





Article

Detection of Denial of Service Attack in Cloud Based Kubernetes Using eBPF

Amin Sadiq¹, Hassan Jamil Syed^{1,2,*}, Asad Ahmed Ansari¹, Ashraf Osman Ibrahim², Manar Alohalay³ and Muna Elsadiq³

¹ Department of Computer Science, National University Computer and Emerging Sciences, Karachi 75030, Pakistan; k214001@nu.edu.pk (A.S.); k201205@nu.edu.pk (A.A.A.)

² Faculty of Computing and Informatics, Universiti Malaysia Sabah, Kota Kinabalu 88400, Malaysia; ashrafosman@ums.edu.my

³ Department of Information Systems, College of Computer and Information Sciences, Princess Nourah bint Abdulrahman University, P.O. Box 84428, Riyadh 11671, Saudi Arabia; mfalohaly@pnu.edu.sa (M.A.); memohamedahmed@pnu.edu.sa (M.E.)

* Correspondence: shjamil@ums.edu.my

Abstract: Kubernetes is an orchestration tool that runs and manages container-based workloads. It works as a collection of different virtual or physical servers that support multiple storage capacities, provide network functionalities, and keep all containerized applications active in a desired state. It also provides an increasing fleet of different facilities, known as microservices. However, Kubernetes' scalability has led to a complex network structure with an increased attack vector. Attackers can launch a Denial of service (DoS) attack against servers/machines in Kubernetes by producing fake traffic load, for instance. DoS or Distributed Denial of service (DDoS) attacks are malicious attempts to disrupt a targeted service by flooding the target's service with network packets. Constant observation of the network traffic is extremely important for the early detection of such attacks. Extended Berkeley Packet Filter (eBPF) and eXpress Datapath (XDP) are advanced technologies in the Linux kernel that perform high-speed packet processing. In the case of Kubernetes, eBPF and XDP can be used to protect against DDoS attacks by enabling fast and efficient network security policies. For example, XDP can be used to filter out traffic that is not authorized to access the Kubernetes cluster, while eBPF can be used to monitor network traffic for signs of DDoS attacks, such as excessive traffic from a single source. In this research, we utilize eBPF and XDP to build a detection and observation mechanism to filter out malicious content and mitigate a Denial of Service attack on Kubernetes.

Keywords: Denial of service (DoS); Distributed Denial of service (DDoS); attack; kubernetes; Extended Berkeley Packet Filter (eBPF); eXpress Datapath (XDP)



Citation: Sadiq, A.; Syed, H.J.; Ansari, A.A.; Ibrahim, A.O.; Alohalay, M.; Elsadiq, M. Detection of Denial of Service Attack in Cloud Based Kubernetes Using eBPF. *Appl. Sci.* **2023**, *13*, 4700. <https://doi.org/10.3390/app13084700>

Academic Editor: Enno van der Velde

Received: 18 January 2023

Revised: 3 March 2023

Accepted: 4 April 2023

Published: 7 April 2023



Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Before the 1990s, the monitoring and analysis of the packets used to be performed using the classic packet filtering mechanism in which all the packets were copied from the kernel space to the userspace, leading to an increased packet processing latency. In 1992–1993, Steven McCanne and Van Jacobson introduced a mechanism named Berkeley Packet Filter (BPF), in which not all the packets are copied from the kernel space to the userspace [1]. The architecture of the BPF virtual machine is designed for a 32 bits machine with a fixed-length instruction set. BPF was first introduced with the Unix operating system. Then, this filtering mechanism was improved to the Extended Berkeley Packet Filter (eBPF), which was adopted by Linux and Windows [2]. eBPF is a Linux technology that allows for the safe and efficient execution of code within the Linux kernel. It provides a powerful mechanism for implementing complex network functions, such as firewalls, load balancers, and network monitoring, without the need for custom kernel modules. eBPF is a program executed on the Linux kernel when some event is triggered. eBPF provides a wide range of

functionalities such as packet monitoring, tracing new processes, and the detection of any event generated by the computer, etc. eBPF has an improved instruction set architecture (ISA) architecture with more registers. Unlike BPF, eBPF filters all the packets at the kernel space to better decrease the latency. Moreover, the high-speed processing power of eBPF facilitates the analysis of every packet in the network. XDP is a networking technology that provides a fast and efficient way to process network packets at the kernel level. It allows for the processing of packets before they reach the higher-level networking stack, enabling faster and more efficient handling of network traffic. XDP is well-suited for use in applications that require high-speed packet processing, such as network security, load balancing, and network monitoring.

BPF Programs are written in restricted C programming language due to the Linux kernel. However, a tool named BPF Compiler Collection, i.e., BCC [3], makes the BPF program much easier to write using Python and some restricted C languages. This BPF code is first compiled into BPF bytecode in userspace using a low-level virtual machine (LLVM) or Clang. After passing through a verifier (for authentication and verification), a just-in-time compiler helps to convert this code into a native machine code and execute the program using a separate virtual machine on Linux Kernel [4]. Infinite loops, global variables, etc., are not supported, so we must consider this limitation while implementing the programs.

Today, Kubernetes has become a leading open-source orchestration tool that provides dynamic scalability, manages different resources needed by the physical or virtual servers, and fulfills the different adaptations and dependencies of the applications. Kubernetes communicates between the different collections of containers on which different applications are running. It is a complex system that allows communication between different components within a cluster and between pods. The structure of a Kubernetes cluster network can be broken down into several key components: cluster network, pods network, etc. Like any complex software system, Kubernetes might have security flaws that attackers could use against users if it is not secured properly. If the cluster network is not properly secured, an attacker could potentially intercept sensitive data or inject malicious traffic into the network. Kubernetes uses containers to deploy applications, and these containers are built from images. If an attacker can inject malicious code into a container image, they could potentially compromise the application running in that container. Kubelet, which facilitates communication between various nodes, and containers that are executing on the nodes can all be big contributors to making the cluster vulnerable to attack while considering the cluster network. In addition, it provides the interface platform for the application that needs to provide high availability and scalability through its microservices [5]. Users of Kubernetes test the deployment features of an application by diverting traffic between different containers. On the other hand, malicious users may compromise Kubernetes clusters to control other users' containers. Over the past years, methods and gadgets for launching an attack have drastically improved. DoS (Denial of Service) or DDoS (Distributed Denial of Service) are potentially the most renowned attacks. The goal behind DoS or DDoS attacks is to bring the server out of reach for others and it will create a potential gap in the performance overhead of the system. eBPF is a revolutionary technology in a Linux kernel that handles everyday tasks and affects the processes running on the server. eBPF has a significant impact on system performance because of different complexities and the frequency of the events. It can be used to perform various tasks ranging from package isolation to opening a particular document in the file system. Unlike other ways to change the behavior of a part, eBPF does not require part recompilation or part module assembly and is protected from execution by security checks by stacking code verifiers. Each second is important in detecting malicious packets so that particular prevention methods can be applied. Express Data Path (XDP) [6] gives the quick execution of group dealing with and programmable association way in the cycle of Linux. XDP generally processes the packages' level of the stack, which prompts the quick presentation without compromising the programmability.

