on the ship's communication network could result in the inability to control the engine system, bridge system, and alarm system, endangering the crew and ship. Such DDoS attacks are frequently used against other types of network [4].

Detection and mitigation of network attacks in the traditional communication networks used in ships require the deployment of special purpose software and hardware devices, as well as the modification of legacy routers and switches, incurring more complex human interventions. Moreover, it requires crew members to become experts in network and security administration, in order 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 to mitigate cyber attacks within ships in an automated way. We propose to utilize the capabilities of Software-Defined Networking (SDN) to design a framework for mitigating these attacks in an automated way. In particular, the decoupling of the control and data forwarding plane in SDN, and the configuration of data plane devices from a centralized controller through programmable interface allows us to achieve automation in mitigating attacks.

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 necessary flexibility to simplify network operation compared to traditional network management techniques, because it allows expression of policies at the controller, which can then be enforced into network devices depending on the status of the network [7]. In SDN, network switches behave as simple forwarding devices, whose forwarding rules can be dynamically configured by a centralized controller. Several studies advocate SDN to simplify the network management tasks for improving the resilience and security in the smart grid and enterprise environment [8–13].

In this paper, we therefore present a framework based on SDN to defend the ship's communication infrastructure against cyber attacks. The traditional communication network of a ship can be changed into an SDN environment incrementally. We focus on mitigating attacks rather than detection, so we assume that there are adequate mechanisms available for detecting attacks, e.g., intrusion detection systems (IDSs) [14–16]. Our framework offers a high-level policy language and a translation mechanism to translate these 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 incurs low computation overhead.

It is worth noting that our work mainly focus on how to realize a dynamic and automatic mechanism to mitigate attacks, while the protection of SDN is out of the scope. Some related work on securing SDNs can refer to [17,18].

The rest of the paper is organized as follows. Section 2 reviews some related work. 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. Section 7 presents a use case showing the applicability of the framework. Experimental results are discussed in Section 8. Section 9 provides limitations and discussion about the framework. Finally, Section 10 concludes the paper.

2. Background

In this section, we firstly present the critical components of ship systems and discuss some relevant studies on building communication networks on the vessels in connection with the critical components. We then survey related work on protecting the communication infrastructure of ships. We further introduce some related studies on policy language and show how our proposed 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

In this part, we describe the general ship systems and the communication infrastructure. Fig. 1 depicts the Integrated bridge controller (IBC) and the Autonomous Engine Monitoring controller (AEMC), which can help manage and control different bridge and engine components on the ship.

- **Autonomous Engine Monitoring Control (AEMC):** It manages the ship systems including propulsion, propeller, cargo, and so on [19]. Depending on specific scenarios, it issues the control command to start, stop, increase, or decrease the speed, and reroute the ship. It can periodically analyze the data received from the sensors of propulsion, cargo and other components managed by the engine, with the purpose of checking the status of the devices, i.e., whether they are working properly or not.
- **Integrated Bridge Controller (IBC):** It supervises the functioning of different bridge components such as a electronic chart display system, radar, echo sounding device, and automatic identification system [3]. It can receive the data from the sensors of these devices and provide a centralized display to the crew on-board to access the data. Also, it issues control commands to `AEMC` in order to start or stop the propulsion control system, and reroute the ship to different directions depending on the information from the bridge devices.

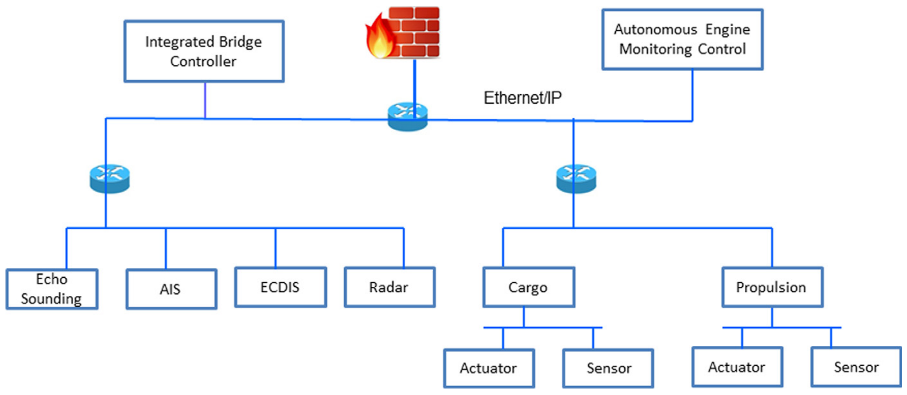

Fig. 1. Ship systems.

- **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 [20].
- **Electronic Chart Display System (ECDIS):** It contains the pre-planned traffic routes, prohibited areas, next port of call, etc. Generally, it is maintained by a computer that is managed by captain.
- **Radar:** It helps detect and monitor objects in the deep ocean to avoid collision.
- **Echo sounding:** It provides related information regarding the water level to help navigate and maintain the ship's balance. Depending on the information from the `Echo sounding` device, `AEMC` can either reduce the water level in the ballast, or fill the ballast tank.
- **Cargo:** It contains the shipment tracking details in the cargo [3]. Depending on the ship's design, it can be either under the control of `AEMC`, or can be separate [19].
- **Propulsion:** It is controlled by `AEMC` and is coupled with the propeller [21]. It is responsible for increasing or decreasing the speed of the ship.

The critical ship components are interconnected through a ship area network. The traditional ship area network is mainly built on wired networks, which applies the National Marine Electronics Association (NMEA) 2000 and shipboard control network (Ethernet based MiTS) [22]. The lowest level devices (e.g., AIS, radar) are connected with a switch or gateway through NMEA protocol. Integrated bridge and engine controllers are connected through an ethernet-based connection.

To effectively transfer data on the ship, Chou et al. [23] 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. [24] designed a network by using ship area network (SAN) to connect different network types (wired and wireless) on the ship. Their experimental results indicate that the deployment of SAN LAN with the STAR topology can improve the communication services.

There have been some studies investigating wireless connection between different network types on the ship. Jeon et al. [25] proposed a wireless communication technique between the shipboard control network and the low-level device network to achieve high data throughput. Chang et al. [26] designed a software-defined wireless networking architecture for high performance in the ship area network. They combined self-organizing time division multiple access (SOTDMA) and software defined wireless networking (SDWN) to reduce end-to-end transmission delay to achieve high performance in ship area networking.

The above studies aimed to build different networks to connect critical components for an effective data transfer. Differently, our goal is to design a framework to achieve automated mitigation using the SDN technology. In particular, we connect both IBC and AEMC through an SDN environment to achieve a dynamic and automated configuration in the network.

2.2. Protection mechanisms for ship systems

With the widespread adoption of ICT, more researchers are aware of security properties of ship systems, i.e., how security breaches within these technologies can compromise the ship's operations and its crew members. However, building ship security mechanisms is still in an early stage, while most research work has mainly focused on identifying potential threats and vulnerabilities [3,27,28]. In particular, BIMCO guidelines [3] draw special attention to different types of cyber attacks that may affect the ship security in the critical components. Bensing [29] performed an assessment to identify vulnerabilities in the ship's control system, by examining the critical infrastructure on a shipboard system. Similarly, Hayes [30] identified some recent cyber attacks and incidents, and provided some recommendations to protect ships.

To the best of our knowledge, limited research work focuses on how to protect the communication infrastructure of a ship against cyber attacks. Babineau et al. [31] proposed an approach to periodically divert the traffic between different switches in the network to protect the critical components, which relies on the redundancy in the design of the ship's communication network to divert the traffic through different paths. A leading company called ABB in industrial automation, proposed to place the critical components in the core of a ship's network [32]. Yunfei et al. [33] and Chen et al. [34] proposed a similar approach to protect a warship system from cyber attacks by deploying access controls, firewalls and intrusion detection systems (IDSs) to mitigate the attacks. Moreover, Penera et al. [35] identified a type of packet scheduling attack on the shipboard network controlled system. A network intrusion detection mechanism based on hidden markov model was presented for communication network of a ship by Wu et al. [36]. Xing et al. [37] proposed an IDS based on collaborative controlling structure to detect signal attacks between different components of the ship.

All the above 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, improving the resilience of the ship control system, and reducing the burden on network operators and crew members, when configuring policies in the network devices. The traditional ship's communication network does not offer the flexibility of dynamic and automated configuration to mitigate attacks. That is, crew members have to configure the network devices manually to deploy rules for attack mitigation. To mitigate the issue, our SDN-based framework proposes a policy language and a translation mechanism to define high-level policies and translate them automatically into network devices.

2.3. Policy language

Our policy language and translation mechanism leverages the benefits of SDN [38], i.e., deploying the rules in the network devices via a centralized control point and an interface between the control and the data plane. It helps the crew members and network administrators to express the policies in human understandable language for better managing the communication network of a ship. In this case, crew members can define the high-level policies to understand syntax without delving into low-level details.

There are some high-level policy languages proposed for SDN networks. Table 1 compares our policy language with Procera [39] and Merlin [40]. In particular, Procera and Merlin are both high-level policy languages developed specifically to express policies for SDNs.

