University of Amsterdam

MSc Security and Network Engineering
Research Project 2

Transparent malicious traffic detection and mitigation using the Nvidia BlueField DPU

July 5, 2021

Students:
Jelle Ermerins
jermerins@os3.nl

Ward Bakker
wbakker@os3.nl

Supervisors:
Cedric Both
cedric@datadigest.nl

Course:
Research Project 2

Abstract

With the ever-growing bandwidth consumption and malicious activity, detecting malicious traffic and reactively filtering this malicious traffic is essential to protect networks. Running an Intrusion Detection System (IDS) can be done on a separate, dedicated machine, but can also be done on a Data Processing Unit (DPU). This research looks into the possibilities and limitations of the Nvidia BlueField DPU. This in regards to the optimization of IDS software, where we implement an IDS on the BlueField DPU. To measure the performance, large amounts of (malicious) traffic are sent to be processed by the DPU. We found several general optimization- and Bluefield specific optimization techniques for the IDS software, like DOCA, DPDK and XDP & eBPF. Unfortunately, due to compatibility constraints, we experienced issues regarding compiling and running these optimizations. Because of the limitations, we experimented with running an IDS on the BlueField DPU without any optimizations and sent traffic with various throughputs to measure its performance. The results showed that the BlueField-1 is capable of running a transparent IDS on the DPU, but lacks the performance when processing large amounts of malicious traffic. This is due to the lack of optimizations within the software, and the lack of hardware acceleration features on the BlueField-1 to process traffic more efficiently.

1 Introduction

Processing large amounts of traffic can be heavy on the CPU. To decrease the load on the CPU, Network Interface Cards (NICs) are used to process the traffic more efficiently. However, certain operations, like cryptographic or memory operations, might still have limited performance on plain NICs [1]. To achieve better performance on these operations, we can use a SmartNIC containing a Data Processing Unit (DPU). Offloading these specific operations to a DPU containing a variety of hardware resources can increase the performance and process traffic even more efficiently.

Detecting malicious traffic and reactively filtering this malicious traffic is essential to protect a network from threats like a ransomware attack. Detection of such an attack can be done using an Intrusion Detection System (IDS) or an Intrusion Prevention System (IPS). Well known examples of these are Snort and Suricata. These applications match the traffic against a predefined set rules. With the implemented ruleset the IDS can indicate if the traffic is malicious or not. An IDS can be a separate or dedicated machine on the network. With the Nvidia BlueField line of products, it is possible to install an IDS on a DPU. This DPU is then able to processes the traffic transparently. An advantage of using a DPU with open-source IDS software is that it might be a cheaper option than using vendor-provided solutions. Another benefit is the programmability of a DPU. Besides running, for example, IDS software, it can also be used for other network functions.

The Nvidia BlueField SmartNIC [2] is a NIC containing a DPU that can be used for offloading certain operations, like cryptographic, regular expression, or memory operations. The BlueField contains a separate ARM-based System-on-Chip, with its own storage and memory. This allows for the installation of a separate operating system. This might be useful for running IDS software on the DPU itself and benefit from the offloading of certain operations to achieve better performance regarding malicious traffic detection and mitigation.

In our research, we analyze the possibilities and limitations of the Nvidia BlueField SmartNIC regarding the performance when running an IDS on the DPU. We analyze optimization techniques to improve the performance of an IDS and if it is possible to implement these techniques on the BlueField DPU. We then create an experimental setup where an IDS is installed on the DPU. From this point, we run experiments where various types of (malicious) traffic are sent to the DPU. This supports us to find out how the DPU performs as an IDS, when processing large amounts of (malicious) traffic.

1.1 Structure

The remainder of this paper is structured as follows: In Section 2, we outline previously performed research that relates to ours. In Section 3 we outline the research question for this research. In Section 4, we provide background information for this research. In Section 5 we define our experiments as well as the experimental environment, along with the structure of gathering the results. In Section 6, we present our findings and in Section 7 we discuss our findings. Based on our findings, we draw conclusions and present these in Section 8. As last, Section 9 suggests topics for future research.

2 Related work

Network cards are the physical building blocks on which modern networks are designed. These network cards are becoming increasingly more powerful. This increase in calculation capacity reduces the load on the CPU, which enables powerful network equipment to be useful in large cloud base operations. Liu et al. concluded in their research that most Network Interface Cards (NICs) are capable of achieving a bandwidth of 100- or 200 Gbps. However, when enabling features like IDS, VPNs, NAT, or load balancing the performance could drop by approximately 70% [1]. Liu et al. also show that using a SmartNIC like the Nvidia BlueField can achieve better performance by offloading certain operations like encryption and memory operations.

In 2019 Bangalore created a system that detects and mitigates Distributed Denial of Service (DDoS) attacks [3]. This is achieved using the known methods of Intrusion Detection Systems (IDS). Bangalore used an anomaly-based approach and with enough data created a signature for the DDoS. Signatures are used because they are more efficient when compared to anomaly detection. Based on the signature an Access Control List (ACL) could be automatically created to stop future attacks that match the signature.

Zhang et al. researched optimizing the Snort IDS using the Data Plane Development Kit (DPDK) [4]. They concluded that using DPDK to optimize and increase the detection rate of Snort solves the performance issues. This is achieved due to the use of a different packet polling method, decreased CPU interrupts, and fewer memory operations.

Qinwen et al. compared the performance of three different Network Intrusion Detection Systems (NIDS), Snort, Bro and Suricata. In their research they investigated key factors (which include systems resources usage, packet processing and packet drop rate) that limit the NIDSs it's ability in large-scale networks, with large amounts of traffic. They used the following experiments to determine these limits, Default configuration, detection algorithm optimizations and modified the DAQ configuration to improve capture performance. The results indicate that certain pattern matching algorithms will drop a significant amount of traffic. Especially using Suricata the CPU load is high when using pattern matching algorithms and the NIDSs uses a small number of memory resources. [5]

Qinwen et al. published on the issues with open-source intrusion detection systems in high-speed networks. In their research, they were able to reach 60Gb/s when using a transparent IDS in combination with eXpress Data Path (XDP). From 60 Gb/s, Suricata would start and drop packets. They also tried to detect malicious activity with a single rule. At this point, they acknowledge that using a single rule is not sufficient for detecting malicious activity. Their research recommends using hardware acceleration solutions to capture traffic on 100Gb/s links. [6]

In 2018 Yang et al. proposed an architecture for regular expression (RE) matching that consumes multiple bytes per time. The architecture contains all the advantages of three Field Programmable Gate Arrays (FPGA)-based algorithms to improve the matching speed: Simple State Merge Tree (SSMT), Distribute Data in Round-Robin (DDRR), and Multi-path Speculation. While maintaining memory efficient, they were able to process 140Gbps on a single FPGA. [7]

3 Research Question

In this research, we look at what optimizations are possible, specifically for an Intrusion Detection System, and how these optimizations can be implemented on the BlueField DPU.

We also implement an IDS on the BlueField DPU and look at how it performs when it is processing large amounts of (malicious) traffic. The following research question will be answered:

What are the limitations of the Nvidia BlueField SmartNIC regarding the detection of large amounts of malicious traffic?

To answer our research question, we first look at what optimizations are possible, where the following sub-question will be answered:

- *What are the possibilities regarding the optimization of IDS software within the BlueField DPU?*

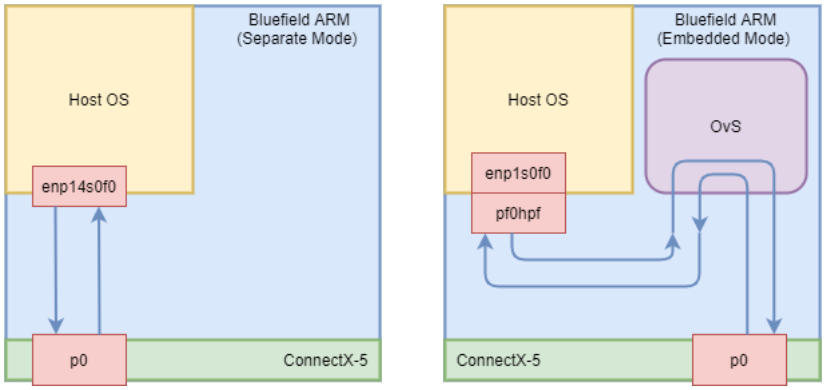
4 Background

4.1 The Nvidia BlueField DPU

The SmartNIC we are working with is the Nvidia BlueField 1 [8]. This card has two 100 Gbps network ports, so in theory, it can send and receive 200 Gbps of traffic. The card has a 16-core ARM CPU and 16GB of DDR4 memory, an entire system on the card itself where we can run, for example, Ubuntu or CentOS.