In the past few years, detection and prevention approaches have been topics of interest in the fields of networks and cybersecurity. One of the major security risks to many networks in the last ten years is a DDoS attack. It can not only stop authorized users from using and accessing network resources, but it can also destroy the network [7]. Cloud services and microservices are at high risk of DDOS attacks [8]. In 2019, it is reported that a Kubernetes cluster that was being utilized to host a cryptocurrency mining operation came under attack from a distributed denial-of-service (DDoS) attack. The attack, which wrecked the Kubernetes cluster and halted the mining process, was launched by the attackers using a huge number of compromised machines. A DDoS attack against a Kubernetes cluster hosting a machine learning platform was reported by researchers in 2020. The attack was carried out by the attackers using a large number of compromised devices, which led to the collapse of the Kubernetes cluster. In [9,10], it was highlighted that these types of cyber attacks are not only limited to a specific area but operate in a different area as well, which helps the potential attackers use the open and less privileged loophole and then make the entire system compromised.

We have analyzed the detailed review of the existing cloud solutions to review the loopholes in the cloud application and also investigate the overhead in the cloud solutions related to the attacks. The literature on these existing solutions is comprehensively reviewed from well-known conferences and journals. The review of the papers mostly contains the major aspects of observability, detection, mitigation, overheads of cloud computing, architecture, cloud types, etc. In addition, we investigated and analyzed the issues related to the performance of the existing solutions. We have investigated the identified research problem by analyzing current experiments on cloud orchestration tools like Kubernetes. Existing cloud solutions are implemented using Kafka, etc. These existing solutions contain many overheads, latency issues, accuracy, etc. However, detecting DDoS attacks in the Kubernetes network remains a challenging problem. High-load traffic attacks are the primary type of DDoS attack through which different network machines produce unnecessary load toward the server. In Kubernetes, such attacks pose a major concern given that it usually contains multiple container applications running on [11]. Traditional DDoS countermeasures cause significant overhead on the Kubernetes cluster. Therefore, it is necessary to devise a solution to observe and detect the traffic of DDoS attacks efficiently. We have proposed an eBPF solution for the observation and detection of DDoS to address the problems identified in the current cloud orchestration tool like Kubernetes. The proposed solution is very lightweight and scalable on multiple clusters, and one of the most important features is that it is very fast so that we can monitor every aspect of the incoming packets. In this research, the proposed DDoS solution using eBPF resulted in a negligible overhead for the virtual machines (VMs) installed on the various Kubernetes Cluster nodes. We summarize the contribution of this study as follows:

- To analyze the state-of-the-art cloud solutions and identify the most relevant problem in the current cloud Kubernetes solutions.
- To propose and implement an effective solution by which we can detect and observe the Distributed Denial of Service attack in the Kubernetes cluster using eBPF.
- To validate and measure the evaluation metric based on overhead while detecting and observing the DDOS attack using eBPF and without it.

The rest of the article is structured as follows. Section 2 reviews the related works. The architecture of the proposed solution is presented in Section 3. The suggested architecture's implementation phases are elaborated on in Section 4. Section 5 provides a discussion of the results. Finally, we conclude the study in Section 6.

2. Related Work

In this section, we discuss recent studies on the eBPF, microservices, DDOS, and the application of the eBPF to various containerized services on Kubernetes. Several researchers [12–14], and [15–17] have proposed different strategies and solutions for the

use of eBPF on different aspects of emerging technologies. We provided a brief description of the related studies in Table 1.

Table 1. Summary of related studies.

| S.No. | Existing Work | Summary | Limitations |
|-------|---------------|--|---|
| 1 | [13] | For the first time, eBPF was used for the detection tasks of the kernel like observing the page faults, over-viewing the memory distribution, etc. The author(s) has encountered the advantage and success factor of eBPF when developing complex networks. | While focusing on the numerous complicated services, the proposed design should also assess the system overhead. |
| 2 | [12] | The author(s) have proposed a solution for the Intrusion detection of the system by using eBPF at the kernel which filtered the huge amount of network packets using the matching rule set of the snorting method. | It seems that there should be state-of-the-art comparison with other features to evaluate the performance of the system |
| 3 | [16] | The aim is to implement an efficient and effective processing pipeline that will help to prevent Distributed Denial of Service (DDoS) attacks. The authors have increased the mitigation power at the server end which will drop the DDoS by using the specific ruleset. eBPF is a flexible tool that will help to sample the network traffic of the operating system using hardware-based filtering. | When the server has a lot of resources preserved on it, the performance of the work will be reduced by the suggested approach. |
| 4 | [17] | The author has proposed a solution based on eBPF which bypasses the kernel and filters all the network packets at the userspace which usually skips the overhead of the Linux network stack. The proposed solution has overcome the performance issue that arises when packets are filtered and analyzed at kernel space. | The recommended architectural designs perform with a small number of alterations. The lack of drivers and other resources will require significant adjustments to the solution. |
| 5 | [15] | The author(s) proposed a non-intrusive protocol-independent intelligent analytics system, more efficient and accurate and also provides the point-to-point observability solution with works on eBPF. It also enables the transparency of the packets for an application like Kubernetes in which we have nodes, clusters, pods, containers, etc. | The proposed machine learning-based observability approach will become more complicated as the number of nodes and pods rises. |
| 6 | [18] | A framework was introduced by the author on Network Functions Virtualization to enhance its capability for in-kernel packet processing applications which provides flexibility dynamically in the run-time environment. The framework used for the development of the complex networks provides efficiency in the kernel data plane and flexible user-space control plane for the persistence environment. | There is a potential bottleneck mechanism that could cause a delay in the mechanism's latency when a large number of packets are transferred from kernel space to userspace. |
| 7 | [14] | The author(s) have highlighted the benefits of XDP and eBPF using the offloading mechanism based on eBPF to the kernel space from the userspace. The offloading-based mechanism is an optimized solution for the packet processing. | When a significant number of packets arrive at the same time, the proposed solution will begin to slow down and SmartNIC will become stuck. |

The above-mentioned studies can be classified into two categories: (1) studies that focused on the use of eBPF in the complex network structure and also the filtering of the kernel packets at the userspace, which will further help to formulate different intrusion patterns [12,13,19]; (2) studies that discussed the mitigation of the DDoS attack on the server side and also bypassing the kernel and filtering out all the detailed information on the user side, which will overcome the overhead on the Linux network stack [16,17]. In [15,18], the accuracy and efficiency of the observability and monitoring based on the microservices were discussed, along with using a framework for Network Function Virtualization to achieve flexibility. In [14], the authors have discussed the offloading mechanism of the packet using eBPF.