Procera uses a complex syntax that is difficult to understand. It does not make use of controlled natural languages (CNL) [41], which can express the high-level policies in a human understandable format. Thus, network administrators do not need to learn device-specific syntax. Procera is also an event-based language, but needs to be registered with its global data structure. It does not provide an unique policy ID to retrieve the policy. A unique index can help retrieve the policies efficiently in a large database containing policies [42].

Merlin [40] enables defining the high-level policies via a declarative language. The main focus of Merlin is resource provisioning in the network to specific flows. It uses the packet header fields to classify and express the policies for a set of packets. A unique identifier is not specified for policy retrieval. Merlin policies are based on the condition–action paradigm, where conditions are continuously checked to trigger a policy in condition–action paradigm.

In contrast, 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 based on our high-level language. In addition, we use a unique ID to efficiently retrieve the policies from the repository.

3. Challenges and motivations

The main goal of our framework is to enable dynamic and automated attack mitigation in a ship, in order to protect its critical components. To achieve this goal, the following designing requirements should be satisfied.

- **Simple and expressive.** The framework should provide a high-level language to express the policies in a controlled natural language [41] without knowing the implementation details. This makes the high-level policy understandable by a human. 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., specifying the network and security requirements according to the context. For example, when the context of a flow changes from legitimate to malicious, the administrator should block the flow at once.
- **Dynamic and automated.** The manual policy enforcement often causes delay by requiring human interference. This issue would be even severe when mitigating attacks for ship systems, as crew members are not IT-expert. Therefore, the framework should be dynamic and highly intelligent, i.e., configuring the forwarding equipments.
- **Service chaining.** The framework should allow network administrators to specify service function chains within its network, i.e., the processing order of flows through a series of service functions or middleboxes. This helps deploy in-depth defense and specify the processing order of network traffic to protect critical components of ship systems. For instance, a network administrator can define an order that a packet coming from the offshore control center should be first processed through a firewall before traversing the core network.
- **Resilient.** Even under attacks, there is a need to ensure the availability of different components of ship systems, i.e., components can communicate with each other. That is, the framework should has the capability to recover from any attack quickly, i.e., reducing the communication delay.
- **Easy deployment.** The framework should be independent without the need of massive security devices for protecting the communication infrastructure and critical components on the ship. A complex deployment is very likely to increase the implementation and management difficulty in practice.

4. SDN enabled CyberShip architecture

In this section, we propose an SDN-based cyber ship framework to protect the ship's communication infrastructure against cyber attacks. In particular, SDN can provide many merits as below.

1. **Centralized controller.** In SDN, the control plane is decoupled from the data plane and a centralized controller is responsible for handling the network, i.e., configuring the data-plane devices according to the policies. In addition, security and network policies can be expressed at the controller, which can be deployed automatically. 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.
2. **Flexibility.** SDN can make the configuration flexible in the ship, i.e., upgrading network applications more easier for ship systems.
3. **Standard API.** The OpenFlow protocol provides a standard API to manage the distributed network equipment. This can facilitate the configuration of network devices according to the requirements from the centralized location.
4. **Programmability.** SDN allows creating a range of customized applications. In cyber ships, this feature can be useful to define the high-level policy language, which can be translated and deployed from the centralized controller. In addition, it facilitates the deployment of security rules in a dynamic way.
5. **Global visibility.** SDN controller can have a global view of the network. This can simplify many network management tasks, and allow administrators to configure data-plane devices from a centralized controller [43]. This can help monitor the ship network more effectively, i.e., configuring the network devices to divert the traffic in case of congestion or if a device is not working properly.

Table 1
Comparison with existing policy languages.

	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

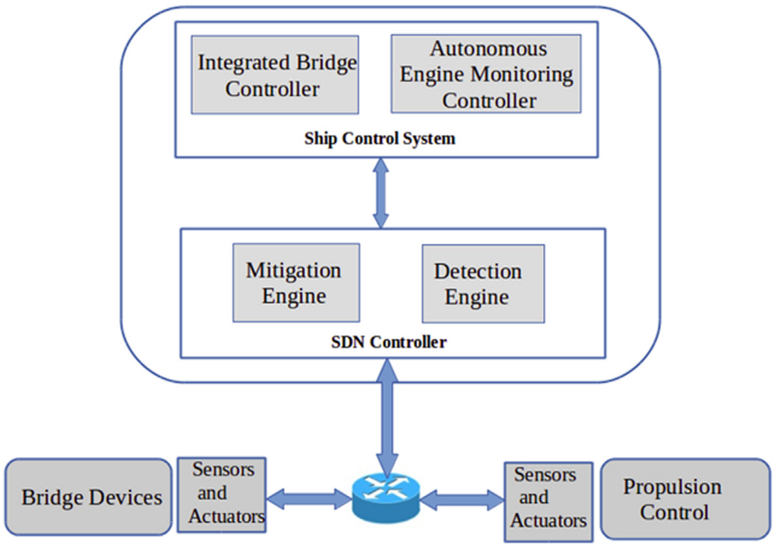

Fig. 2. Framework of the CyberShip.

4.1. *Components of the framework*

In this section, we introduce the major components of our framework, including physical- and data-plane components and the control-plane components. The major components are shown in Fig. 2. Next, we first describe the physical and data plane components.

1. **Sensors and actuators.** Sensors and actuators are attached to different physical components on the ship relating to the bridge, engine and propulsion control devices. These sensors can forward relevant data to the `Integrated Bridge Controller` and the `Autonomous Engine Monitoring Controller` for analysis.
2. **SDN switches.** These are programmable switches at the data plane, which can be dynamically configured by the controller or centralized applications via the southbound API.

Next, we describe the control-plane components of our framework.

1. **SDN controller.** It is a software platform deployed as a remote entity, which can provide network abstractions for handling the network [44]. It can provide the centralized intelligence and global network visibility to control and manage the network. It also offers an interface that makes it possible to deploy the rules in the switches through a centralized location.
2. **Detection engine.** The framework employs a detection engine to identify suspicious and malicious activities by examining the network traffic. Due to the modular design of our framework, network administrators can deploy the detection mechanism in a flexible way. Network operators can deploy various mechanisms to check and identify malicious flows [45,46]. Upon detection of any suspicious events, the detection engine can notify security administrators accordingly.
3. **Mitigation engine.** This engine is responsible for mitigating any attacks in the cyber ship environment. It contains a repository consisting of many security and network policies defined in high-level language. Details about the high-level policy is given in Section 5. It also maintains a table containing the location of middleboxes (like firewall and IDSs) in the network, i.e., switch and port through which the devices are connected (details can refer to Section 7). Further, it maintains a list of network paths with switch ID and output port along with its label. A label can be inserted in the VLAN ID field or as an MPLS header in the flow. Labels can help fast forward the traffic because the core switches only check the label for forwarding.
4. **Autonomous Engine Monitoring Controller (AEMC).** It manages the propulsion control, main engine, propeller devices of the ship [19]. In the architecture, it is located at the SDN controller as shown in Fig. 2, while it can be deployed on separate SDN controller. Upon detecting attack or fault in the engine component, it can inform the `Mitigation Engine` to redirect the traffic via another route.
5. **Integrated Bridge Controller.** It offers a centralized platform to transfer the information from different bridge devices. Upon detecting the fault, failure or attack, it notifies

the Mitigation Engine to divert the network traffic via another route or to the middleboxes for further processing. Similar to AEMC, it can be deployed on separate SDN controller.

5. Security policy specification

In this section, we describe how to express the high-level policies in the mitigation engine module. These policies can be translated into low-level OpenFlow rules in the SDN switches. The automation eliminates the need of manual configuration for policy enforcement. One major goal of our policy language is to enable crew members writing simple network and security policies in order to manage the ship network.

5.1. Grammar of high-level policy

Listing 1: Grammar of High-level policy language

```
<Policy>=<PolicyID><Target><Rules>
<Target>=<DeviceID><IP_Address>
<Rules>=Set(<Rule>)
<Rule>=<Event><Conditions><Action>
<Event>=<Attack_Type>|<Fault_Type>
<Conditions>=<Condition>[<Connective><Conditions>]
<Condition>=<Field_Name><Comparison_Operator><Value>
<Connective>=And|Or
<Comparison_Operator>=less than|equal to|greater than|
       Not
<Action>=Drop|Redirect|Forward
```

The high-level policy syntax provides necessary guidelines to define the abstract policy. To achieve this, we need to define a grammar and relevant formats. It enables both the network operator and crew members with little IT background to express the security policies in an easily understandable language without knowing the implementation details.

In the framework, we use Event–Condition–Action (ECA) paradigm for policy specification [47]. The reasons for choosing ECA is two-fold: (1) it allows to specify a wide variety of events that can trigger particular actions; (2) it enables to define an asynchronous notification mechanism to response to these events. Our policy grammar, as 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 via the northbound API and the SDN controller.

In particular, our policy is composed of 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 like attack and fault types. It is not limited to only these events, other types of events can also be defined using our policy language. A network operator thus only needs to specify the corresponding conditions to define new events. Table 2 shows a list of events and matching items at the low-level OpenFlow rule for deploying in the SDN/OpenFlow switches. For instance, UDP and TCP flood are specified by Source and Destination IP addresses.