4.1.1 BlueField SmartNIC Modes

The BlueField can operate in two modes: Separated mode and Embedded (SmartNIC) mode [9][10]. In separate mode, the host system and the ARM system on the DPU act as separate entities, where the host only uses the connectX-5 module of the SmartNIC to send and receive traffic. The other mode is the Embedded mode, also called SmartNIC mode. In this mode, all traffic going to and from the host passes the ARM system on the SmartNIC. Figure 1 shows these two modes. On the left, we can see the separated mode. On the right, we can see the embedded mode, where all the traffic is going through the ARM system.



(a) Separate Mode (b) Embedded Mode

Figure 1: The two modes of operation on the BlueField

When running embedded mode, we have access to a separate ARM system between the host and the outer world. This also means we can run certain applications on this ARM system and we can offload some processes from the host. For example, we can run an IDS on the ARM system that transparently detects and filters malicious traffic going to the host.

4.2 IDS Optimization Techniques

4.2.1 DPDK: Data Plane Development Kit

DPDK is a set of libraries and drivers to optimize packet processing by bypassing kernel space and process packets in user space. When using DPDK, the ports of the NIC need to be unbound from the kernel driver and bind to a DPDK-compatible driver [11]. The DPDK drivers then make it possible to interact with the ports of the NIC in user space, and kernel space is bypassed. Traditionally, packets are processed in the Linux kernel networking stack using interrupts (IRQs). The DPDK driver is a poll mode driver (PMD), which instead of interrupts, always is polling for incoming packets [12]. This poll mode driver in combination with the bypassing of the kernel results in better performance when processing packets. Currently, there are some projects going on where DPDK is used for IDS software like Snort and Suricata [13][14]. However, when installing these applications on the DPU, we encountered issues regarding the compilation of the software and segmentation faults when running the software. Furthermore, OvS can be officially built with DPDK to use the DPDK libraries to operate entirely in user space [15]. In our research, we successfully installed OvS with DPDK on the BlueField DPU. However, when unbinding the ports of the BlueField NIC from the kernel and binding it to a DPDK-compatible driver, OvS would not recognize the added ports as DPDK capable and couldn't use the ports of the BlueField NIC.

4.2.2 eBPF (Berkley Packet Filter) & XDP (Express Data Path)

While bypassing kernel space packet processing achieves high performance as DPDK does, this technique also has a disadvantage. Because the kernel space is bypassed, the networking functionality provided by the kernel can no longer be used. This means that user space programs need to re-implement this functionality instead of letting the kernel take care of the processing [16]. With XDP (Express Data Path), kernel functionality can still be used, while also achieving high performance regarding packet processing. Instead of bypassing kernel space into user space, XDP does the opposite by moving user space networking programs into kernel space [16]. eBPF (enhanced Berkley Packet Filter) allows connecting to kernel hook points, where an user-supplied program can be executed when the kernel reaches such a hook point. XDP is an eBPF program that operates in the kernel network code. It adds an early hook in the RX path of the kernel in the NIC driver after the interrupt processing when a packet comes in. At this hook, a user-supplied eBPF program is executed [17]. The advantage of executing user-supplied programs at these early hooks in the RX path is that decisions can be made before certain code in the Linux networking stack is executed. When XDP passes packets to an eBPF program, packets can be, for example, dropped, redirected, or passed to the normal network stack. XDP provides a set of predefined actions to process the packets, like XDP_DROP, XDP_REDIRECT, or XDP_PASS [18]. Suricata can be built with eBPF and can use eBPF and XDP programs to bypass kernel functionality. The advantage of using eBPF and XDP in Suricata is that decisions on packets can be made at the earliest stage [19]. If a packet is malicious it can be dropped before the packet is processed by the kernel. To run Suricata with XDP, the kernel needs to support XDP and a network card is needed that supports XDP in the driver [19]. When running Suricata with XDP on the DPU, this resulted in segmentation faults. This was due to a missing XDP module in the kernel of the DPU.

4.2.3 DOCA and Deep Packet Inspection

DOCA is a software framework by Nvidia to develop applications that use the BlueField DPU for its offloading and acceleration capabilities. While it is possible to access drivers directly to use the offloads and accelerators on the DPU, the main benefit of DOCA is that it provides the use of the drivers at a higher and more abstract level. DOCA consists of an SDK and a runtime platform for the DPU. The SDK contains APIs, libraries, developer

tools and reference application sources to create applications for the DPU. The runtime platform includes some reference application executables and runtime tools to run on the DPU. DOCA applications can be run on the host, on the DPU, or a combination of both [20]. One of the DOCA libraries is the DPI: Deep Packet Inspection library. With DPI you can examine the full content of packets traversing a monitored checkpoint to, for example, identify and block malicious traffic [21]. The DPI libraries are running on DPDK libraries and the RegEx matching engine on the DPU [20]. This makes DOCA with the DPI libraries useful for the optimization of IDS software to identify and filter malicious traffic more efficiently. Unfortunately, because the BlueField-1 is missing some hardware acceleration components, like the RegEx acceleration, DOCA is not supported on the BlueField-1 but only on the BlueField-2 [22].

4.3 Cisco Realistic traffic generator

The Cisco Realistic traffic generator (Trex) is an open-source and low-cost traffic generator that is built upon the Data Plane Development Kit (DPDK) [23]. Alongside DPDK, Trex uses Scapy [24] to build the packets. With these core components, Trex can generate traffic up to 200Gbps on a single server. To generate traffic Trex offers three operational modes, which are: stateful, stateless and advanced stateful mode.

4.3.1 Stateful mode

The stateful mode of Trex is a flow generator the uses pre-processing and smart replay of real traffic templates. This is achieved by using one or more pcap files, which can contain TCP or UDP flows. These flows can be replayed multiple times with new addresses and port numbers from the client and server [25]. While Trex is able to replay traffic from a pcap file, it does not implement a full TCP/IP stack. This means that Trex only replicates the pcap with manually rebuild stateful flows. If a TCP/IP stack is required, the advanced stateful mode needs to be used. Which is explained in 4.3.3.

To generate traffic based upon pcap files, Trex requires a YAML file. This file defines some important parameters. The following parameters are required: IP address range, port range and location of the pcap files. Every defined pcap file can only contain one flow. For this reason, to create a reasonable amount of traffic, multiple pcap files can be defined in the same configuration. Together with the connections-per-seconds (cps) parameter, the total amount of bandwidth can be determined. When executing the YAML configuration, Trex allows for a CLI argument "-m" (multiplier). This multiplier allows for settings a target bandwidth, such that Trex generates e.g. 10Gbps. [25]

Because Trex is able to replay pcap files, a great number of different protocols can be used to generate traffic. By default, Trex includes a variety of test cases that are able to generate different types of traffic, e.g. VoIP, HTTP, Video streaming, Internet-Mix (Imix) and Extended internet Mix (Emix). The Imix test case is a case that is based upon actual real traffic that is captured from some internet core routers, which aids in being a scenario where real-world traffic is required. Emix is the extended version of Imix, which enables the Trex user to generate more flows that carry a higher bitrate. [23]

As mentioned Trex can recreate flows that are captured within a pcap file. If a pcap that contains UDP traffic is used, Trex will create the stateless flow according to the definition in de YAML file. This function is called mimicking of stateless flows. However, in practice, Trex keeps generating the flow with each time a different IP address and different port. This means that Trex treats this type of traffic as a TCP flows. This means that it will keep state and store the information in memory. This means that all kinds of protocols can be used to generate packets that can be manually crafted by using Scapy. [25]

4.3.2 Stateless mode

In stateless mode, Trex is focused on generating streams of packets without keeping state. Trex can generate between 10 and 22 Mpps per CPU core. These generated packets can be created using different traffic profiles, which can be used simultaneously on a single interface. However, there is a limit of 10000 parallel streams, so multiple profiles are required when more streams are required. For a stream to work, the following needs to be defined: Packet template, Field Engine program, Mode, Rate and the Action. All these define what the stream(s) are made upon. For example, The protocol, the L2 or L3 addresses, the size of the data within the packet, continuous or burst stream and the rate at which the packets need to be sent. [26]