In [12,13], the kernel used eBPF for detection duties like looking for page faults and monitoring the distribution of memory, among other things. The benefit and success

element of eBPF was experienced while creating complex networks. Use of eBPF at the kernel, which filtered a sizable number of network packets using a matching rule set of the Snort approach, as the solution for the system's intrusion detection.

In [16,17], the objective is to develop a processing pipeline that is effective and efficient, which will aid in preventing DDoS attacks. By employing a specific ruleset, the authors have boosted the server-end mitigation strength, decreasing the DDoS. eBPF-based solutions essentially bypass the kernel and filter all network packets in userspace, avoiding the Linux network stack's overhead in the process. The performance problem that occurs when packets are filtered and examined in the kernel space has been resolved by the suggested fix.

In [15,18], a non-intrusive, protocol-independent intelligent analytics system that operates on eBPF is more effective, accurate, and offers a point-to-point observability solution. Additionally, it makes the packet transparency for the microservices possible. In addition, it enhances its capability for in-kernel packet processing applications, which provides flexibility dynamically in the run-time environment.

In [14], the advantages of XDP and eBPF utilizing the eBPF-based offloading method from the userspace to the kernel space have been noted. It is ideal to use the offloading-based approach for packet processing. Unlike earlier studies, our proposed solution stands out for its lightweight monitoring and its scalability.

Unlike the previous studies, this work applied eBPF and XDP which can significantly contribute to the safety and stability of Kubernetes clusters, to detect and observe the attacks on the server, and therefore determine and prevent disruption or flooding of the target's service with network packets. Therefore, utilizing a Denial-of-Service attack against Kubernetes using eBPF and XDP, we suggested a detection and observation mechanism in this study to filter out the harmful information. By using the eBPF and XDP programs, malicious packets can be rejected based on the eBPF program's defined routines. To the best of our knowledge, no existing work has been done on the detection and observation of DDOS. The technique that is used for the cluster's nodes is very effective and the script does not overburden the nodes working on different VMs. Our proposed solution is a lightweight, efficient, and scalable solution.

3. Architecture of the Proposed Solution

In this section, we discuss the architecture of the proposed solution for detecting and observing the DDoS attack in different Kubernetes nodes. Furthermore, we show how the proposed system can scale horizontally and vertically and be dynamic. Vertically scalable nodes can manage numerous resources simultaneously, whereas horizontally scalable nodes will continue to function as the number of nodes in the network grows. Scaling the nodes horizontally or vertically needs to be done on the cluster of the Kubernetes on which we are operating the proposed solution. We note that the suggested solution detects and observes DDoS attacks either with the help of the eBPF software or without it. Figure 1 illustrates the schematic block diagram of the Proposed Solution. In both cases, minimal human engagement is necessary.

3.1. Generalized Overview of the Proposed Solution

An eBPF is a robust tool for examining and modifying the Linux kernel in real time. Small, in-kernel programs known as "eBPF programs" can be attached to a variety of locations along the kernel's execution path, including system calls, kprobes (dynamic function probes), and tracepoints (static function probes). These applications can be used to gather information, filter events, and even change how the kernel behaves. In the context of detecting a DDoS attack in Kubernetes, eBPF can be used to attach a program to a network interface to monitor incoming traffic and identify patterns that may indicate a DDoS attack. The proposed solution contains the information from all the VMs working on the cloud, i.e., the cloud contains different nodes, which form the cluster of Kubernetes. In our experiment, we have one master node and multiple workers or child nodes working

on different networking IP addresses. We have launched separate VMs on each node with different OS interfaces. The effectiveness and low overhead of eBPF for DDoS detection in Kubernetes are two of its main advantages. eBPF programs can process data quickly without affecting system performance because they run directly in the kernel. This is crucial when using Kubernetes since the enormous number of containers and microservices it supports might produce high network traffic. Overall, using eBPF for DDoS detection in Kubernetes can offer a quick and effective solution to watch incoming traffic and spot potential attacks, assisting in the defense of the cluster and preserving service availability.