When an event is triggered, the associated parameters should be checked against the conditions specified in the policy. A *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 as shown in Listing 1, *condition* is specified with a field name and a value. Field name contains the condition name while the value represents the parameters specified in the condition. Table 2 shows a list of conditions and matching items at the low-level OpenFlow rule. Network operators can include other information for a condition. At the low-level rule specification, conditions are represented by Destination IP, Input Port, VLAN ID, and IP protocol.

As shown in Table 2, a condition can contain flow class, impact severity, traffic types and component status in our policy language. More specifically, *flow class* categorizes the flow into three different classes as: suspicious, malicious and legitimate. 'Suspicious' indicates that the flow is a combination of legitimate and malicious flow. The flow which is confirmed to contain malicious traffic can be regarded as malicious flow. The benign traffic is defined as legitimate flow. 'Traffic type' specifies different types of flows based on their protocols such as UDP traffic, HTTP traffic, TCP, etc. 'Impact severity' specifies the impact of malicious traffic, and the values are set as low, medium and high. 'Component status' indicates whether a critical component on the ship is working or not.

In addition, 'action' specifies the high-level decision that should be enforced when the conditions are met for the event. In Listing 1, we define three types of actions, while OpenFlow mainly specifies two actions: Forward and Drop [48]. In Table 2, we list some high-level OpenFlow actions under our framework. High-level action Drop is enforced when it is confirmed by the Detection Engine or controllers. Redirect action is enforced when there is suspicious flow confirmed by the Detection engine or a component is not working. Suspicious traffic is not directly blocked, but should be diverted through another path with low-bandwidth in order to reduce the collateral damage. This is because directly blocking the suspicious traffic would also cause management complexity. Forward action is enforced for the legitimate traffic. Fwd_Middlebox specifies how to route the traffic towards middleboxes like firewall and IDS.

5.2. Examples

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

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

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 different resources in the network. For example, passengers' entertainment network should be separate from the network of critical components like the engine, since it can affect the safety and security of ship systems. The policy below indicates that the traffic originating from a source and targeting at certain destination should be dropped.

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

Quality of Service (QoS). This can be measured based on the flow class and the flow source. For instance, legitimate flow originating from integrated bridge controller can be allocated with a high bandwidth path.

Flow: 154.70.56.1; Flow Class: Legitimate;
Action: high_bandwidth_path

Service Chaining. It protects a system by chaining different security middleboxes such as firewall, IDS, etc. The following policy specifies that the traffic arriving at the engine controller should traverse through a set of middleboxes.

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

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

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

In Summary, our policy language allows crew members to specify different network and security policies. The high-level policy is translated into low-level rule to be enforced in the network devices.

5.3. Semantics of low-level rules

A policy with the high-level language must be translated into low-level rules in order to be enforced in the network. High-level policies are translated into common format rules first, and then are translated into device specific configuration. It is very important to avoid ambiguous rules. In this section, we present a formal way to translate the high-level policy into low-level rules.

In our framework, a packet-type includes a list of field names in a packet along with their types. For example, the packet-type can be defined by (srcip : IPADDR, dstip : IPADDR, protocol : proto). A concrete packet specifies a value corresponding to each field of the 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 a pair (pf, a), where `pf` is a symbolic packet field and `a` is an action. With this notation, a rule for dropping a malicious flow can be specified as: (srcip=ip1, dstip=ip2, protocol=udp, action=drop).

Similarly, a rule for forwarding 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 items have only concrete values. It is obtained by replacing each variable and wildcard with a potential value of corresponding type. To evaluate a flow condition, a flow test can be performed, which is of type $CT(f1, \dots, fn) = c$, where fn is a packet field and all fn's are different. In our example, $CT(srcip) = x$ and $CT(dstip = y)$ are a test with the field *srcip* and *dstip*, respectively. Listing 2 shows the format of low-level rules.

Listing 2: Format of low-level rules

```
<Rules>=Set(<Rule>)
<Rule>=<Packet_header> "—>" <Action>
<Packet_header>=<srcip>":"<dstip>":"<port>":"<protocol>"
      :"<vlan_id>
<Action>=<Output_Port>|Drop
```

Depending on the high-level action, a low-level rule can be 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 deployed.

5.4. OpenFlow rule templates

OpenFlow rule templates provide a way 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 could be specified in the data-plane devices for different types of actions. It also provides an additional level of modularity to specify the parameters for translating high-level actions into low-level OpenFlow rules. OpenFlow rule templates are maintained in the *mitigation engine* module.

5.4.1. Template for specifying the path

As mentioned earlier, in our framework, a path can be specified as a set of *switch IDs* and *output ports*. The Listing 3 shows a template to define a path 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 is maintained in the *mitigation engine*.

Listing 3: A template for specifying the path

```
Path={Switch:'SwitchID','Port':Output_Port,..., Switch:'
      SwitchID','Port':Output_Port}
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. 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. Generally, 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. 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

```
Ingress Switch: ['Source_IP': addr, 'Destination_IP':
      addr, 'SetLabel':Vlan_ID,'Action':Out_Port]
Core Switch: ['Action':Out_Port,'Label':Vlan_ID]
Egress Switch:['Destination_IP': addr, 'Label':pop,'
      Action':Out_Port]
All the flows originating at a source: ['Source_IP':
      addr, 'SetLabel':Vlan_ID,'Action':Out_Port]
All the flows traversing to a destination: ['
      Destination_IP': addr, 'SetLabel':Vlan_ID,'Action':
      Out_Port]
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 presents an abstract template to specify the action `drop` depending on the condition in the network. For instance, we have to specify the source IP, destination IP and protocol to drop a specific flow. By contrast, dropping all flows originating from a particular source only requires to specify the source IP and the action `drop`. Similarly, we only need destination IP address and action `drop` to filter all the flows traversing to a particular destination. Blocking rule is generally specified at the ingress switch without causing collateral damage to legitimate traffic in the core network.

Listing 5: OpenFlow rule template to Drop the Flows

```
Drop specific flows: ['Source_IP': addr, 'Destination_IP
    ': addr, 'Protocol':proto,'Action':Drop]
All the flows originating at a particular source: ['
    Source_IP': addr, 'Action':Drop]
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 in case of link failure or congestion in the current path of a flow. Listing 6 describes a template to specify action `redirect` at the switches in the network. To redirect specific flows, OpenFlow rules can be specified with: source IP, destination IP address, modifying a label and forwarding the flow through an output port at the ingress switch. To redirect all the flows originating from a particular source, we only need to specify the source IP address of a flow and set a new label in the `VLAN ID` field at the ingress switch. 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, as well as specifying the output port at the ingress switch.

Listing 6: OpenFlow rule template to redirect the flow

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

Algorithm 1 describes how to deploy policies using the previous templates. It can takes the following inputs:

1. The high-level action can be one of followings: `forward`, `redirect`, and `drop`.
2. The path contains a set of switchIDs and relevant output ports.
3. Flow information consists of source IP, destination IP, and protocol type.
4. Network Service Header (NSH) mainly contains VlanID that is used to identify the path.

Based on these inputs, Algorithm 1 composes the rules based on relevant OpenFlow rule templates. In the end, our algorithm deploys the rules at the switches for processing the traffic. The rules can be deployed in the core switches first, and can be then deployed at the entry switches. This helps reduce packet loss, especially when the flow is redirected through another path. The detailed information of NSH can refer to Section 6.

Algorithm 1 *OpenFlow rule deployment*