4.3.3 Advanced Stateful mode

Advanced Stateful (ASTF) mode is an Open Systems Interconnection (OSI) layer 7 (L7) traffic generator that has built-in supports for user space TCP stack emulation. This stack is built on top of DPDK to increase performance. With the support of DPDK, the developers claim a maximum achievable bandwidth of 200Gbps. The maximum bandwidth would be achievable when using packet size between 600-bytes and 1500-bytes. The emulation of L7 open the possibility to test different devices with different features, e.g. router, firewall, switches and deep-packet-inspection (DPI) engines. Like stateful- and stateless mode ASTF uses python scripts to execute the defined tests and thus allows for high flexibility in terms of packet processing, testing of functionalities and automation. [27]

4.3.4 Trex User Interface

Trex offers two ways to control the traffic generator, a graphical interface and a command-line tool. The graphical interface offers the full capabilities, e.g. monitoring, planning and executing the traffic generator. The command-line based tool offers to monitor and execute the pre-planned traffic generation test. Using the graphical interface packets can be custom made or pre-generated pcap files can be adjusted, both fit the purpose of the test. Creating packets or adjusting packets in a pcap file means that the user has complete control over every bit of information that is sent, including all the attached headers. These can be added to a stream to tell Trex what information has to be sent. The statistics that follow that execution of the traffic generation can be graphed or exported to a file. [28]

The command-line interface offers to execute pre-planned YAML or python scripts. The statistics that follow can be printed on the command line and show a variety of information, e.g. layer- 1 and 2 bandwidth, packets per second, dropped packets, line-rate percentage and much more. These statistics cannot be exported to a file or be graphed, for this the user will need to add or write a custom monitoring script in python that is executed alongside the used traffic definition file. [28]

4.4 GENESIDS - idseventgenerator

Erlacher et al. has presented a novel system for generating attacks based upon the Snort rule base [29]. In their research they have shown the ability to recreate, OSI layer- 4 till 7, packets that can be used in an HTTP based attack scenario [1]. Not only is their application able to generate the necessary packets, but the application is also able to transmit the packets to the target. This enables the possibility to create a single pcap file, which can be used by a traffic generator for testing network devices.

[1] https://github.com/felixe/idsEventGenerator

5 Methodology

This section outlines our methodology. First, we will describe our experimental setup, then we will cover which IDS configuration we analysed. Next, we will describe the different experiments that were conducted in our research, and last we will describe the data set that has been used to perform the experiments.

5.1 Lab setup

For our experimental setup, we used two servers using commodity hardware. The first server, the traffic sender, contains 32GB of ram, has a Ryzen 9 5900X and runs the Ubuntu 20.04 operating system. The other server, the traffic receiver, uses an Intel i5-7500, has 16GB of ram and runs the Ubuntu 20.04 operating system. Both systems contain BlueField-1 ConnectX-5 SmartNICs. These BlueField-1 SmartNICs contain eight dual-core Cortex-A72 ARM processors, with 16GB of ram and 16GB of storage. The BlueField SmartNICs run on firmware version 18.30.1004, and use a by Nvidia provided Ubuntu 20.04 operating system. Both BlueField ConnectX-5 cards contain two 100GbE QSFP28 interfaces. To connect these cards, we used two 2 meter long Direct Attach Copper (DAC) cables. The driver version used on both BlueField ConnectX-5 cards is the MLNX OFED version 5.0-2.1.8.0.1.

Figure 2 shows an overview of the experimental setup. The host on the left is the traffic sender, where the BlueField operates in separate mode. The host on the right is the traffic receiver, where the BlueField operates in embedded mode so that all traffic goes through the Data Processing Unit (DPU) where an IDS, Suricata, will be installed. To forward the incoming traffic coming in on the BlueField to the host, OvS is installed on the DPU.

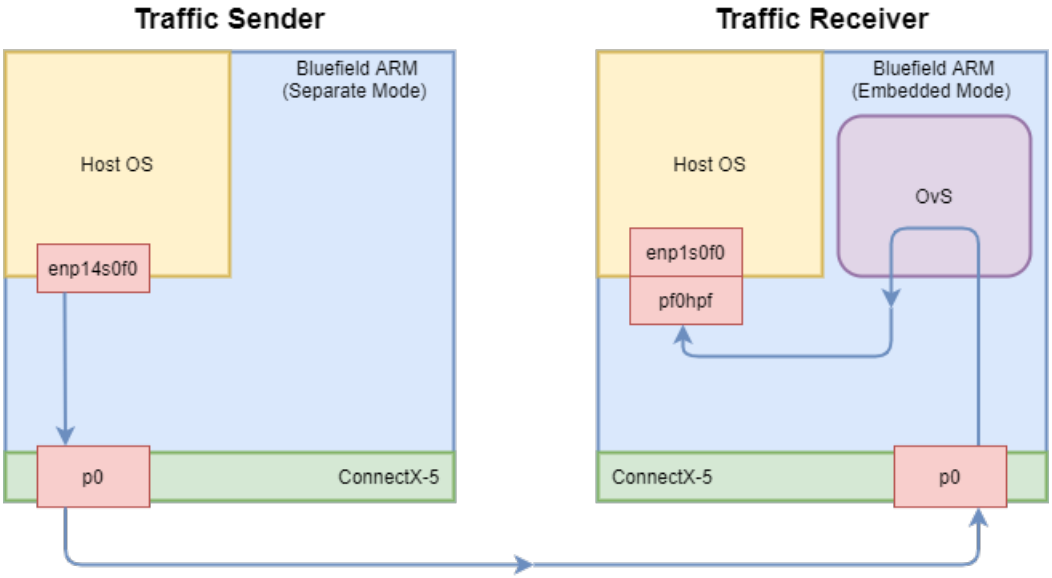


Figure 2: Overview of the basic experimental setup

5.2 Intrusion Detection Software

The Intrusion Detection Software (IDS) software used for this research is Suricata version 6.0.2. Figure 3 illustrates the placement of Suricata within our experimental setup. This shows that Suricata is installed on the DPU. Note that because of the issues of running Suricata with optimizations like DPDK and XDP, plain Suricata without optimizations is

used for the experiments. The ruleset that is used during the experiments is the Emerging Threats OPEN Ruleset [30], containing 22422 rules that are successfully loaded from a rule file when running Suricata. Besides this ruleset, two custom rules are added, shown in listing 1. These rules match any TCP and UDP packet, which means that every packet going through the DPU should be matched by Suricata and therefore should be alerted or dropped.

We experiment with Suricata in two modes: IDS and IPS mode [31]. Suricata in IDS mode runs alongside OvS and is only monitoring traffic. In this mode, Suricata can only alert when traffic is matched against a rule in the ruleset. In IPS mode, Suricata is active between the interface of the BlueField and the host and forwards the traffic itself. Because this makes OvS no longer needed in this mode, OvS is not running here. Besides alerting matched traffic, Suricata in IPS mode is also able to drop packets that are matched against rules in the ruleset. To make Suricata drop all packets, the 'alert' keyword in listing 1 is changed to 'drop'.

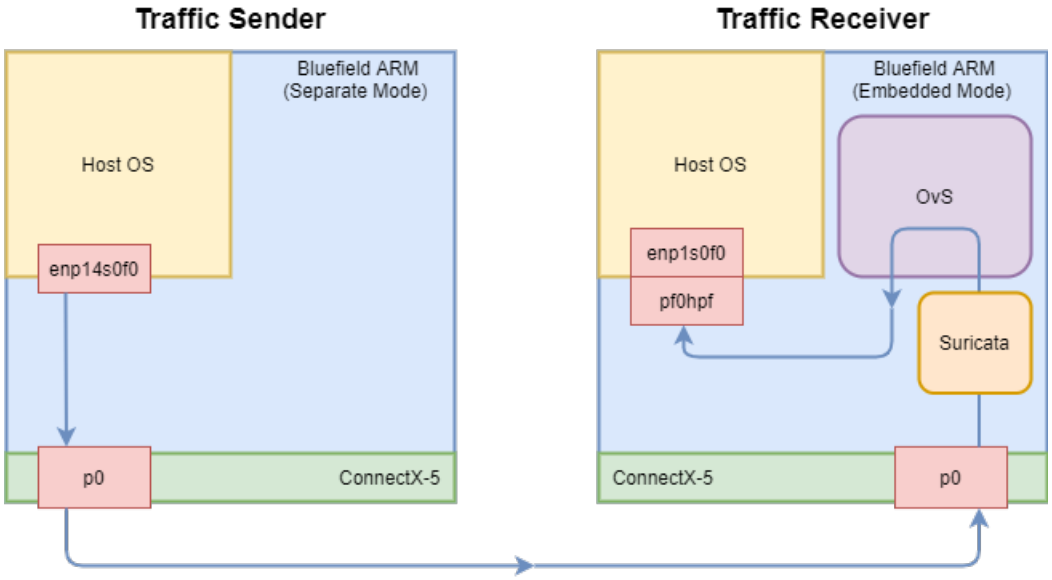


Figure 3: Overview of the experimental setup with Suricata installed on the receiving DPU in embedded mode

```
alert tcp any any -> any any (msg: "TCP"; flow: to_server;sid:3113;)
alert udp any any -> any any (msg: "UDP"; flow: to_server;sid:1337;)
```