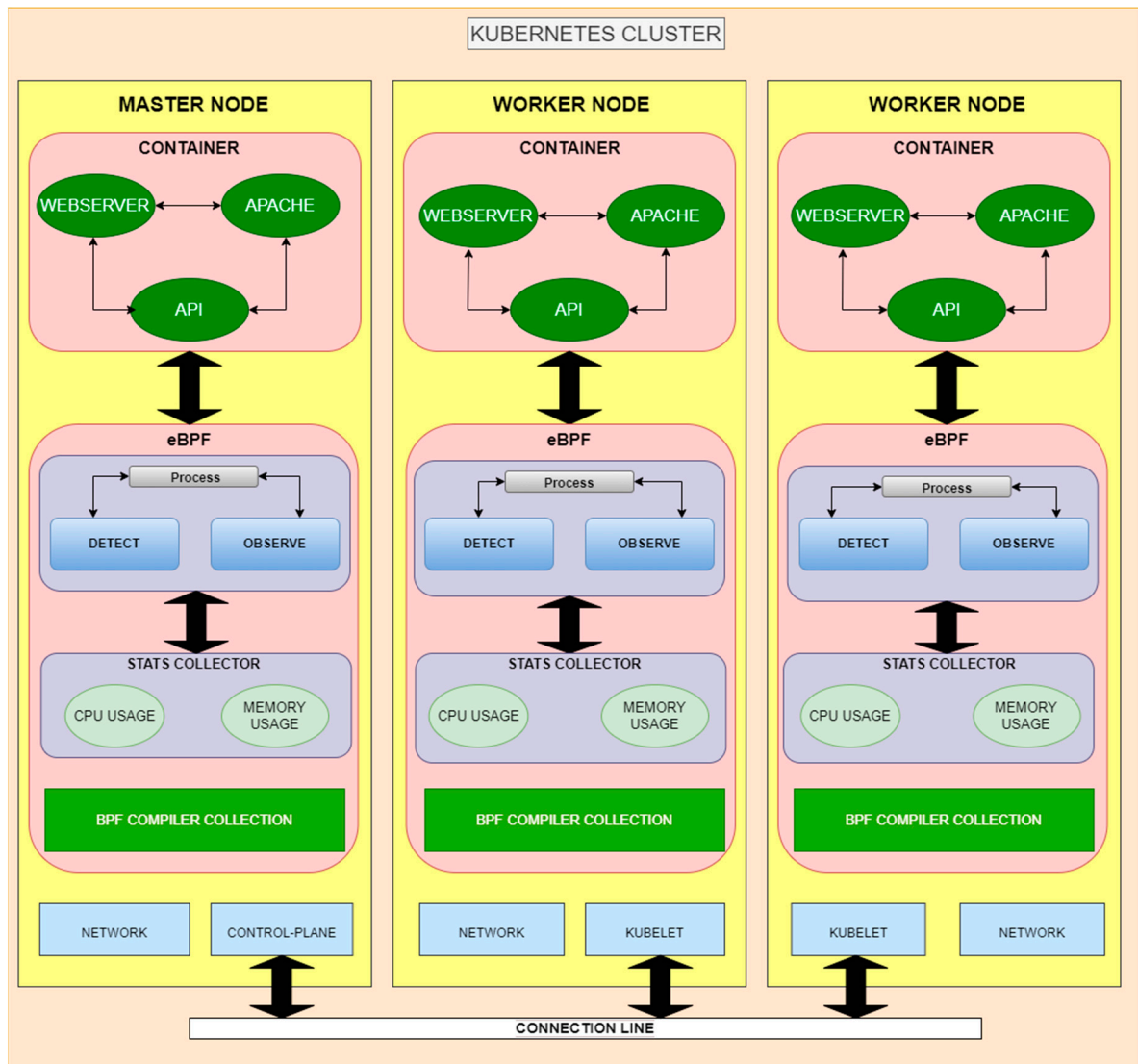


Figure 1. Schematic block diagram of the proposed solution.

3.2. Algorithm(s) Used for the Proposed Solution

In this subsection, we discuss our proposed algorithms for the observation and detection of DDoS attacks in Kubernetes using eBPF. Our proposed algorithm, which is presented as a pseudocode, addresses the issues of the detection and observation of DDoS attacks. Our suggested method uses the given approach to gather data. The following are the main problems that we suggested the approach attempts to solve:

- To detect the IP address used for the attack.
- To observe or monitor the running processes.

The algorithms that were utilized to gather the information for these measures are listed below:

- Algorithm 1: To detect the IP Address with the help of payload.
- Algorithm 2: To observe the running processes with respect to process ID (PID).

Algorithm 1 specifies how the IP address of the attack is determined on the targeted node and is used to detect DDoS attacks.

Algorithm 1: Detection of the attack Pseudocode

Require: Define a variable to hold the number of connections from each IP address
 Initialization of the System

```

1  Read the values
2  Ensure: kprobe_tcp_v4_connect, tcp_probe->ipv4.connect
3  IP_address ← Set_IP_of_the_connection
4  Count_Array ← Increment_count_for_IP_address_in_connection
5  Ensure: tracepoint_raw_syscalls_sys_enter_connect
6  Show the number of connections from each IP address
7  if (Count_Array ≥ 10) then
8      src,dst ← tcp_probe.ip4.saddr, tcp_probe.ip4.daddr
9      get_load ← tcp_probe.ip4.payload
10     c_payload ← bpf_skb_load_bytes(get_load, BPF_CURRENT_CPU)
11     if (c_payload is malicious) then
12         Possible DDOS attack from the IP address
13     end
14 end
```

In Algorithm 1, we have presented the detection of the DDOS attack on the Kubernetes cluster in the form of the pseudocode. In this algorithm, firstly, we have declared all the variables like *IP_address* and *Count_Array* for holding the numbers of the nodes' IP along with their connection. In line 1, the system will start reading the values. In Line 2, we have defined BPF hooks which will be triggered when a TCP connection is established. In line 3, with the help of the BPF hook, we can have the different IPs that are connected. In Line 4, the count_array will increment the counter of the DDOS attack against each IP. Lines 5 and 6 provide information to the user about the connection on the targeted node. In Line 7, we have checked the threshold value of each connection request. In Lines 8 and 9, we have to get the source and destination IP address which helps to get the source attack IP. In line 10, we have gathered the payload which is sent from the attacking node. In line 11, we analyzed the payload using the BPF hook to check the malicious content. In the last line, it will show the potential attack from the IP address. Algorithm 1 is used to detect the attack, which is coming from multiple nodes, i.e., different IPs with the same type of request and payload. The connections are built on the extrapolated scale, so its DDOS can be shown in Figure 2 that the detection is from different nodes. In addition, the payload helps to detect multiple requests, i.e., DDOS requests or the genuine request of GET or POST that are generated from different IPs. The script works by monitoring network traffic in real time and tracking the number of connections from each IP address. The threshold of ten connections is arbitrary and can be adjusted to suit the specific needs of the network being monitored. A higher threshold may be more appropriate for networks that experience high volumes of legitimate traffic, while a lower threshold may be more effective for networks that are more susceptible to DDOS attacks. In general, setting a threshold for the number of connections from a single IP address can help to identify potential DDOS attacks by detecting patterns of traffic that are inconsistent with normal network behavior.

```
vagrant@master-node:~$ sudo bpftrace -e '#include "ddos.bt"'
IP 10.0.0.11 has made 50 connections
IP 10.0.0.12 has made 30 connections
IP 10.0.0.13 has made 9 connections
IP 10.0.0.14 has made 40 connections
IP 10.0.0.15 has made 6 connections
IP 10.0.0.16 has made 15 connections
Possible DDOS attack from 10.0.0.11
Possible DDOS attack from 10.0.0.12
Possible DDOS attack from 10.0.0.14
Possible DDOS attack from 10.0.0.16
```

Figure 2. Detection of the Attack on the target node.

The observation of the processes that pass through eBPF is presented in the form of pseudo-code, shown below in Algorithm 2. Through this eBPF software, the processes' observation is kept under surveillance. The monitoring of the process includes all pertinent information about the process, including its status, remote and local IP addresses, port number, acknowledgement status, and Process ID, which is one of the most crucial elements. In Algorithm 2, we have used a dictionary for each IP that will get connected. In Line 1, we simply read all the values from the system. In Line 2, we have gotten all the possible TCP connections. In lines 4–7, we get the local and remote address, the status of the connection, and most importantly, the PID of each process.

Algorithm 2: Observation of the Processes Pseudocode

Require: Define a dictionary to map TCP states to their human-readable names
 Initialization of the System

- 1 Read the values
- 2 $tcp_connections \leftarrow \text{Get_list_of_all_TCP_connections}$
- 3 **while** ($tcp_connections \neq 0$) **do**
- 4 $Local_Address_Port \leftarrow \text{Get_Local_Address_Port}$
- 5 $Remote_Address_Port \leftarrow \text{Get_Remote_Address_Port}$
- 6 $Status \leftarrow \text{Get_Status_Name}$
- 7 $Process_ID \leftarrow \text{Get_PID}$
- 8 **end**

4. Implementations

In this section, we discuss and provide the process we used to implement the suggested solution. This section's primary goal/motivation is to provide a detailed description of the experimental setup and implementation of the suggested solution, along with the necessary data collection methods and the parameters used to analyze the performance of our suggested algorithm. The section begins with evaluating the results for real and simulation environments. Then, the request and the response of the Kubernetes nodes created using the simulation environments are analyzed. We build a test bed for the detection and observation of our suggested prevention mechanism in the set of trials using an improved Berkeley packet filter to characterize the lightweight elements of our method. Additionally, a summary of the evaluation metric used is provided in the section.