```
1:  procedure POLICY_DEPLOYMENT(Action,path,flow,NSH)
2:      $path \leftarrow [SwitchID : outpput\_port, ..., SwitchID : output\_port]$
3:      $Action \leftarrow (forward, redirect, drop)$
4:      $A \leftarrow Action$
5:      $NSH \leftarrow VlanID$
6:      $flow \leftarrow (sourceIP, destinationIP, protocol)$
7:      $forward \leftarrow template\_forward[forward, path, flow, NSH]$
8:      $drop \leftarrow template\_drop[flow, drop]$
9:      $redirect \leftarrow template\_redirect[redirect, path, flow, NSH]$
10:     for Action A in concrete rules do
11:         $rule \leftarrow (A.flow, A.NSH, A.output\_port)$ // Compose the rule with the information
12:     end for
13:     $Deploy\_rule(rule)$
14: end procedure
```

6. Network service header (NSH)

Our framework uses the Network Service Header (NSH) [49] to steer traffic, including an end-to-end path identifier (global segment) and a middlebox or switch identifier (local segment). It helps in specifying the processing order of traffic by performing service chaining through a set of middleboxes or switches in the network. In particular, local segment can avoid unnecessary traffic processing through middleboxes, i.e., reducing the processing overhead on the middleboxes as well as the delay caused by middlebox processing. We use VLAN ID (MPLS and VXLAN header can be used as well) to provide NSH with local and global segments.

1. **Global segment** helps to steer the flows from an 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 forwarded. Flow states with NSH are maintained only at the ingress switch. This can help significantly reduce the flow table entries in the core switches [50].
2. **Local segment** is used to identify the middlebox through which the traffic of interest is processed. It avoids broadcasting the traffic to be processed by the middleboxes [51]. It is worth noting that middleboxes can be deployed inside the main data-path in the network. It helps reduce the processing overhead on the middleboxes by avoiding the examination of all traffic in the path.

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 reduced. In addition, NSH can reduce the number of flow-table entries in the core switches, as its operations only occur at the ingress switches. Further, software-based switches can be used at the ingress point to overcome the limitations of flow-table entries.

7. Use case study

In this section, we provide a use case of DDoS attack mitigation to demonstrate our approach in improving the resilience of the `CyberShip`. The DDoS attack is one of the most prevalent attack in the Internet [3,4]. The main purpose is to exemplify the process of policy translation and enforcement for the data-plane devices. Fig. 3 describes an example on how to mitigate the attacks on `AEMC`. It is worth noting that the mitigation is also applicable to other critical components of ship systems.

The network topology is consists of an attacker denoted as *A*, `IBC` and `AEMC`. `Mitigation` and `Detection` engine are deployed

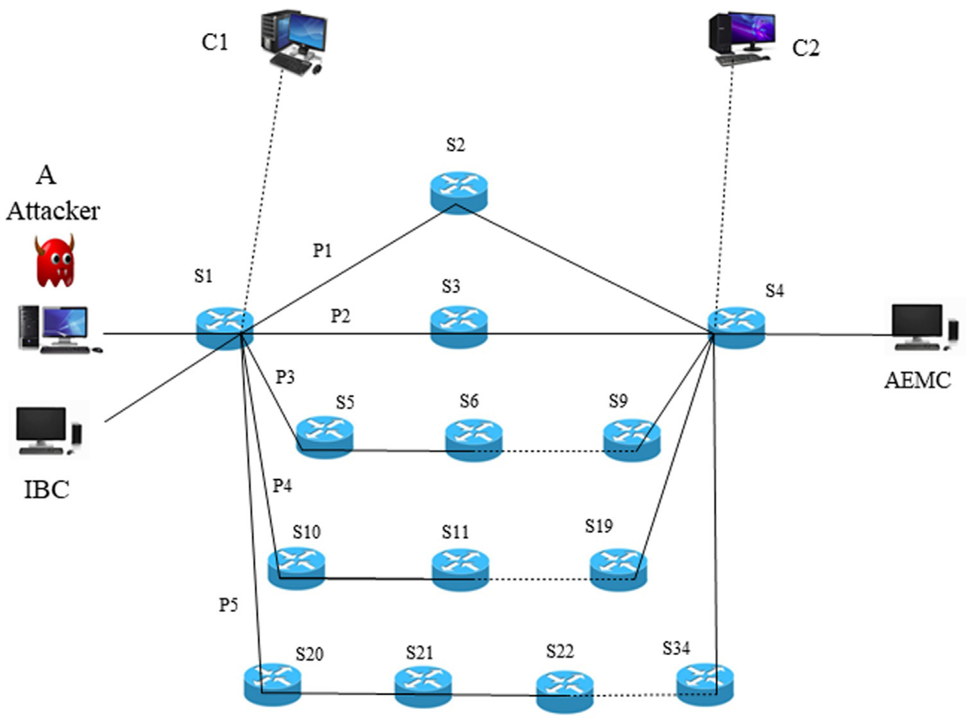

Fig. 3. Network topology configuration in the experiment.

on a separate controller, denoted as C_1 and C_2 respectively. Controller C_1 was connected through the switch S_1 and managed all switches in the network except for the switch S_4. By contrast, controller C_2 and `AEMC` were connected through the switch S_4. In this case, the `Detection Engine` is deployed close to `AEMC` to ensure the detection effectiveness. For our use case, detection is performed based on a threshold set regarding packet arrival rate, average bytes per flow, and average duration per flow [45]. If both packet arrival rate and average bytes per flow are bigger than the threshold, then the `Detection engine` can generate an alarm. As the bridge controller can collect and analyze the data received from different bridge devices, it is able to either increase or decrease the speed by sending messages to `AEMC`, or to reroute the ship through different waypoints.

In the remaining parts, we firstly describe the deployment settings and then provide a step-by-step example to showcase how a high-level mitigation policy can be translated into low-level rules.

7.1. Settings for attack mitigation

Alert Message sent by the AEMC. Security alerts are used to report any malicious traffic, which include network and assessment attributes. In our work, the security alert adopts the IDMEF format, as it can provide the flexibility to add additional data field [52]. Our framework can also be applicable with other formats, like the IODEF (Incident Object Description Exchange Format) [53]. Listing 7 shows the security alert to the SDN controller forwarded by `AEMC`.

- The network attributes are comprised of *IP address* and *attack type*. A security alert contains network information such as: source IP: 10.0.0.1 (*A*), destination IP: 10.0.0.3 (*AEMC*), and attack type: UDP_Flood.
- The assessment attributes describe the attack impact. For instance, a low impact severity indicates that the attack impact is not very critical. In this work, we extend the IDMEF alert with an additional flow class of *suspicious*.

DDoS mitigation policy at the SDN controller. It contains the required policies to guide how to process the received security alerts. Listing 8 provides a high-level action `Low-suspicious-path` for mitigating 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>
```

Template instantiation to process suspicious and malicious traffic. It contains the required information of redirecting the suspicious and malicious traffic to middleboxes or through paths provisioned in the network. Path with higher hops is provisioned for the 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 detailed information of concrete path along with the switchID and the output port, in order to steer the suspicious traffic. As shown in Listing 9, we use *labels* to identify paths, i.e., a label

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

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

associated with the path P_3 is "3". A label can be inserted in each flow to steer through a path. Vlan ID is used in our use case, while other fields like MPLS can also be used to insert labels.

Listing 8: Policy to guide how 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 with switchID and output ports

```
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}
```

7.2. Step-by-step example

In this part, we describe an example on how to transform a high-level mitigation policy into low-level rules with the purpose of mitigating the impact of malicious traffic. This example is described under the sample network configuration as shown in Fig. 3 and the high level policy presented in Listing 8. The detailed procedure and steps are described as below.

1. Upon detecting an attack from a machine, denoted as "A", the `Detection` engine sends a security alert to the `Mitigation` engine, which is deployed at the SDN controller C_1. This alert should be written in the IDMEF format on how to process the suspicious traffic. Suppose an intruder launches the flooding attack at the rate of 50 Mbps. In our topology, we used IPERF [54] to generate the traffic.
2. Upon receiving the alert, the `Mitigation` engine extracts the information including source IP (10.0.0.1), destination IP (10.0.0.3), event type (UDP_Flood), flow class (suspicious) and impact severity (low).
3. Then the `Mitigation` engine checks its database to retrieve the high-level action for policy enforcement. Depending on the event type (UDP_Flood), flow class (suspicious) and impact severity (low), it can decide a high-level action as "Low-suspicious-path". The high-level policy for mitigating such UDP flood traffic is described in Listing 8.
4. The high-level action "Low-suspicious-path" is then refined to get the concrete path for redirecting the traffic. For instance, the `Mitigation` engine checks Table 3 to retrieve the path to reroute traffic, i.e., obtaining the path P_3 to redirect the flow. In our topology, we select to provision 40, 30 and 10 Mbps of bandwidth for the traffic with low, medium and high impact severity, respectively.
5. Then, we utilize the template as shown in Listing 9 to get the concrete path along with its label, according to the high-level path name P_3, i.e., the label associated with the path P_3 is "3". It is worth noting that flow from 10.0.0.1 to 10.0.0.3 was traversing through the path P_1.
6. Based on the high-level action "Low-suspicious-path" along with the concrete path and flow information (10.0.0.1, 10.0.0.3), the mitigation engine can enforce the rules in the switches to redirect the suspicious flows.

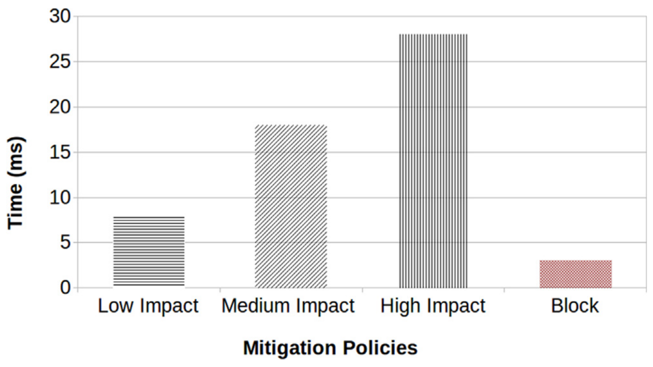

Fig. 4. Deployment time of mitigation policies.

To update the rules, the `Mitigation` engine can modify the label in the VLAN ID field to "3" and the output port to "6" at the switch S_1. the If a flow is traversing through the path P_1, then the label in the VLAN ID field from the source IP (10.0.0.1) to the destination IP (10.0.0.3) is "1". Once a policy has been modified in the data-plane devices, traffic is redirected through another path. It is worth noting that the local segment is not stacked in the label, as we do not redirect the suspicious flow to middleboxes.

8. Experimental results

This section investigates the effectiveness of our response mechanism in reducing the impact of malicious traffic and the high-level policy translation mechanism. The scenarios were created using the Mininet emulator [55]. As explained earlier, suspicious flows can be classified into three classes according to their impact severity: low, medium and high. We rely on the existing mechanisms to detect and classify the flows [56].