Listing 1: Custom rules added to Suricata to match every TCP and UDP packet that is going through the DPU

5.3 Experiments

In this section, there is a detailed description of all the experiments that we preformed during this research. We have separated our experiments by transport protocol, by Maximum Transmission Unit (MTU), with the IDS enabled and disabled, and packet forwarding via Suricata or OpenVswtich (OvS). For all our experiments, the detection system will use AF_PACKET mode. Using AF_PACKET, all traffic will be intercepted at L2. This way the detection system can detect malicious activity transparently. Secondly, all the measurements will be taken using a MTU of 1500 and 9000. This is based on the information of Mohsin and Farid, from which we determined that increasing the MTU could yield an

increase in performance from 10Gbps and up [32]. Third, for the tests mentioned above, we will measure the DPU- and memory utilization.

In our first experiment we will send UDP and TCP based traffic using OvS. This means that Suricata will not be running during the experiments. For our second experiment, we will keep the setup as the first experiment, but now Suricata will be enabled in IDS mode. This means that OvS will forward the traffic through Suricata to analyze the traffic. In our third experiment, we will disable OvS and change the detection mode to prevention. This causes Suricata to intercept and forward the traffic to the host. Additionally, the detection rules will be set to alerting. The last experiment will change the detection rules from alerting to dropping. All our measurements will be taken when sending throughputs of 1-, 10-, 25-, 50-, and 100 Gbps.

Table 1 below shows an overview of the experiments including the types of traffic if Suricata is enabled and in what mode, what software does the forwarding from the BlueField to the host, and if Suricata drops or alerts matched packets.

Table 1: Overview of the experiments

Traffic	Suricata	OvS	Suricata Mode	Alert / Drop	Forwarding to Host
UDP, TCP	Disabled	Enabled	-	-	OvS
UDP, TCP	Enabled	Enabled	IDS	Alert	OvS
UDP, TCP, Malicious	Enabled	Disabled	IPS	Alert	Suricata
UDP, TCP, Malicious	Enabled	Disabled	IPS	Drop	Suricata

5.4 Traffic dataset

Using figure 3, on the traffic sender side, we run Cisco TRex on the host [33] to send TCP, UDP, and generated realistic malicious traffic. The TCP and UDP traffic are sent with random data with a throughput of 1-, 10-, 25-, 50-, and 100 Gbps. The realistic malicious traffic is a pcap containing packets that are generated using the application GENESIDS [29]. This pcap consists of packets that are recreated using the "Emerging Threats" open ruleset of Suricata [30]. To create the pcap file we loaded the ruleset into GENESIDS. GENESIDS creates the packets and sends these to a capture system, which runs an instance of *TCPdump*. The *TCPdump* instance captures the packets and saves the packets in a pcap file. Because this capture file is based upon the ruleset within Suricata, this matches the packets of the malicious traffic that is sent. The generated pcap is replayed multiple times within TRex where the inter-packet gap can be varied resulting in higher or lower throughput.

5.5 Gathering results

We measure the incoming throughput on two interfaces. The first incoming interface is 'p0'. This is the interface where the traffic comes in on the BlueField. The second interface is 'enp1s0f0'. This is the interface on the host, where the traffic has gone through the DPU and is forwarded to the host. The interfaces can be seen in figure 3 on the receiver side. Measurements are taken by reading network interface related statistics with *sysfs*. This is done because it does not rely on the *libpcap* library, and therefore is fast enough to support measuring multi-gigabit rates [34]. The measurement script can be found in listing 2 in Appendix A. To measure the DPU and memory utilization, the application *sar* is used. This application is combined with a custom bash measuring script that gathers the information during the experiments. The gathered information is saved in a text file. From the gathered information we have created several figures using the diagram feature in Google sheets.

6 Results

6.1 Suricata in IDS Mode

Figures 4 and 5 show the incoming bandwidth received on the BlueField and the host, when Suricata is disabled and enabled for UDP traffic. In figure 4 a MTU of 1500 is used and in figure 5 a MTU of 9000 is used. For both MTUs we see that the traffic sent is not equal to the traffic received on the BlueField with both Suricata disabled and enabled. Furthermore, we can see that the incoming bitrate on the host, when traffic has gone through the DPU, is significantly lower than the incoming bitrate on the BlueField, both when Suricata is disabled and enabled. When sending traffic, with a throughput of 10 Gbps to 100 Gbps, the host only receives a maximum bitrate of approximately 8 Gbps when using a MTU of 1500. This increases to approximately 23 Gbps when using a MTU of 9000. What we also observe is a slight difference in the incoming bitrate on the host when Suricata is disabled or enabled. This is a difference of approximate 2 Gbps.

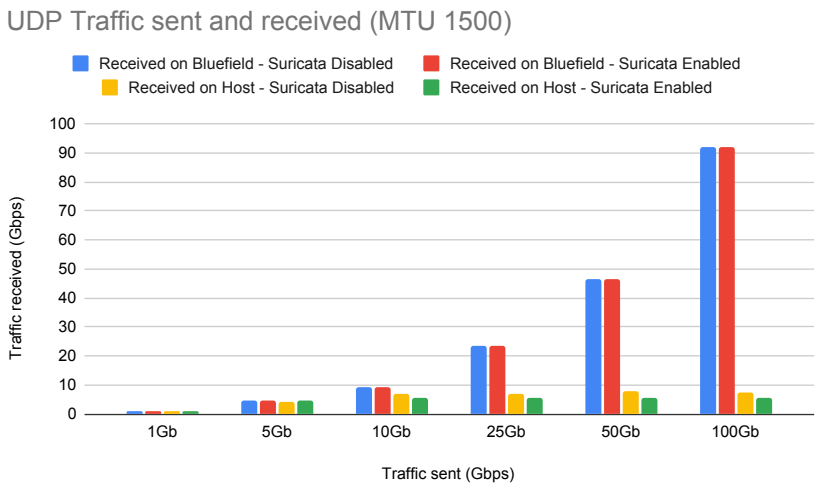


Figure 4: UDP traffic sent and received with a MTU of 1500. When Suricata is running, it runs in IDS mode alongside OvS and is only alerting.

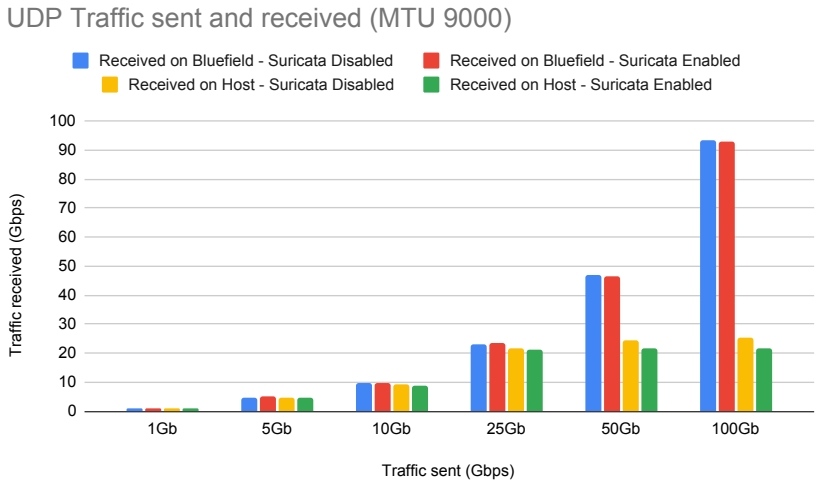


Figure 5: UDP traffic sent and received with a MTU of 9000. When Suricata is running, it runs in IDS mode alongside OvS and is only alerting.