The proposed solution is implemented on the cloud with the help of the Kubernetes cluster mechanism. The implementation is very straightforward. It can be easily understood by looking at the detailed implementation of the virtual machine, i.e., VMs running different operating systems and Kubernetes processes. Each process has a unique process ID (PID) on the process running on it. The processes not only belong to the OS but are also related to the Kubernetes processes, like how many nodes are connected with the master node and the status of the nodes running on different VMs. The PID number can easily identify all

the relevant information on the VM operating system. Below is the observation working of the DDOS, as shown in Figure 3.

PROCESS ID

PROCESS NAME

vagrant@worker-node01:~\$ sudo tcpstates-bpfcc

| SKADDR | C-PID | C-COMM | LADDR | LPORT | RADDR | RPORT | OLDSTATE | -> | NEWSTATE | MS |
|------------------|--------|------------|------------|-------|------------|-------|-------------|----|-------------|----------|
| ffff8f218f4c88c0 | 397180 | docker-pro | 172.18.0.1 | 52872 | 172.18.0.2 | 80 | ESTABLISHED | -> | CLOSE_WAIT | 0.000 |
| ffff8f218f4cf1c0 | 769038 | apache2 | 172.18.0.2 | 80 | 172.18.0.1 | 52870 | ESTABLISHED | -> | FIN_WAIT1 | 0.000 |
| ffff8f2186da3d40 | 397180 | docker-pro | 172.18.0.1 | 52870 | 172.18.0.2 | 80 | ESTABLISHED | -> | CLOSE_WAIT | 0.000 |
| ffff8f2193f84600 | 761051 | apache2 | 172.18.0.2 | 80 | 172.18.0.1 | 52838 | ESTABLISHED | -> | FIN_WAIT1 | 0.000 |
| ffff8f2193f83480 | 397180 | docker-pro | 172.18.0.1 | 52838 | 172.18.0.2 | 80 | ESTABLISHED | -> | CLOSE_WAIT | 0.000 |
| ffff8f218f4ce900 | 900256 | xerxes-II | 172.17.0.2 | 54712 | 10.0.0.11 | 80 | CLOSE_WAIT | -> | CLOSE | 3020.401 |
| ffff8f2192939180 | 900256 | xerxes-II | 172.17.0.2 | 54718 | 10.0.0.11 | 80 | CLOSE_WAIT | -> | CLOSE | 3008.622 |
| ffff8f219293b480 | 900257 | xerxes-II | 172.17.0.2 | 54728 | 10.0.0.11 | 80 | CLOSE_WAIT | -> | CLOSE | 3008.556 |
| ffff8f219293f1c0 | 397180 | docker-pro | 10.0.0.11 | 80 | 172.17.0.2 | 54740 | FIN_WAIT2 | -> | FIN_WAIT2 | 2709.190 |
| ffff8f219293f1c0 | 397180 | docker-pro | 10.0.0.11 | 80 | 172.17.0.2 | 54740 | FIN_WAIT2 | -> | CLOSE | 0.002 |
| ffff8f2197ef6900 | 900256 | xerxes-II | 172.17.0.2 | 0 | 10.0.0.11 | 80 | CLOSE | -> | SYN_SENT | 2375.221 |
| ffff8f2197ef6900 | 900256 | xerxes-II | 172.17.0.2 | 56604 | 10.0.0.11 | 80 | SYN_SENT | -> | ESTABLISHED | 0.775 |
| ffff8f2197ef2bc0 | 900256 | xerxes-II | 0.0.0.0 | 80 | 0.0.0.0 | 0 | LISTEN | -> | SYN_RECV | 2376.088 |
| ffff8f2197ef2bc0 | 900256 | xerxes-II | 10.0.0.11 | 80 | 172.17.0.2 | 56604 | SYN_RECV | -> | ESTABLISHED | 0.010 |
| ffff8f2197ef0000 | 900256 | xerxes-II | 172.17.0.2 | 0 | 10.0.0.11 | 80 | CLOSE | -> | SYN_SENT | 0.000 |

Figure 3. Observation of the attack on the target node.

Detecting the DDoS attack on the target machine is also very important. We implemented the detection of the DDoS attack on the containerized-based application, i.e., a web application working along with an Apache server. After the attack is initiated, it is detected by the eBPF program based on a specific threshold and considered as a possible IP source of the attack. Figure 2 shows an example of the detection of the attack on the target node.

Based on the preliminary data acquired, we assessed the monitoring capabilities of the suggested solution and highlighted its advantages. The following qualities are addressed to validate it:

- Overhead

To measure the monitoring overhead of the Kubernetes that was created using the Vagrant Cloud. The following is the overhead that is analyzed.

4.1. CPU Overhead

To validate the overhead of the VM deployed on the node of the cluster, we utilize the top command. The Linux top program collects all the metrics of the VMs which are running on the cloud that was created using a grant. We have executed the eBPF program script for the detection of the DDOS attack on the nodes of the cluster so we can find the CPU usage with and without the eBPF program. The script was running continuously so that the observation of the pattern of the DDOS attack on the targeted node can be visualized. The eBPF application is used to observe CPU utilization, and it was also done without its assistance. An average CPU usage score of 10 is obtained.

4.2. Memory Overhead

A cloud microservices system may be severely impacted by memory overhead. The SAR tool was used to acquire and collect memory statistics to calculate the memory overhead. Almost 10 s on average was recorded. The memory overhead was measured on the master and worker nodes of the Kubernetes cluster to demonstrate that our proposed method does not produce any memory usage. The proposed solution is implemented on the Kubernetes cluster node that contains the master and worker nodes. Both the nodes of the cluster, i.e., master and worker, are used to collect the important data through which memory usage and its utilization are observed. Memory usage for the overhead is the first measure on the master node. The master node has a VM deployed on it and a containerized

service for the website that is working on it. The observation for memory usage is done with the help of the eBPF program on which an average of 10 is measured and it was also done without the help of the eBPF program.

5. Results and Discussion

The evaluations we conducted in the previous section are all discussed in this section, along with an empirical analysis of the suggested solution. We have carried out several experiments to achieve the desired outcomes for detecting and observing. To evaluate the evaluation metric based on the elements described in the previous section, we primarily aim to observe and detect the DDoS attack on the Kubernetes cluster using eBPF. The evaluation findings of the performance of VM with various Kubernetes nodes are also covered. The proposed solution provides the features of observing and detecting DDoS attacks in Kubernetes using eBPF. Our proposed solution is efficient, accurate, and lightweight due to its working on user mode and low overhead producing mechanism. The evaluation process for observation and detection capabilities is presented in this section. The graphical information is displayed to assess the performance. To verify the effectiveness of our suggested solution, the following feature is measured, i.e., overhead.