8.1. Evaluation metrics

The objective of our evaluation is to explore the time consumption of implementing the high-level policies into low-level OpenFlow rules at the switches. We also attempt to evaluate how our framework can successfully reduce the collateral damage on a legitimate traffic caused by suspicious traffic. Table 4 presents the evaluation metrics in this work, including the time consumption of policy implementation, packet loss rate, throughput and network jitter.

8.2. Implementation time of mitigation policy

In Fig. 4, it is observed that the implementation time of deploying the policy to handle suspicious flows remains under 28 ms, even for flows with high impact severity. Intuitively, it is much higher than that of mitigation policies with low and

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 number of data-plane devices for deploying the rules.	Seconds
Packet loss	It evaluates the number of lost packets in the network. For instance, packet loss can occur if the network is congested, or if the rule deployment is delayed at the switches.	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	It measures the variation in arrival time between packets and facilitate the understanding of congestion or malicious traffic on the system. It increases dramatically in case of congestion.	Milliseconds

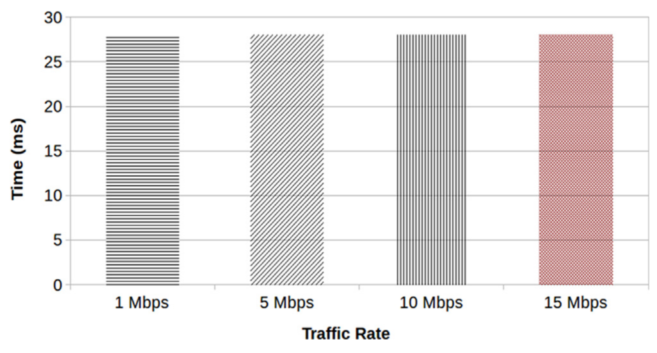

Fig. 5. With different traffic rates, time required to deploy rules to process the high impact severity flows.

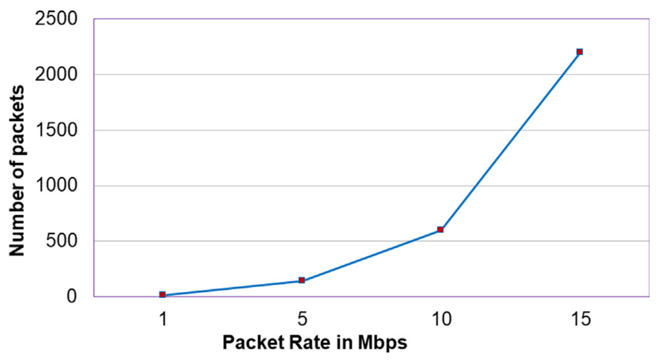

Fig. 6. The number of packets that bypass the ingress switch during the implementation of block action.

medium impact severity, or simply blocking the malicious flows. This is because the path with the highest number of hops has to be provisioned for processing the high impact severity flows. If we consider the number of switches in the path for deploying the rules, it is still reasonable and acceptable between 25 to 30 ms. The implementation time of deploying the policy to handle low and medium impact severity flows are around 8 and 18 ms respectively.

Interestingly, the deployment time of blocking malicious flows is significantly smaller than that of other policies. This is because the block action is only enforced at the ingress switch, and no further deployment is required. Based on our scenario, we consider that the policy deployment time is reasonable. To further reduce the deployment time, we can use some pre-installed rules in some devices within the network.

We also conducted an experiment to evaluate the policy deployment time in handling suspicious flows with different traffic rates from 1 Mbps to 15 Mbps. The purpose is to check whether traffic rate could affect the implementation time. Fig. 5 shows that the time to translate the high-level policy into low level rules are the same after the security alert is received by the mitigation engine. It indicates that the traffic rate does not affect our policy translation process and the deployment of corresponding low-level rules in the network.

8.3. *Malicious traffic filtering*

It is expected that malicious flows can be filtered at the border router of the ship's communication network, as it reduces the impact on the legitimate traffic traversing through the core network. Therefore, we aim to measure how many packets can bypass the ingress switch while security alerts are being processed. As shown in Fig. 6, when the traffic rate is 1 Mbps, around 18 packets could bypass the ingress switch and reach AEMC. If the traffic rate is 5 Mbps, then around 125 packets can reach AEMC. Generally, the bypassed packets have a positive correlation to the traffic rate.

Further, it is found a sharp change when the traffic rate increases from 10 to 15 Mbps. This is because some more delay caused by processing the security alerts and the deployment of low-level rules in dropping the malicious flows. Once the traffic is blocked, controller C_1 can check the flow statistics periodically to understand whether the traffic is still attempting or is no longer active.

8.4. *Packet loss*

We also measure the packet loss during the deployment of low-level rules with low impact severity according to our mechanism. Our mechanism first deploys the low-level rules in the core switches, and then modifies the rules in the egress and ingress switches. In this case, when the rules are being deployed in the core switches, a flow traverses the previous path that can reduce the packet loss.

As shown in Fig. 7, there is an increasing trend for packet loss. With our mechanism, we can minimize the packet loss to a large extent. There is no packet loss when the traffic rate is around 1 Mbps. Even if the traffic rate reaches 15 Mbps, packet loss is only around 7 percent as compared to a 32 percent without our mechanism. This shows that our mechanism can greatly reduce the packet loss rate by deploying the low-level rules in the OpenFlow switches.

In summary, our mechanism can reduce the packet loss rate through deploying the low-level rules in the core switches first, before modifying the rules in the egress and ingress switches.

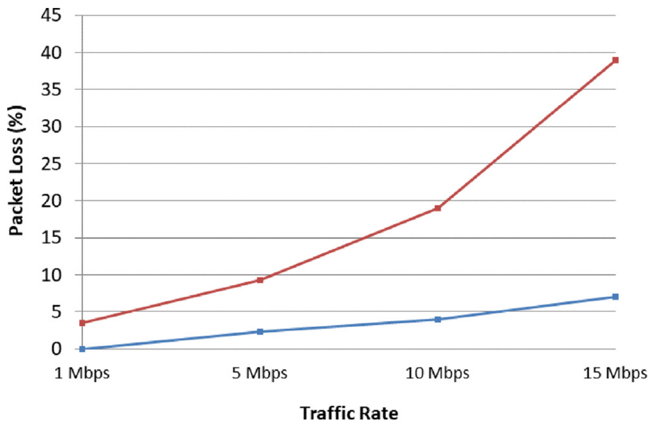

Fig. 7. Packet loss during the deployment of the policy.

Rule Modification at the ingress and egress switches may result in a redirection of flows via a new path. Nevertheless, packet loss is hard to eliminate, because the delay caused by deploying the low-level rules from the controller to the border switches.

8.5. *Throughput of legitimate traffic*

We conducted an experiment to examine the throughput of legitimate traffic under the DDoS attacks. As shown in Fig. 8, we investigate the impact on the legitimate traffic from IBC to AEMC. It is observed that the throughput of legitimate traffic dropped sharply once the DDoS attack started, whereas it can recover to the normal level quickly. This is because the *detection engine* sends an alert, containing the flow information (source and destination IP), security class (suspicious) and impact severity (low), to the *mitigation engine* deployed at the controller C_1. Then, the *mitigation engine* extracts the alerts and checks its policy database for deploying appropriate policies. In the end, the corresponding OpenFlow rules are deployed in the switches to redirect suspicious flows through the provisioned path.

8.6. *Network jitter of legitimate traffic*

Fig. 9 shows how the network jitter of legitimate traffic varies under the DDoS traffic. It is found that the network jitter started to increase when the attack traffic congested the network, while it could decrease quickly when the *mitigation engine* redirects the traffic flows upon receiving the security alerts from the *detection engine*.

9. Discussion and limitations

In this part, we first discuss the major features of our proposed approach and then analyze some limitations.

9.1. *Discussion on our framework*

The main purpose of our framework is to enable a dynamic and automated mitigation of attacks for the ship's communication infrastructure. We utilize the SDN controllers to help react to an event dynamically, i.e., configuring security policies to handle suspicious and malicious flows. Our experimental results demonstrate that our framework can mitigate the attacks by quickly translating high-level policies into low-level OpenFlow rules in the data-plane devices. Upon detection of an attack, the detection module can share the alert with the mitigation module.

Second, the multi-path routing approach in our framework can provide robustness against link failure and traffic congestion. Thanks to the global visibility provided by the SDN controller, we can modify the concerned flows quickly such as flow details, labels and low-level actions. In addition, rules can be pre-configured in the legitimate path in order to reduce the latency. In the provisioned path, rules can be dynamically deployed in the data-plane when it is necessary.

Third, policy language and translation mechanism designed in our framework can offer flexibility to express high-level policy without knowing the specific syntax in the low-level devices. That is, crew members do not need to learn the low-level syntax to express and enforce security policies. This can lighten the burden on crew members and reduce the errors caused by human–computer interaction.

In addition, our framework can reduce the collateral damage to the legitimate traffic, by redirecting the suspicious traffic via lower bandwidth paths. Our framework can also promote the collaboration among controllers in managing various network devices and critical components of ship systems. For instance, AEMC can report the suspicious activities to the SDN controller, and request the Mitigation Engine module to mitigate attacks. The Detection Engine deployed around AEMC provides better detection of attacks, since it controls all the traffic reaching AEMC. This collaboration among controllers helps mitigate attacks in the physical layer as well as in the cyber layer.