Figures 6 and 7 show the incoming bandwidth received on the BlueField and the host when Suricata is disabled and enabled for TCP traffic. Here we see the same results as with the UDP traffic. That is the sending bitrate is not equal to the receiving bitrate on the BlueField for both MTUs. Additionally, the incoming bitrate on the host is significantly lower than on the BlueField. Comparing the measurements we can observe a couple of key differences. These differences are, using TCP traffic with Suricata enabled, the incoming bitrate on the host is higher when compared to the measurements with Suricata disabled. Using a MTU of 9000, using TCP when Suricata is enabled, this bitrate is more than doubled.

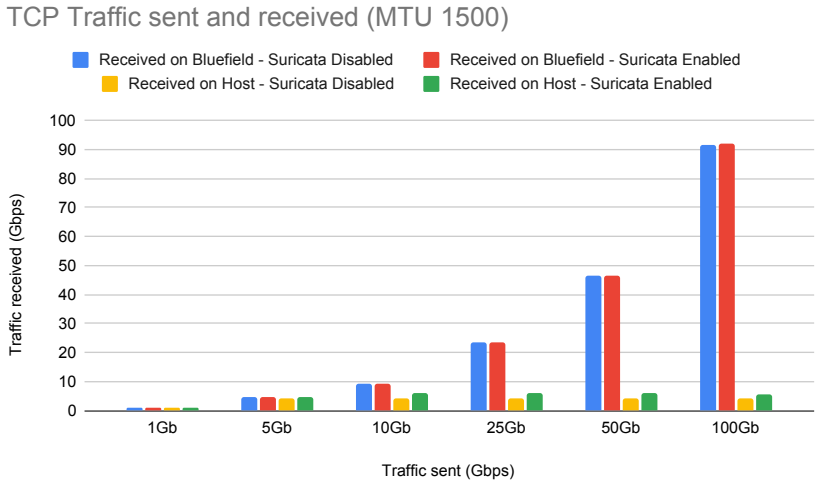


Figure 6: TCP traffic sent and received with a MTU of 1500. When Suricata is running, it runs in IDS mode alongside OvS and is only alerting.

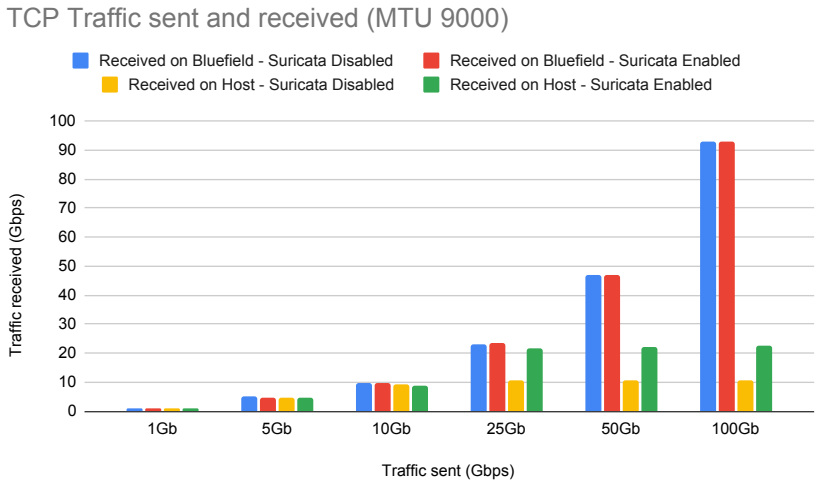


Figure 7: TCP traffic sent and received with a MTU of 9000. When Suricata is running, it runs in IDS mode alongside OvS and is only alerting.

Figure 8 shows the DPU utilization when Suricata is disabled and only OvS is running for TCP and UDP traffic with a MTU of 1500. When sending 1Gbps TCP and UDP traffic, we see that the DPU is less than 20% utilized. When sending 5 Gbps TCP and UDP traffic, the DPU utilization is approximately 30%. When sending 10 Gbps and higher, the DPU utilization is between 40% and 50%. Figure 9 shows the DPU utilization when we enable Suricata in IDS mode. Here we also see that for 1 Gbps TCP and UDP traffic the utilization is 20%, with peaks to 50%. Sending with 5 Gbps or higher results in an utilization of approximately 50%. When using a MTU of 9000 we see the same results, but the maximum DPU utilization on average is 11,75% higher. Figures 10 and 11, we see the memory usage of the BlueField when Suricata is disabled and enabled. The memory usage when Suricata is disabled en enabled is 46%, and between 54% and 58% respectively. When Suricata is enabled, the memory usage increases slightly when the throughput increases.

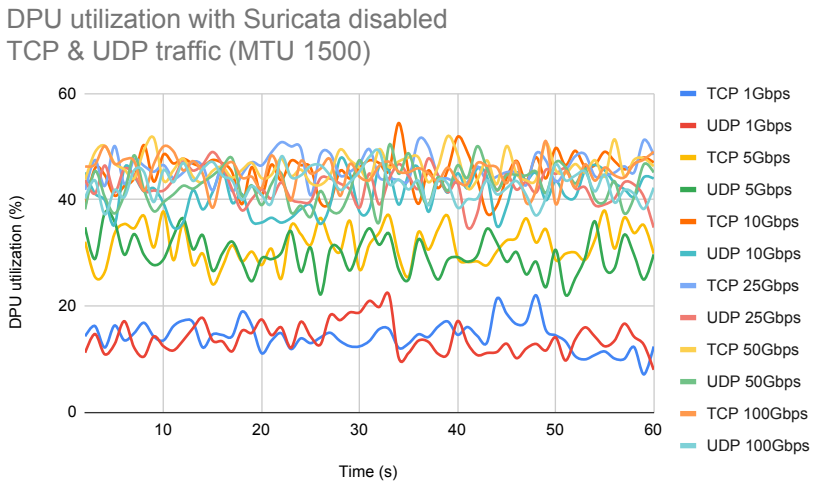


Figure 8: DPU utilization when Suricata is disabled and OvS enabled. Sending TCP & UDP traffic with a MTU of 1500.

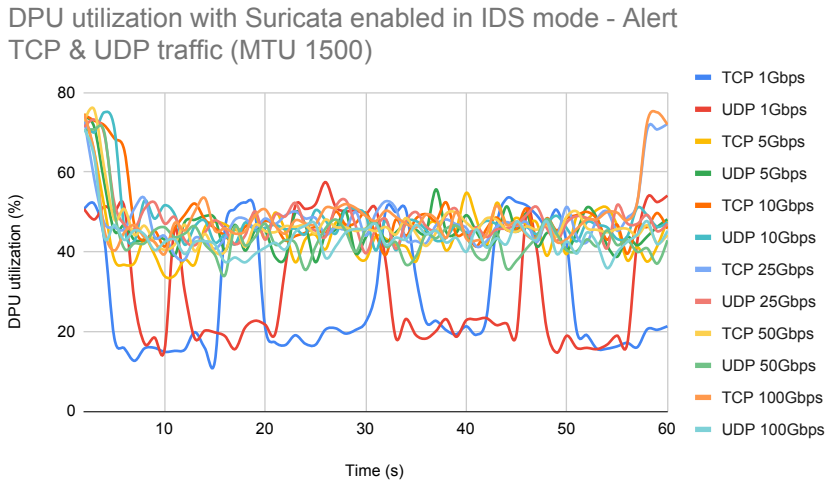


Figure 9: DPU utilization when Suricata is enabled in IDS mode running alongside OvS. Sending TCP & UDP traffic with a MTU of 1500.