If the microservice, in collaboration with the eBPF solution, creates a reasonable performance overhead on the system, it is not a good solution, even if the other features of that solution are fine; however, creating a fair overhead renders it unacceptable. In this part, we evaluate the overhead of our suggested solution and demonstrate that it has no noticeable impact on either the master node or the worker nodes of the Kubernetes cluster. To evaluate the overhead on the overall cloud system, we measure the following attributes on the node of the Kubernetes cluster, i.e., master node or worker nodes. To measure the overhead of our proposed solution, we have executed the nodes on which the VMs are deployed and launched using different operating system images.

Table 2 demonstrates the high efficiency of our proposed solution, which is attributed to the effective utilization of BPF hooks for detecting and monitoring DDOS attacks. The proposed solution effectively captures the traffic load of requests by utilizing BPF hooks and probes. The payload detection BPF hook is highly effective at handling the load of incoming requests, ensuring that all requests sent from various nodes are thoroughly checked. Our Kubernetes cluster network is distributed, and all nodes share the same kernel from the host machine. This allows the eBPF to work on the host server and enables all nodes to detect attacks, regardless of the number of containerized applications deployed on the cluster. Each node in the cluster has established a varying number of connections, which our algorithm evaluates by analyzing the number of requests that exceed a predetermined threshold value. The payload allows for the identification of DDOS attacks versus genuine requests. As a result, our eBPF-based solution has proven to be highly efficient and effective. In [20,21], various network policies, pod policies, and generic policies were implemented to secure the Kubernetes cluster from unwanted traffic and external unauthorized users. In addition, these measures help to prevent unexpected attacks on the Kubernetes cluster network.

5.1. CPU Overhead

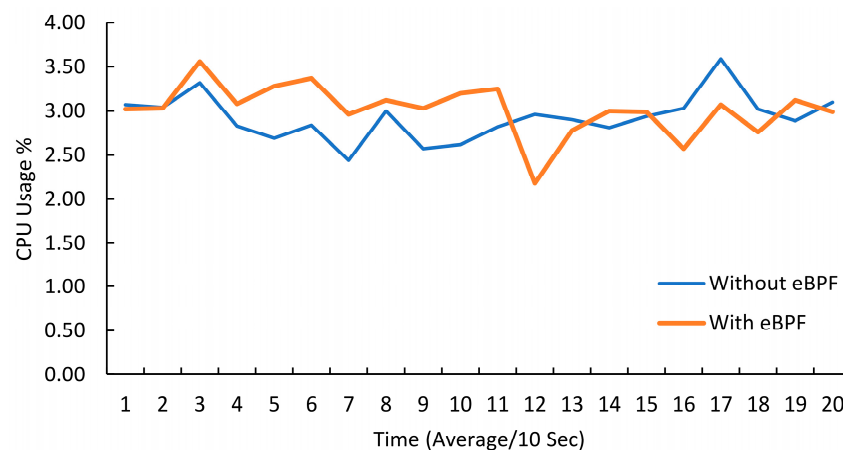
In our proposed solution, we have measured the CPU utilization using eBPF programs and without them. To determine the CPU overhead, we measured it by running it on the master node and the worker nodes.

Table 2. Experimental Results.

| eBPF Hooks | | | No. of Connections | Arbitrary Threshold Value (≥ 10) | Action | | DDOS Request for Total No. of Connections % | Efficiency of Solution with Respect to eBPF (%) |
|---|---|-----------------|--------------------|---|-------------|-----------------|---|---|
| kprobe_tcp_v4_connect, tcp_probe>ipv4.connect | bpf_skb_load_bytes (Load, BPF_CUR-ENT_ CPU) | tcpstates-bpfcc | | | DDOS Attack | Genuine Request | | |
| | 10.0.0.11 | | 50 | Yes | 40 | 10 | 80% | 100% |
| | 10.0.0.12 | | 30 | Yes | 25 | 5 | 84% | 100% |
| | 10.0.0.13 | | 9 | No | N/A | N/A | N/A | N/A |
| | 10.0.0.14 | | 40 | Yes | 20 | 20 | 50% | 100% |
| | 10.0.0.15 | | 6 | No | N/A | N/A | N/A | N/A |
| | 10.0.0.16 | | 15 | Yes | 13 | 2 | 87% | 100% |

i. CPU overhead on the Master Node

To determine the overhead of CPU utilization, we have used the Linux tool for the recording of the CPU utilization statistics, i.e., system activity report (SAR). In an average of every 10 s, the record was examined. When stats are gathered, either the eBPF application is used or not, as illustrated in Figure 4, to record the information. When using the eBPF programs versus not using them, there is rarely any change in CPU usage. Thus, it was established that the master node's CPU overhead is not present.

**Figure 4.** Master Node CPU Overhead Analysis.

ii. CPU overhead on the Worker Node

The worker node of the Kubernetes cluster also has a VM launched on it. Our proposed solution also recorded the statistics of the CPU utilization of the usage with the help of the SAR tool on an average of every 10 s. Although we have done the same experiment on the worker node as on the master node, we have recorded the stats as shown in Figure 5, which is with and without eBPF. Similarly, it is revealed that there is no such overhead on the utilization of the CPU.

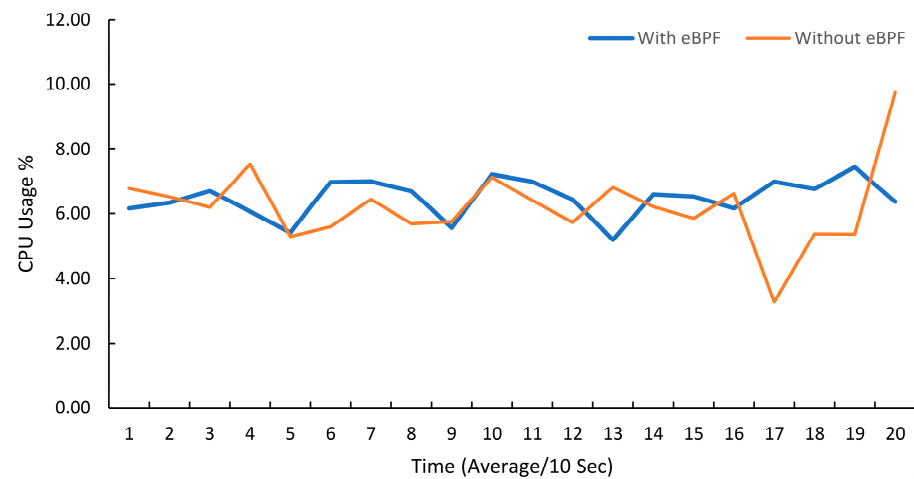


Figure 5. Worker Node CPU Overhead Analysis.