9.2. *Limitations*

To the best of our knowledge, our work is an early work in discussing how to combine SDN with ship systems. There are some challenges and limitations on this topic.

Policy conflict. In some cases, policies may be ambiguous if they shadow each other or completely deny one action. For example, it is possible to deploy a forwarding policy that may violate a security policy. Therefore, we have to define priority for different policies, i.e., a security policy should be given higher priority over a forwarding policy.

Computing optimized path. In the current work, we did not consider how to compute an optimized path to redirect suspicious traffic. Paths can be optimized according to different parameters, such as the number of flows in the path, bandwidth of the link, and middleboxes to be traversed. If there is a topology change, then a topology discovery mechanism can be used to retrieve the actual list of valid paths [57]. This could be one of our future directions.

However, computing paths for each security alert may cause processing overhead and create a bottleneck in the framework. Thus, there is a need for adopting some optimization methods to make our framework scalable. To address this problem, we can use some existing solutions like running the multiple instances of the framework in a distributed manner.

Scalability of the framework. Scalability is an important requirement for our framework. For example, the ingress switch can cause a bottleneck in the framework due to the limited TCAM entries. To address this issue, we can use some scalable and efficient solutions like DIFANE [58]. Currently, software-based switches (OVS switches) are also widely deployed in the network to avoid the limitation of flow table entries of hardware switches [59]. In addition, relying on a single SDN controller may create a single point of failure in the framework. Therefore, there is a need to design a framework with multiple SDN controllers. To further reduce delay during the controller-switch communication, some modern SDN controllers can be considered, which can process millions of flows per second [59]. For instance, Open vSwitch is capable of installing tens of thousands of flow rules per second with sub milliseconds of latency [60].

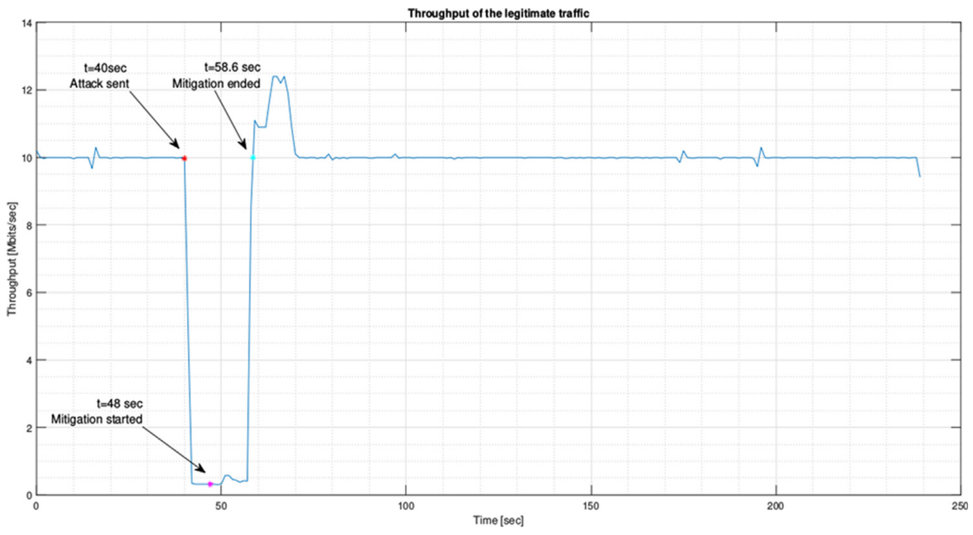

Fig. 8. Throughput of legitimate traffic.

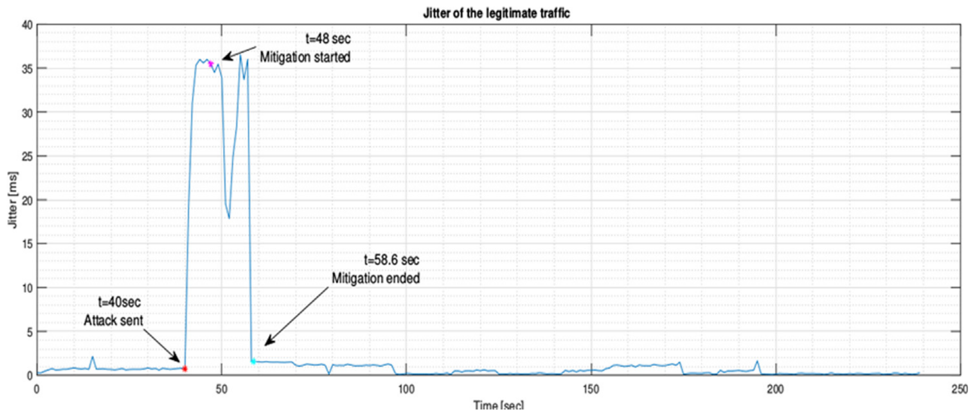

Fig. 9. Network jitter of legitimate traffic.

Middlebox deployment. In this work, we evaluate our framework in an environment with multiple paths and different number of switches. The topology is typical when considering the communication topology of a ship. We utilize labels to help route the traffic via different paths and middleboxes in the network. However, chaining the middleboxes in the ship's communication network may cause some issues like the order of chaining and routing conflicts for network devices. In our future work, we plan to consider other middlebox deployment and validate our results.

Topological configuration. In addition to the wired topology on the ship, it is important to configure wireless topology between devices and switches. Wireless communication can be helpful in controlling the ship from anywhere on the deck. In our future work, we plan to extend the framework by considering wireless communications.

Security of SDN controller. The SDN controller also has many security concerns in practical implementation [61]. Although it is out of the scope in this work, we plan to adopt approach security mechanisms in the literature, e.g., [62–66], to protect the controllers.

10. Conclusion and future work

In this paper, we presented the CyberShip-IoT framework to provide dynamic and automated mitigation of attack traffic in the communication network of ship by leveraging the SDN paradigm. Our framework allows the network operator on-board and crewmembers with little security expertise to express the necessary network and security policies in a human readable language. Moreover, the framework offers a translation mechanism to translate 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. Moreover, it is not required for the network operator and crew member to be present all the time to configure the network for mitigating 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 more efficient mitigation of threats. Finally, our framework relies on multi-path 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.

Acknowledgments

This work was partially supported by the project of Cyber Resilience for the Shipping industry (CyberShip), from the Danish Maritime Fund.

Conflict of interest

None.

Declaration of competing interest

The authors declared that they had no conflicts of interest with respect to their authorship or the publication of this article.