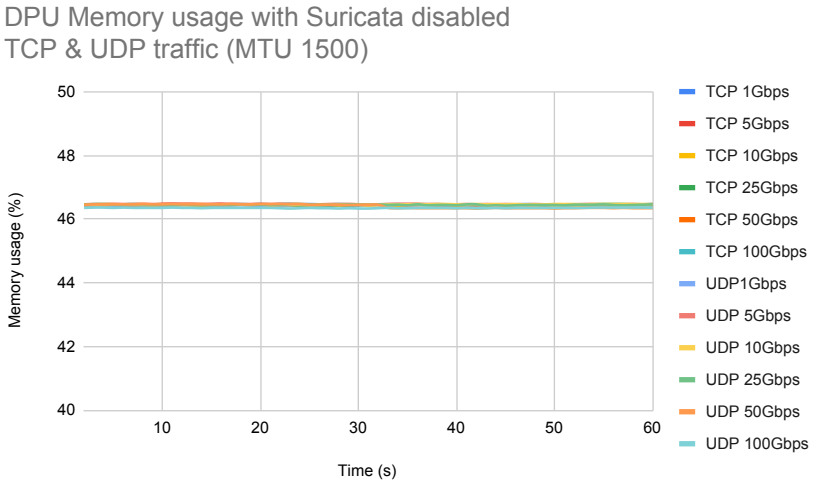


Figure 10: Memory usage when Suricata is disabled and OvS enabled. Sending TCP & UDP traffic with a MTU of 1500.

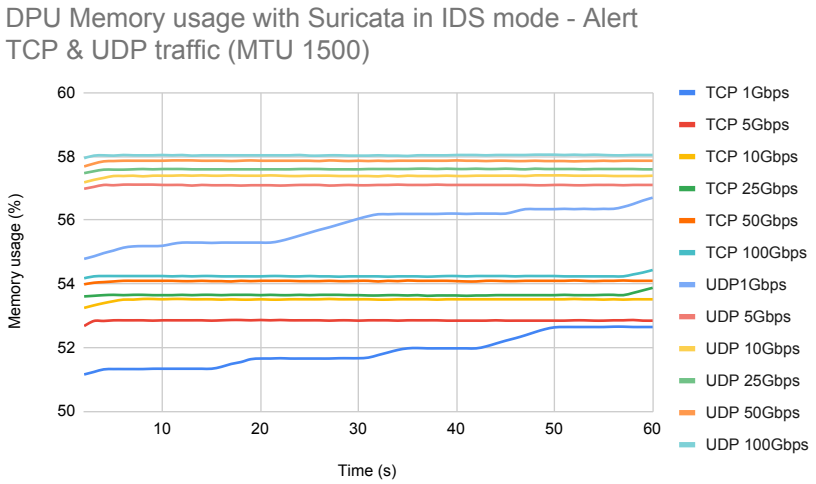


Figure 11: Memory usage when Suricata is enabled in IDS mode running alongside OvS. Sending TCP & UDP traffic with a MTU of 1500.

6.2 Suricata in IPS Mode - Alerting

6.2.1 UDP & TCP Traffic

Figure 12 shows the incoming bandwidth received on the BlueField and the host when sending 1 Gbps of TCP & UDP traffic using a MTU of 1500. Here, Suricata is enabled in IPS mode and only alerts matched packets, which means that all packets should reach the host. We can see that in the first seconds, the BlueField receives 1 Gbps of traffic and start declining. Throughout the test the incoming bitrate declines to 749- and 767 Mbps for TCP and UDP respectively. Which is a loss of 25% and 23% respectively. On the host, the incoming bitrate is volatile and drops several times to zero. The maximum bitrate received on the host is approximately 300 Mbps, with an average of 135 Mbps and 117 Mbps respectively. When we generate traffic with a higher throughput, we see the same results. The maximum bitrate received on the host stays at approximately 300 Mbps. However, after a decline in the incoming bitrate on the BlueField, we see the incoming bitrate increasing to the sending rate again. This is shown in figure 13, where 100 Gbps is sent instead of 1 Gbps. When using a MTU of 9000, we observe the same behaviour. The only difference when using a MTU of 9000 is that the maximum incoming bitrate on the host is now approximately 2 Gbps instead of approximately 300 Mbps.

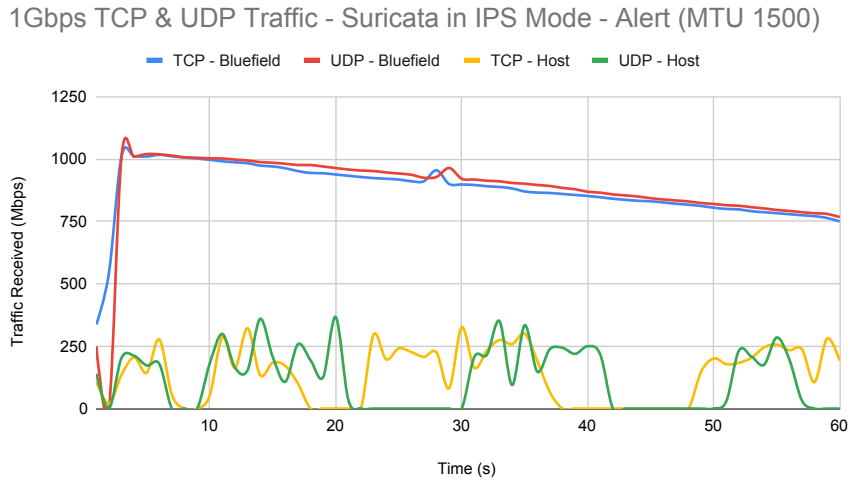


Figure 12: TCP and UDP traffic with a MTU of 1500 received on the BlueField and host. Suricata is running in IPS mode and is only alerting.

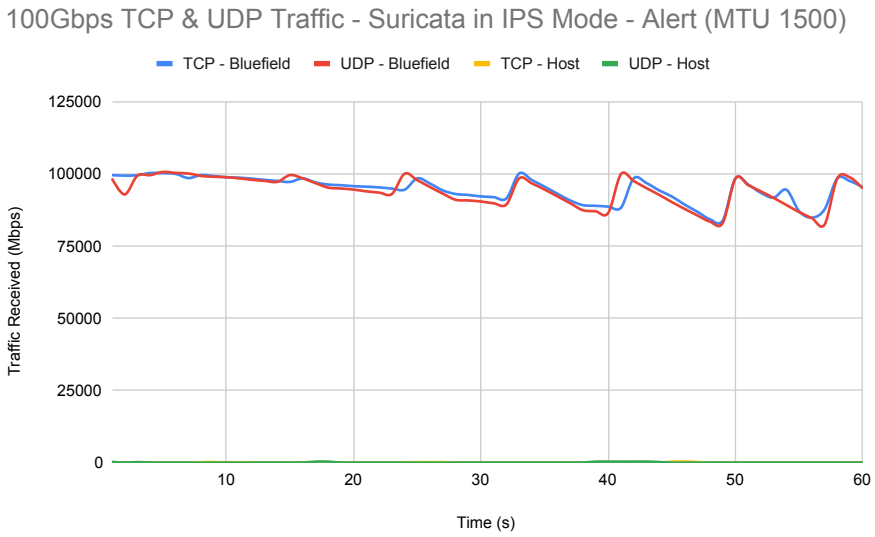


Figure 13: TCP and UDP traffic with a MTU of 1500 received on the BlueField and host. Suricata is running in IPS mode and is only alerting.

In figures 14 and 15 we can see the DPU- and memory utilization when sending TCP and UDP traffic and Suricata running in IPS mode. The DPU utilization figure shows consistency irregularities. During all the experiments the DPU is never 100% utilized. When overlaying figure 12 with figure 14, there appears to be a link with the utilization and the peaks and valleys of the throughput measurements. However, this is not always the case. Comparing the memory utilization with the throughput and the DPU utilization, we obverse an increase in memory utilization each time the throughput increases. From the first 1 Gbps to the last 100 Gbps experiments, we can observe an increase in memory usage.

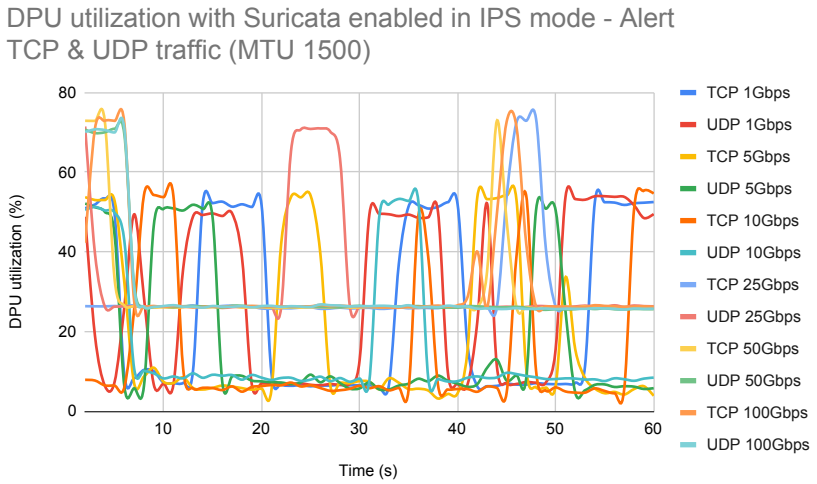


Figure 14: DPU utilization when Suricata enabled in IPS mode, only alerting. Sending TCP & UDP traffic with a MTU of 1500.