5.2. Memory Overhead

Memory also plays an important role in the use of microservices and has several impacts on it. To determine the memory utilization on the nodes of the Kubernetes cluster, i.e., the master node and the worker node, we have used the SAR tool to record the average 10-second stats. To demonstrate that the proposed solution does not create any type of overhead, we have measured it on the master node and the worker node.

i. Memory overhead on the Master Node

We have executed our proposed solution on the master node of the Kubernetes cluster and it shows in Figure 6 that our solution does not create any type of overhead on the memory of the master node. The difference between the recorded values of the programs running with the eBPF or without it is between 0.1–0.8. This difference shows that there is no such overhead imposed on the master node.

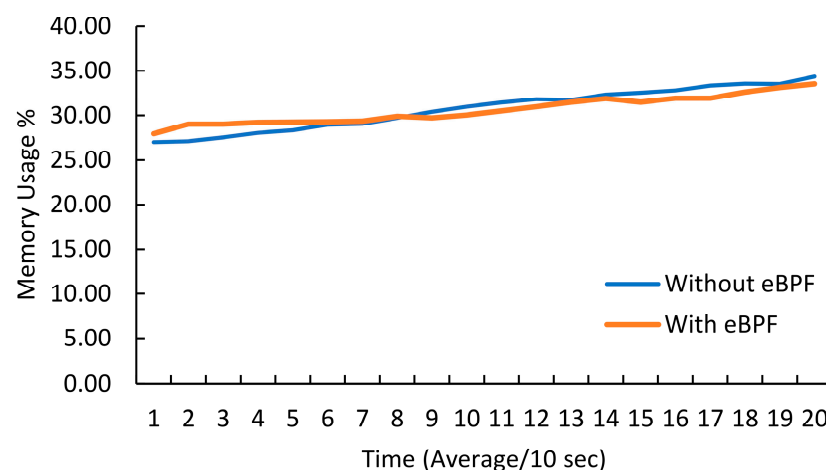


Figure 6. Master Node Memory Overhead Analysis.

ii. Memory overhead on the Worker Node

Our proposed solution gathered all the recorded information using the eBPF program and without it. In Figure 7, it is shown that, in the initial stage, there is a huge difference in the execution of the memory with the use of eBPF because of the full utilization of the memory. However, we have also observed that, after a while, the difference between the eBPF program and without it became very small. Therefore, it can be easily concluded that after some time the difference will become near 1–2 and, in some cases, less than 1, so on the worker node we also have a negligible overhead of memory.

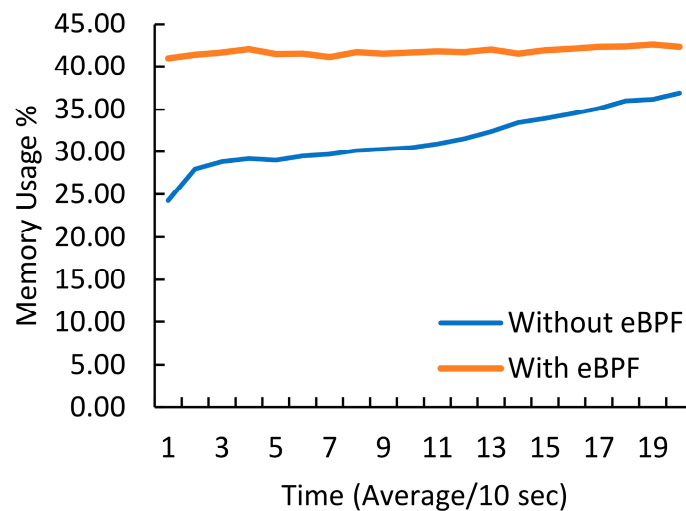


Figure 7. Worker Node Memory Overhead Analysis.

6. Conclusions

In conclusion, detecting and observing DDoS attacks in Kubernetes using eBPF is a highly effective way to improve the security of containerized applications. With the increasing popularity of Kubernetes for deploying and managing containerized applications, the risk of DDoS attacks is also increasing. eBPF provides a powerful way to monitor and analyze network traffic at the kernel level, allowing for highly granular and real-time visibility into the behaviour of the system. By using eBPF, we can detect and observe DDoS attacks in real time, identifying attacks by detecting and observing them before they cause significant damage to the system. This not only improves the security of the system but also helps maintain the high availability and uptime of the applications running in Kubernetes. The use of eBPF can also significantly reduce the workload of security teams, automating the process of detecting and mitigating attacks and enabling faster incident response times. Compared to conventional methods, using eBPF for DDoS detection in Kubernetes has many advantages. The effective execution of programs on live network packets is made possible by the high-performance technology known as eBPF, which is crucial for real-time DDoS detection.

Additionally, eBPF has a low overhead, allowing it to be used in production settings without degrading the cluster's performance, which we also observe in our results and discussion sections. Kubernetes' eBPF for DDoS detection can boost the cluster's security and dependability. DDoS attacks can be prevented using eBPF's real-time detection and mitigation, which guarantees that users can continue to access services while the cluster is protected. In this approach, eBPF can significantly contribute to the safety and stability of Kubernetes clusters.

We have presented the detection and observation of the DDOS attack in Kubernetes using eBPF, which is based on the cloud. To perform the detection and observation, we have implemented the BPF script on each node of the Kubernetes cluster, which is running on different VMs. Afterwards, we evaluated the detection and observation of the DDOS attack which the help of the implemented scripts. The detection will be done on the incoming packets on their extrapolated full load toward the targeted victim node. On the behalf of the results, we have evaluated the evaluation metrics which are based on the overhead of CPU utilization and memory usage. It is also observed from the evaluation metrics that using eBPF generates low overhead on CPU and memory and it is more effective while using detecting and observing DDOS attacks. The proposed solution is quite an effective technique for microservices like Kubernetes. To the best of our knowledge, no one has detected and observed DDOS in the research. The technique that is used for the cluster's nodes is very effective, and the script does not overburden the nodes working on different VMs. Our proposed solution is lightweight, efficient, and scalable.

The proposed solution for the observation and detection of the DDOS attack has some limitations. While using eBPF to detect and observe DDoS attacks in a Kubernetes cluster has its benefits, it also has some limitations and areas for future work. Some of these include the adoption of this technology by organizations may be difficult due to the complexity of eBPF programs and the lack of standards in the eBPF ecosystem. As Kubernetes clusters continue to scale, it will be important to ensure that eBPF programs can scale with them. eBPF can be integrated with other Kubernetes technologies to provide a more complete picture of network traffic and security and it will help to create easiness to adopt and use.

Overall, the use of eBPF for detecting and observing DDoS attacks in Kubernetes is a highly effective and scalable solution that can significantly improve the security and reliability of containerized applications. As containerization continues to gain momentum, the importance of robust security measures will only increase, and eBPF offers a powerful tool in the fight against cyber threats.