References

[1] Cyber-enabled ships:Deploying information and communications technology in shipping, Tech. rep., Lloyd Register (2016).
[2] H. R.H.ayes, 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, Commun. Surv. Tutor. 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, http://dx.doi.org/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, in: CPSS '15, ACM, 2015, pp. 61–68, http://dx.doi.org/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, http://dx.doi.org/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, J. Netw. Comput. Appl. 131 (2019) 89–108, http://dx.doi.org/10.1016/j.jnca.2019.01.019.
[14] W. Li, W. Meng, L.F. Kwok, Investigating the influence of special on-off attacks on challenge-based collaborative intrusion detection networks, Future Internet 10 (1) (2018) http://dx.doi.org/10.3390/fi10010006.
[15] W. Meng, W. Li, L. Kwok, EFM: Enhancing the performance of signature-based network intrusion detection systems using enhanced filter mechanism, Comput. Secur. 43 (2014) 189–204, http://dx.doi.org/10.1016/j.cose.2014.02.006.
[16] W. Meng, Intrusion detection in the era of iot: Building trust via traffic filtering and sampling, IEEE Comput. 51 (7) (2018) 36–43, http://dx.doi.org/10.1109/MC.2018.3011034.
[17] 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.
[18] I. Alsmadi, D. Xu, Security of software defined networks: A survey, Comput. Secur. 53 (2015) 79–108.
[19] Final Report:Autonomous Engine Room, Tech. rep., MUNIN:Maritime Unmanned Navigation through Intelligence in Network (2015).
[20] Final Report:Autonomous Bridge, Tech. rep., MUNIN:Maritime Unmanned Navigation through Intelligence in Network (2015).
[21] Specification concept of the general technical system redesign, Tech. rep., MUNIN:Maritime Unmanned Navigation through Intelligence in Net- work (2013).
[22] K. RIJECI, S. Krile, D. Kezic, F. Dimc, NMEA Communication Standard for Shipboard Data Architecture NMEA komunikacijski standard za arhitekturu podataka na brodu, 2013.
[23] 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.
[24] 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.
[25] D.-K. Jeon, Y. Lee, A Ship Area Network with WiMedia Wireless Gateway Applying a Cooperative Transmission, 2014.
[26] S.-H. Chang, H. Mao-Sheng, A novel software-defined wireless network architecture to improve ship area network performance, J. Supercomput. 73 (7) (2017) 3149–3160, http://dx.doi.org/10.1007/s11227-016-1930-5.
[27] Guidelines on Maritime Cyber Risk Management, Tech. rep., IMO (2017).
[28] Cyber Security in the Shipping Industry, Tech. rep., Deloitte (2017).
[29] R. Bensing, An Assessment of Vulnerabilities for Shipbased Control Systems (Master's thesis), 2009.
[30] C.R. Hayes, Maritime Cybersecurity: The Future of National Security, Tech. rep., Naval Postgraduate School (2016).
[31] 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, http://dx.doi.org/10.1109/THS.2012.6459832.
[32] Cyber threat to ships - real but manageable, Tech. rep., ABB (2014).
[33] L. Yunfei, C. Yuanbao, W. Xuan, L. Xuan, Z. Qi, A framework of cyber-security protection for warship systems, in: 2015 Sixth International Conference on Intelligent Systems Design and Engineering Applications (ISDEA), 2015, pp. 17–20, http://dx.doi.org/10.1109/ISDEA.2015.14.
[34] 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, http://dx.doi.org/10.2991/iccmcee-15.2015.31.
[35] E. Penera, D. Chasaki, Packet scheduling attacks on shipboard networked control systems, in: 2015 Resilience Week (RWS), 2015, pp. 1–6, http://dx.doi.org/10.1109/RWEEK.2015.7287421.
[36] W. Wu, Ship communication network intrusion signal identification based on hidden markov model, J. Coast. Res. (2018) 868–871.
[37] B. Xing, Y. Jiang, Y. Liu, S. Cao, Risk data analysis based anomaly detection of ship information system, Energies 11 (12) (2018).
[38] Open Networking Foundation, SDN Security Considerations in the Data Center, Tech. rep., ONF (2013).
[39] 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, in: HotSDN '12, ACM, New York, NY, USA, 2012, pp. 43–48, http://dx.doi.org/10.1145/2342441.2342451.
[40] 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, in: HotNets-XII, ACM, New York, NY, USA, 2013, pp. 24:1–24:7, http://dx.doi.org/10.1145/2535771.2535792.
[41] T. Kuhn, A survey and classification of controlled natural languages, Comput. Linguist. 40 (1) (2014) 121–170.
[42] W. Han, C. Lei, A survey on policy languages in network and security management, Comput. Netw. 56 (1) (2012) 477–489.
[43] 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, http://dx.doi.org/10.1109/ICC.2013.6655093.
[44] D. Kreutz, F.M.V. Ramos, P.E. Verissimo, C.E. Rothenberg, S. Azodolmolky, S. Uhlig, Software-defined networking: A comprehensive survey, Proc. IEEE 103 (1) (2015) 14–76, http://dx.doi.org/10.1109/JPROC.2014.2371999.
[45] R. Braga, E. Mota, A. Passito, Lightweight ddos flooding attack detection using nox/openflow, in: IEEE Local Computer Network Conference, 2010, pp. 408–415, http://dx.doi.org/10.1109/LCN.2010.5735752.
[46] 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.
[47] R. Kowalski, M. Sergot, A logic-based calculus of events, New Gen. Comput. 4 (1) (1986) 67–95.
[48] Open Networking Foundation, OpenFlow Switch Specification Version 1.4.0, Tech. rep., ONF (2013).
[49] 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.
[50] S. Homma, D. Lopez, D. Dolson, A. Gorbunov, N. Leymann, K. Naito, M. Stiemerling, P. Bottorff, don.fedyk@hpe.com, Analysis on Forward- ing 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.

[51] 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, p. 24.

[52] B. Feinstein, D. Curry, H. Debar, The Intrusion Detection Message Exchange Format (IDMEF), in: Request for Comments, RFC 4765, RFC Editor, 2015, http://dx.doi.org/10.17487/rfc4765, URL https://rfc-editor.org/rfc/rfc4765.txt.

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

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

[55] 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, in: Hotnets-IX, ACM, New York, NY, USA, 2010, pp. 19:1–19:6, http://dx.doi.org/10.1145/1868447.1868466.

[56] 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, http://dx.doi.org/10.1109/LCN.2010.5735752.

[57] 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, http://dx.doi.org/10.1109/ICSPCS.2014.7021050.

[58] M. Yu, J. Rexford, M.J. Freedman, J. Wang, Scalable flow-based networking with difane, SIGCOMM Comput. Commun. Rev. 41 (4) (2010) –.

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

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

[61] W. Li, W. Meng, L. Kwok, A survey on openflow-based software defined networks: Security challenges and countermeasures, J. Netw. Comput. Appl. 68 (2016) 126–139, http://dx.doi.org/10.1016/j.jnca.2016.04.011.

[62] 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.

[63] W. Meng, K.R. Choo, S. Furnell, A.V. Vasilakos, C.W. Probst, Towards bayesian-based trust management for insider attacks in healthcare software-defined networks, IEEE Trans. Netw. Serv. Manage. 15 (2) (2018) 761–773, http://dx.doi.org/10.1109/TNSM.2018.2815280.

[64] 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, in: CCS '14, ACM, New York, NY, USA, 2014, pp. 78–89, http://dx.doi.org/10.1145/2660267.2660353.

[65] I. Alsmadi, D. Xu, Security of software defined networks: A survey, Comput. Secur. 53 (2015) 79–108.

[66] 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, in: HotSDN '13, ACM, New York, NY, USA, 2013, pp. 19–24, http://dx.doi.org/10.1145/2491185.2491203.

Rishikesh Sahay is a Postdoctoral researcher at Technical University of Denmark. He completed his PhD from Télécom SudParis and University of Pierre and Marie Curie (UPMC) in France. Currently, he is working on cyber resilience for shipping industry. His PhD thesis was focused on autonomic cyber defense using software-defined networking. His research interests include autonomic cyber defense, policy-based network management, software-defined networking, cyber resilience, and network security.

Weizhi Meng is currently an assistant professor in the Cyber Security Section, Department of Applied Mathematics and Computer Science, Technical University of Denmark (DTU), Denmark. He obtained his Ph.D. degree in Computer Science from the City University of Hong Kong (CityU), Hong Kong. Prior to joining DTU, he worked as a research scientist in Infocomm Security (ICS) Department, Institute for Infocomm Research, A∗Star, Singapore, and as a senior research associate in CS Department, CityU. He won the Outstanding Academic Performance Award during his doctoral study, and is a recipient of the Hong Kong Institution of Engineers (HKIE) Outstanding Paper Award for Young Engineers/Researchers in both 2014 and 2017. He is also a recipient of Best Paper Award from ISPEC 2018, and Best Student Paper Award from NSS 2016. His primary research interests are cyber security and intelligent technology in security, including intrusion detection, smartphone security, biometric authentication, HCI security, trust management, blockchain in security, and malware analysis. He served as program committee members for 20 + international conferences. He is a co-PC chair for IEEE Blockchain 2018, IEEE ATC 2019, IFIPTM 2019, Socialsec 2019. He also served as guest editor for FGCS, JISA, Sensors, CAEE, IJDSN, SCN, WCMC, etc.

Daniel Alberto Sepúlveda Estay is a researcher at the Department of Management Engineering in the project CyberShip, Cyber-Resilience for the shipping industry. He holds a MSc in Management Science from the Massachusetts Institute of Technology (MIT), a MSc. in Industrial Engineering from the Pontifical Catholic University of Chile (PUC), and a PhD from the Technical University of Denmark (DTU) in the topic of Cyber risk and security in the global supply chain. Prior to obtaining his PhD, Daniel worked for over 11 years in the supply chain and operations departments of multinational companies both in operational and management roles. He conducts research in the area of Resilience of complex systems, focusing particularly in the development dynamic models of systemic structures and policies that result in a better response to disruptions.

Christian Damsgaard Jensen is an Associate Professor and the Head of the Cyber Security section at the Department of Applied Mathematics and Computer Science, Technical University of Denmark. He holds an M.Sc. in Computer Science from the University of Copenhagen and a Ph.D. in Computer Science from Université Joseph Fourier (Grenoble I, France). He held a position as Lecturer in Computer science at Trinity College Dublin from 1998 to 2002, where he was appointed to his current position. He conducts research in the area of security in distributed systems, where he is particularly interested in the development of models, policies and mechanisms that support secure collaboration in open distributed systems, such as pervasive computing, mobile computing and sensor networks. He has published more than 60 peer-reviewed papers in international journals, conferences and workshops.

Michael Bruhn Barfod is currently an Associate Professor at Technical University of Denmark (DTU), Denmark. His main activities lie within customized decision analysis, decision support systems, group decision making and applied risk assessments. He holds in-depth knowledge about appraisal methods and methodologies concerning transport infrastructure projects — both in respect to research and teaching. His research in particular applies theory in practice, and has often been carried out using real case data in various projects.