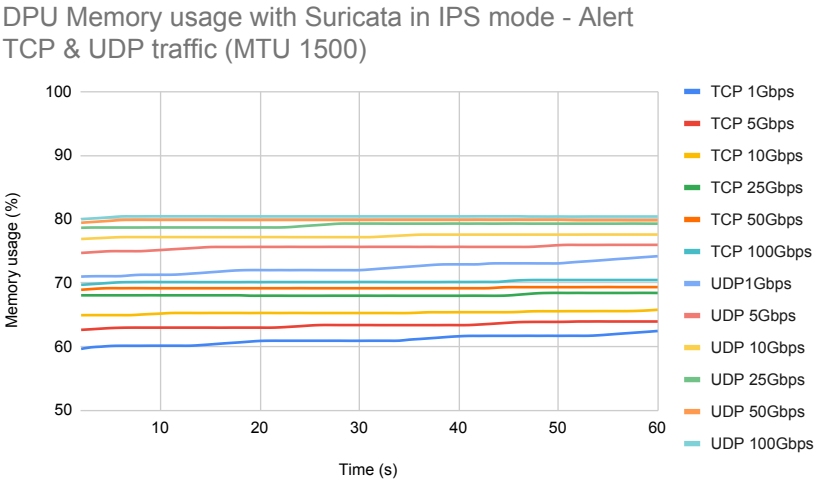


Figure 15: Memory usage when Suricata enabled in IPS mode, only alerting. Sending TCP & UDP traffic with a MTU of 1500.

6.2.2 Malicious Traffic

Figure 16 shows our throughput results when testing the transparent IPS with malicious traffic. For this test we used our custom made pcap file. This pcap file contains 19967 traffic streams, which is just below that maximum of 20000 streams that Trex supports. To reiterate Suricata is in IPS mode and alerts matched packets. The before mentioned figure illustrates the incoming bitrate on the BlueField and the host when sending 1 Gbps of the generated malicious traffic. We see that the incoming bitrate on the BlueField is very volatile, compared to the results in figure 12. However the opposite is also true for the the maximum incoming bitrate at the host, this time the bitrate stays stable at 250 Mbps. Looking at the average incoming bitrate of figure 16, we see that the average bitrate increases by 185%.

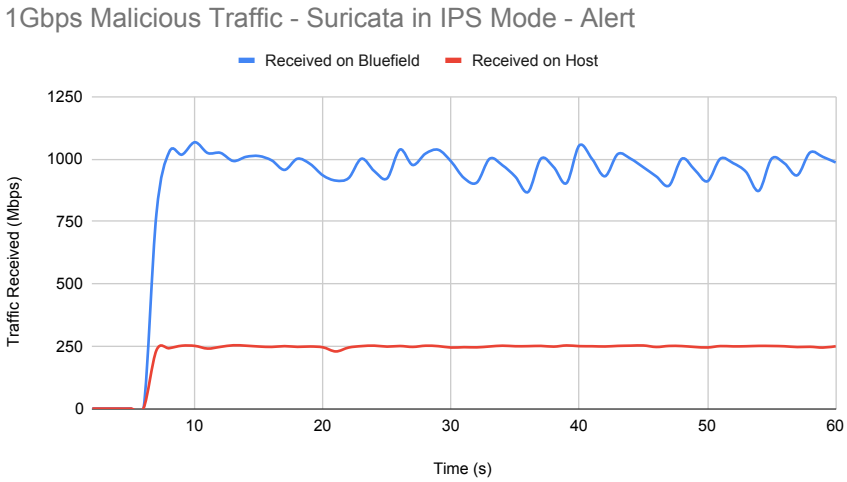


Figure 16: Malicious traffic received on the BlueField and host. Suricata is running in IPS mode and is only alerting.

Comparing figure 17 against figure 16, we observe the following. As described in sections 6.2.1 we see significant peaks and valleys in the DPU utilization. The utilization is mostly consistent, with occasionally a significant peak of 50% in the DPU utilization. However, without the peaks, the utilization is consistent with the utilization of figure 8. However, during the 1 Gbps throughput experiment, we did observe a slight increase in memory utilization. After all the experiments figure 18 shows the memory is 86% in use. This means that 13.76 Gigabyte is being used by the system. Which for a Linux based machine with Suricata as the only running application, is rather high.

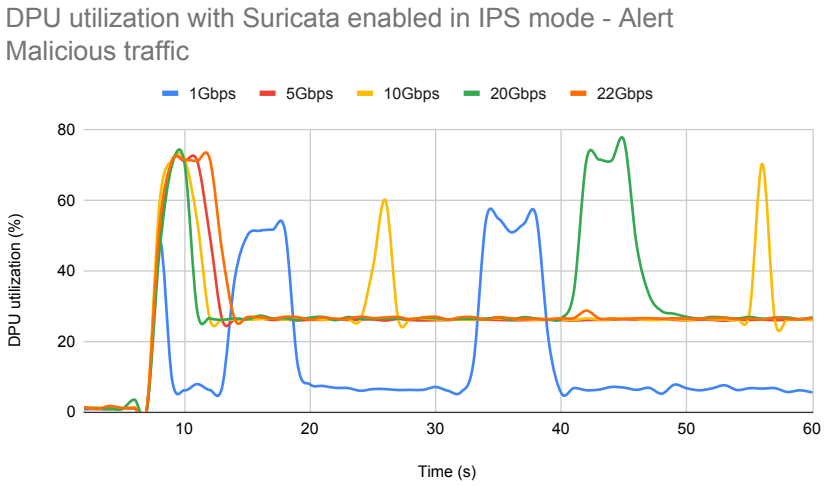


Figure 17: DPU utilization when Suricata enabled in IPS mode, only alerting. Sending malicious traffic.

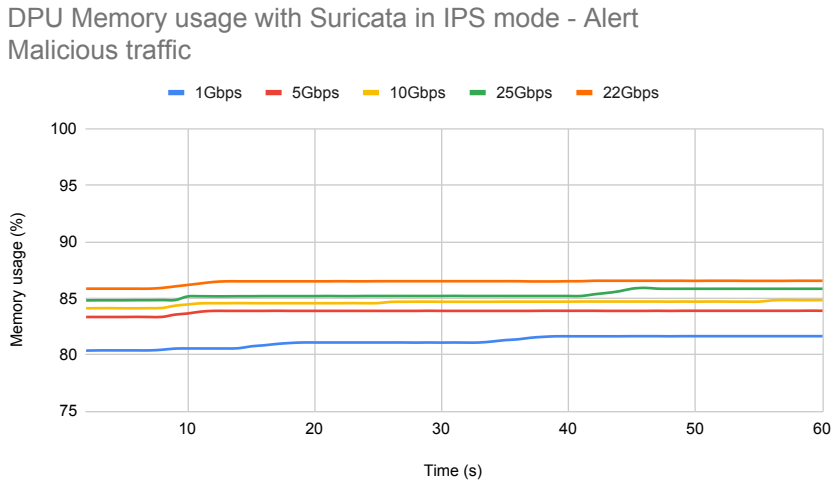


Figure 18: Memory usage when Suricata enabled in IPS mode, only alerting. Sending malicious traffic.

6.3 Suricata in IPS Mode - Dropping

When changing the Suricata rules from 'alert' to 'drop' and sent TCP and UDP traffic, we observed that every packet was dropped on the BlueField and no packets were incoming on the host, as expected. This was also the case when sending the malicious traffic. However, we sometimes observed small bursts of a couple of megabits incoming on the host. This is shown in figure 19. When we look at the DPU utilization and memory usage, we see the same results as in figures 17 and 18, where Suricata alerts instead of drops matched packets.