Author Contributions: Conceptualization, A.S. and H.J.S.; methodology, A.S. and H.J.S.; validation, A.S., H.J.S. and A.A.A.; investigation, A.O.I., M.A. and M.E.; resources, A.O.I., M.A. and M.E.; data curation, A.O.I., M.A. and M.E.; writing—original draft preparation, A.S.; writing—review and editing, A.S., H.J.S. and A.A.A.; visualization, A.O.I., M.A. and M.E.; supervision, H.J.S.; project administration, A.O.I., M.A. and M.E.; funding acquisition, M.A. and M.E. All authors have read and agreed to the published version of the manuscript.

Funding: This project is funded by Princess Nourah bint Abdulrahman University Researchers Supporting Project number (PNURSP2023R383), Princess Nourah bint Abdulrahman University, Riyadh, Saudi Arabia.

Institutional Review Board Statement: Not Applicable.

Informed Consent Statement: Not Applicable.

Data Availability Statement: Not Applicable.

Acknowledgments: This work was supported through Princess Nourah bint Abdulrahman University Researchers Supporting Project number (PNURSP2023R383), Princess Nourah bint Abdulrahman University, Riyadh, Saudi Arabia.

Conflicts of Interest: The authors declare that they have no conflict of interest.

References

1. McCanne, S.; Jacobson, V. The BSD Packet Filter: A New Architecture for User-Level Packet Capture. In Proceedings of the USENIX Winter, San Diego, CA, USA, 25–29 January 1993; Volume 46.
2. Vieira, M.A.; Castanho, M.S.; Pacifico, R.D.; Santos, E.R.; Júnior, E.P.C.; Vieira, L.F. Fast packet processing with eBPF and XDP: Concepts, code, challenges, and applications. *ACM Comput. Surv. CSUR* **2020**, *53*, 1–36. [\[CrossRef\]](#)
3. Scholz, D.; Raumer, D.; Emmerich, P.; Kurtz, A.; Lesiak, K.; Carle, G. Performance implications of packet filtering with Linux eBPF. In Proceedings of the 2018 30th International Teletraffic Congress (ITC 30), Vienna, Austria, 3–7 September 2018; Volume 1, pp. 209–217.
4. Nelson, L.; Van Geffen, J.; Torlak, E.; Wang, X. Specification and verification in the field: Applying formal methods to {BPF} just-in-time compilers in the Linux kernel. In Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20), Virtual Conference, 4–6 November 2020; pp. 41–61.
5. Bernstein, D. Containers and cloud: From LXC to docker to Kubernetes. *IEEE Cloud Comput.* **2014**, *1*, 81–84. [\[CrossRef\]](#)
6. Høiland-Jørgensen, T.; Brouer, J.D.; Borkmann, D.; Fastabend, J.; Herbert, T.; Ahern, D.; Miller, D. The express data path: Fast programmable packet processing in the operating system kernel. In Proceedings of the 14th International Conference on Emerging Networking Experiments and Technologies, Heraklion, Greece, 4–7 December 2018; pp. 54–66.
7. Fan, C.; Kaliyamurthy, N.M.; Chen, S.; Jiang, H.; Zhou, Y.; Campbell, C. Detection of DDoS attacks in software defined networking using entropy. *Appl. Sci.* **2021**, *12*, 370. [\[CrossRef\]](#)
8. Alashhab, Z.R.; Anbar, M.; Singh, M.M.; Hasbullah, I.H.; Jain, P.; Al-Amiedy, T.A. Distributed Denial of Service Attacks against Cloud Computing Environment: Survey, Issues, Challenges and Coherent Taxonomy. *Appl. Sci.* **2022**, *12*, 12441. [\[CrossRef\]](#)
9. Heidari, A.; Jabraeil Jamali, M.A. Internet of Things intrusion detection systems: A comprehensive review and future directions. *Clust. Comput.* **2022**, *2022*, 1–28. [\[CrossRef\]](#)
10. Heidari, A.; Navimipour, N.J.; Unal, M. A Secure Intrusion Detection Platform Using Blockchain and Radial Basis Function Neural Networks for Internet of Drones. *IEEE Internet Things J.* **2023**, *2023*, 3237661. [\[CrossRef\]](#)

11. Riadi, I.; Umar, R.; Sugandi, A. Web forensic on Kubernetes cluster services using GRR rapid response framework. *Int. J. Sci. Technol. Res.* **2020**, *9*, 3484–3488.
12. Wang, S.-Y.; Chang, J.-C. Design and implementation of an intrusion detection system by using extended BPF in the Linux kernel. *J. Netw. Comput. Appl.* **2022**, *198*, 103283. [[CrossRef](#)]
13. Miano, S.; Bertrone, M.; Risso, F.; Tumolo, M.; Bernal, M.V. Creating complex network services with eBPF: Experience and lessons learned. In Proceedings of the 2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR), Bucharest, Romania, 18–20 June 2018; pp. 1–8.
14. Hohlfeld, O.; Krude, J.; Reelfs, J.H.; Ruth, J.; Wehrle, K. Demystifying the Performance of XDP BPF. In Proceedings of the 2019 IEEE Conference on Network Softwarization (NetSoft), Paris, France, 24–28 June 2019.
15. Liu, C.; Cai, Z.; Wang, B.; Tang, Z.; Liu, J. A protocol-independent container network observability analysis system based on eBPF. In Proceedings of the 2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS), Hong Kong, China, 2–4 December 2020; pp. 697–702.
16. Bertin, G. XDP in practice: Integrating XDP into our DDoS mitigation pipeline. In Proceedings of the Technical Conference on Linux Networking, Netdev, Montréal, QC, Canada, 6–8 April 2017; Volume 2.
17. Miano, S.; Doriguzzi-Corin, R.; Risso, F.; Siracusa, D.; Sommesse, R. Introducing smartnics in server-based data plane processing: The DDoS mitigation use case. *IEEE Access* **2019**, *7*, 107161–107170. [[CrossRef](#)]
18. Miano, S.; Risso, F.; Bernal, M.V.; Bertrone, M.; Lu, Y. A framework for eBPF-based network functions in an era of microservices. *IEEE Trans. Netw. Serv. Manag.* **2021**, *18*, 133–151. [[CrossRef](#)]
19. Abranches, M.; Michel, O.; Keller, E.; Schmid, S. Efficient Network Monitoring Applications in the Kernel with eBPF and XDP. In Proceedings of the 2021 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), Heraklion, Greece, 9–11 November 2021; pp. 28–34.
20. Shamim, M.S.I.; Bhuiyan, F.A.; Rahman, A. XI commandments of Kubernetes security: A systematization of knowledge related to Kubernetes security practices. In Proceedings of the 2020 IEEE Secure Development (SecDev), Atlanta, GA, USA, 28–30 September 2020; pp. 58–64.
21. Minna, F.; Blaise, A.; Rebecchi, F.; Chandrasekaran, B.; Massacci, F. Understanding the security implications of Kubernetes networking. *IEEE Secur. Priv.* **2021**, *19*, 46–56. [[CrossRef](#)]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.