Volume 118, Issue , May 2021, Page 492–494

DOI: https://doi.org/10.1016/j.future.2020.08.040

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Corrigendum

Corrigendum to ‘‘CyberShip-IoT: A Dynamic and Adaptive SDN-Based Security Policy Enforcement Framework for Ships’’ [Future Gener. Comput. Syst. 100 (2019) 736–750]

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

[a] *Department of Applied mathematics and Computer Science, Technical University of Denmark, DK-2800 kgs., Lyngby, Denmark*
[b] *Department of Management Engineering, Technical University of Denmark, DK-2800 kgs., Lyngby, Denmark*

A R T I C L E I N F O

Article history:
Available online 4 September 2020

The authors regret to inform that some small issues are found on two figures ∼Figs. 8 and 9, and attach our new figures as below. The main differences are described as follows:

(a) Experimentation is conducted for a short time (180 s)
(b) Attack traffic is launched at 30 s.
(c) Mitigation starts at 40 s.
(d) Legitimate traffic is completely recovered at around 50 s

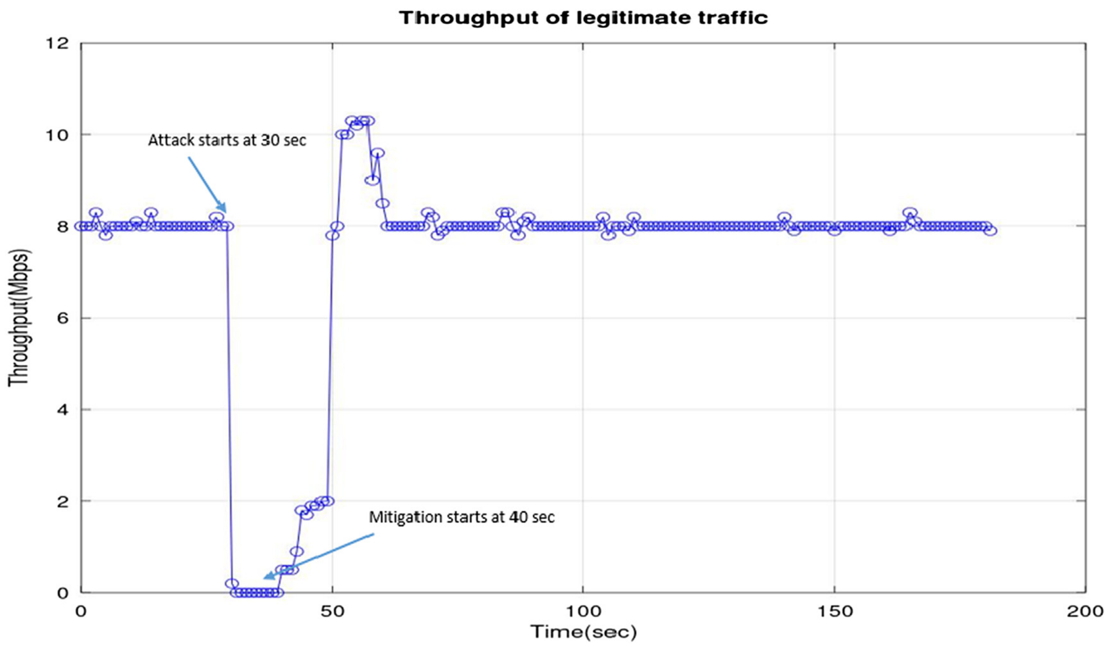

Fig. 8.

DOI of original article: https://doi.org/10.1016/j.future.2019.05.049.
* Corresponding author.
E-mail address: weme@dtu.dk (W. Meng).

https://doi.org/10.1016/j.future.2020.08.040
0167-739X/© 2020 Elsevier B.V. All rights reserved.

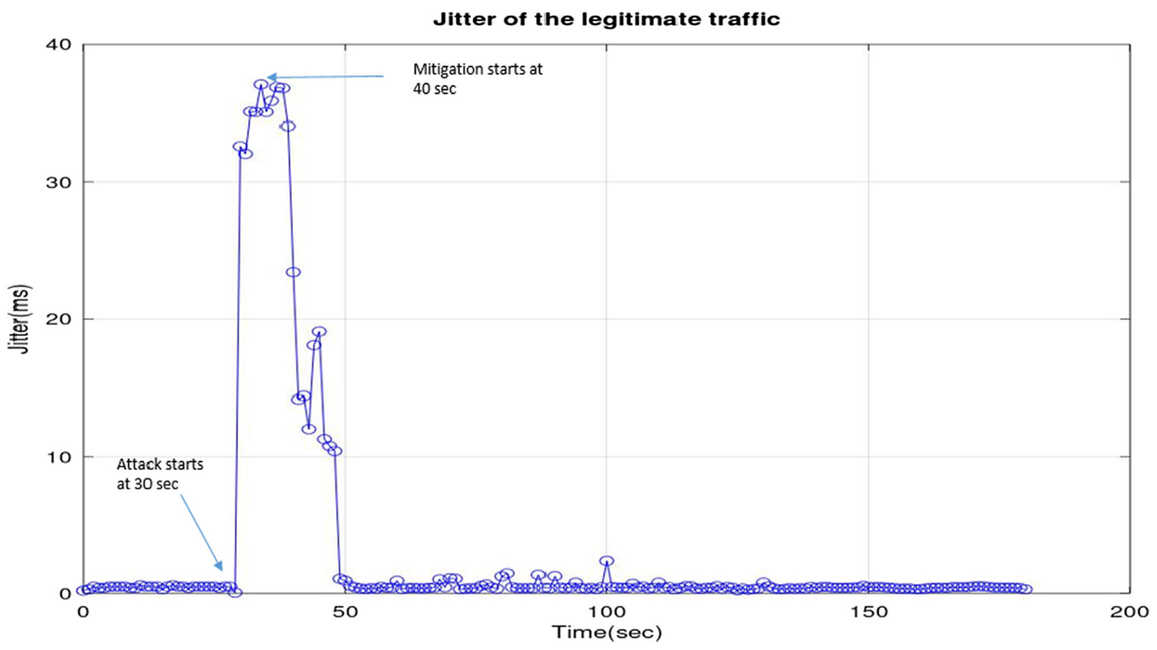

Fig. 9.

In addition, we need to add one reference, which is relevant to this work.

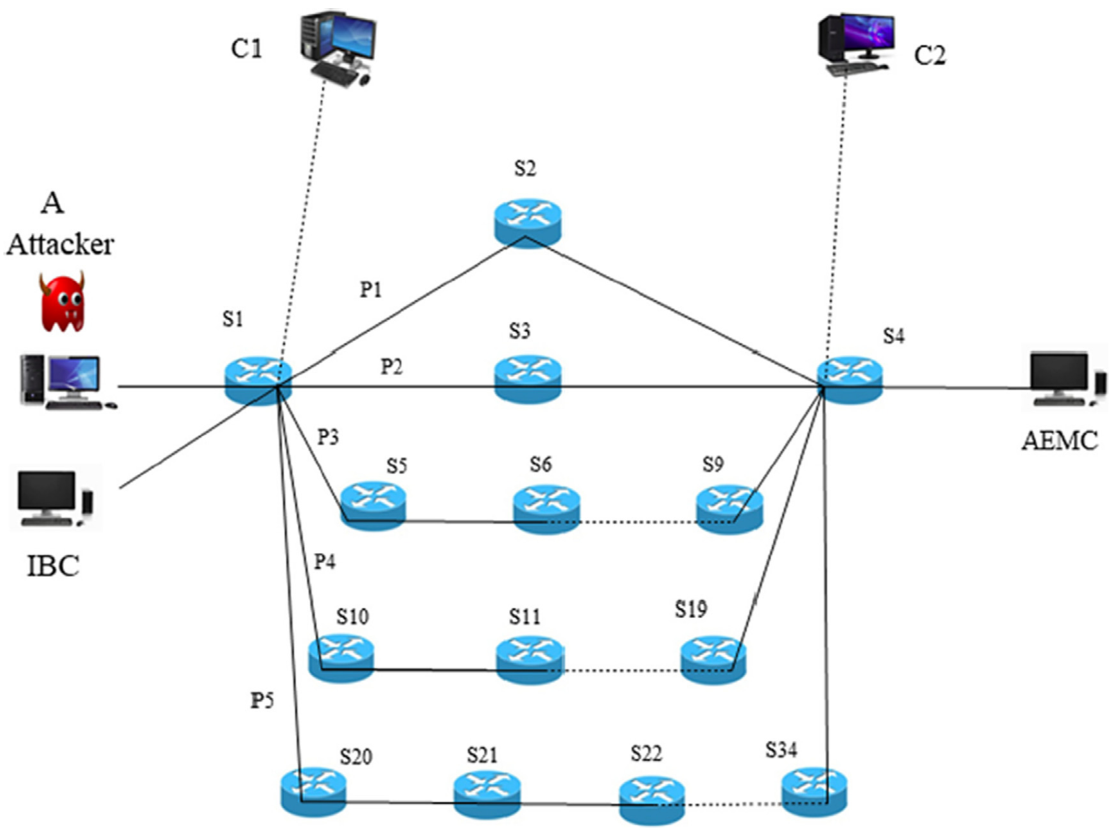

Fig. 3. Network topology configuration in the experiment.

7. Use case study

In this section, we provide a use case of DDoS attack mitigation to demonstrate our approach in improving the resilience of the CyberShip. The DDoS attack is one of the most prevalent attack in the Internet [3,4]. The main purpose is to exemplify the process of policy translation and enforcement for the data-plane devices. Fig. 3 describes an example on how to mitigate the attacks on AEMC. The experimentation setup is expanded from a study done on a simple model [1]. It is worth noting that the mitigation is also applicable to other critical components of ship systems.

The authors would like to apologize for any inconvenience caused.

References

[1] S. Lagouvardou, "Maritime cyber security: concepts, problems, and models", DTU, 2018.