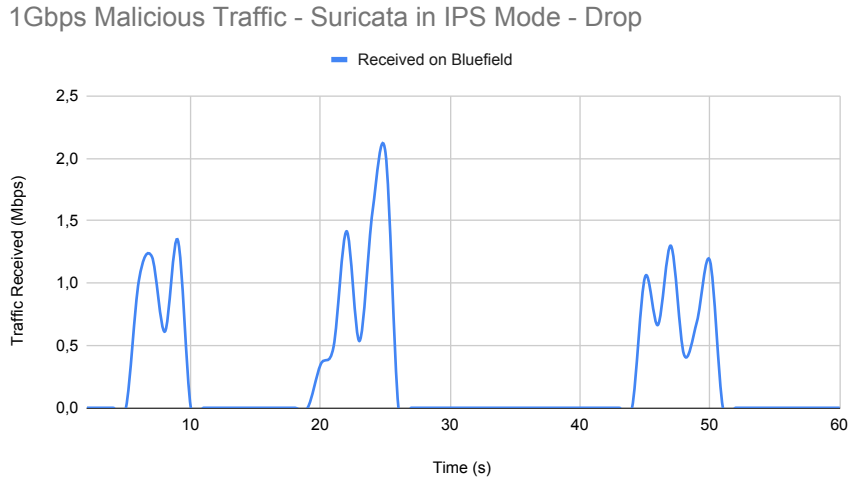


Figure 19: Malicious traffic received on the BlueField and host. Suricata is running in IPS mode and is only alerting.

7 Discussion

During this research, we experienced problems and made specific choices that are relevant to discuss. The first and largest issue was being unable to fully utilize the offloading capabilities the BlueField-1. During the length of the research, we repeatedly tried to get the offloading and packet processing in user space working. We especially invested a large amount of time in Suricata with DPDK and XDP, which would have a significant positive influence on our results. This means that Suricata needed to use the DPDK libraries. To get this to work we tried many different Suricata and DPDK combinations, compiled our own kernel and kernel module to enable XDP. Because this caused segmentation faults when running Suricata, we started Suricata without the optimizations. Because OvS would not recognize the Mellanox 100Gb NICs as DPDK capable, we also used OvS without the DPDK optimizations. The choice to work without the optimizations worked well. However we were able to install different Mellanox OFED versions on the BlueField, but the necessary acceleration libraries would not load. Even without hardware acceleration of Perl Compatible Regular Expressions (PCRE) support. Since Suricata relies heavily on PCRE for its detection mechanism. With the variations in our results and the dependency on solutions like DPDK, XDP and hardware-accelerated PCRE, we believe that the results would have been higher [35]. Still, even without the hardware acceleration, missing acceleration modules and unrecognized interfaces we were able to achieve a stable 250 Mbps of throughput. Where Suricata would perform all the operations needed, this means that we did not use separate channel, like *mirror ports* to analyze the traffic.

Also, it is worth mentioning that during our research and having to search for information, we had to figure out how to obtain and set up the required resources. This is because there seems to be a lack of working documentation. Both on the Nvidia (Mellanox) developer website and the web in general. During the research, we found that most of the documentation is written for the BlueField-2, which has significant improvements and better support over the BlueField-1. Unfortunately, during our research, the BlueField-2 were already ordered but until the day of writing not delivered. According to a Nvidia representative, not a single BlueField-2 is currently in the Netherlands. Working with a BlueField-2 would have given us better results. These improved results would be from the upgraded support, enabling hardware offloading and hardware acceleration for PCRE, which the BlueField-1 doesn't support. Nevertheless, using sources like [35], we can still estimate the gain. The performance gain would have been about 400 times our results. Please keep in mind that there is no direct comparison possible. This is because there isn't any research that specifically compared these BlueField cards the way we have tested them. It still is hard to exactly estimate the improvement, due to the different device hardware specifications and a difference in supported accelerators. However, the assumption that there is a significant- and tangible increase in throughput is safe.

Next, when changing the transport protocol to TCP, especially when using a MTU of 9000, we received unexpected results. Comparing the results of UDP and TCP we assumed that the throughput would stay about the same. However, we noticed that the overall throughput went down when using the BlueField as a switch. The high throughput results decreased by a factor of 2. While the results with the IDS enable stayed the same. It would be logical to assume that Suricata processes packets faster when compared to OvS. However neither use any kind of offloading and process the packets using the Linux kernel. It would be interesting to examine what exactly happens with the packets when using a high MTU in combination with Suricata and OvS.

In our related work, we discussed a more efficient approach. This approach involved a mirror port through which the traffic would be replicated and sent to the IDS for analysis. From that point, based on the IDS analysis, an access control entry could be added to the OvS

bridge to allow or deny a traffic flow. We originally had a choice using this transparent approach. However, although it could have yield different results, this approach was already performed by other researchers. From our Suricata research, we knew that Suricata has a transparent mode called "AF_PACKET". With these two pieces of information, we chose to use "AF_PACKET" mode to create a transparent IDS. With the knowledge of a built-in transparent IDS mode, research that already has been performed and limited time, we determined that using less software that works easily would benefit the research. These benefits would be in the form of more results within the given time frame.

8 Conclusion

In this research, we tried to find the limitations of the Nvidia BlueField by answering the following research question:

What are the limitations of the Nvidia BlueField SmartNIC regarding the detection of large amounts of malicious traffic?

First, we looked at the possibilities regarding the optimization of IDS software within the BlueField DPU. We found several techniques, like DPDK, eBPF & XDP, and DOCA. Unfortunately, we encountered some compatibility, compilation and runtime problems when combining these techniques with Suricata on the BlueField DPU. For this reason, we conducted a transparent IDS implementation on the BlueField-1 without optimizations like DPDK and XDP & eBPF. When experimenting with Suricata and OvS, we found that the BlueField was able to receive traffic up to 100 Gbps. However, the largest limitation is the DPU not being able to process traffic at line rate. This is mainly due to OvS and Suricata running without optimizations and the lack of hardware acceleration features to process traffic more efficiently.

In the end, we were able to run a transparent intrusion detection system on the BlueField DPU. While we couldn't use any IDS optimization, Suricata was still able to detect, alert and drop malicious packets. Which is the goal of an intrusion detection system. The limitations of the BlueField are the missing accelerators. These missing accelerators ensure the quick processing of IDS operations. This resulted in the lower throughput received on the host. However, this limitation problem might be solved using the Bluefield-2, which supports optimizations like DOCA and more hardware acceleration features.

9 Future Work

In our research, we worked with the BlueField-1 DPU and found out it has some limitations, like missing support of DOCA and RegEx acceleration hardware. These important features are supported on the BlueField-2. Therefore, future research could be done with the newer BlueField-2 to look at the performance when DOCA with DPI and the RegEx acceleration hardware can be used. Furthermore, the research on the performance of IDS software can be done on other hardware solutions than the BlueField DPU. For example, research could be done on FPGA-based SmartNICs, like the SmartNICs from Napatech [36] and see how IDS software in combination with the researched optimization perform on those cards. Besides using software like Suricata and OvS with optimization techniques like DPDK, XDF & eBPF and DOCA, future research could focus on working with software like VPP [37] and regex optimization like Hyperscan [38] for the x86 architecture. Finally, a few months before our research Nvidia announced the BlueField-3, which contains a GPU. Other research could be performed to detect and filter malicious traffic using this GPU hardware for further optimizations.

10 Acknowledgements

We would like to thank Datadigest for having us for the duration of this research. With special thanks to our supervisor, Cedric Both, for his guidance, insight and support.

References

[1] J. Liu, C. Maltzahn, C. Ulmer, and M. L. Curry, "Performance characteristics of the bluefield-2 smartnic," *arXiv preprint arXiv:2105.06619*, 2021.

[2] NVIDIA, "Nvidia bluefield dpu." https://www.nvidia.com/en-us/networking/products/data-processing-unit/.

[3] S. B. Venugopal, "Automatic generation of access control list on mellanox switch for ddos attack mitigation using ddos fingerprints," 2019.

[4] D. Zhang and S. Wang, "Optimization of traditional snort intrusion detection system," vol. 569, no. 4, p. 042041, 2019.

[5] Q. Hu, M. R. Asghar, and N. Brownlee, "Evaluating network intrusion detection systems for high-speed networks," in *2017 27th International Telecommunication Networks and Applications Conference (ITNAC)*, pp. 1–6, IEEE, 2017.

[6] Q. Hu, S.-Y. Yu, and M. R. Asghar, "Analysing performance issues of open-source intrusion detection systems in high-speed networks," *Journal of Information Security and Applications*, vol. 51, p. 102426, 2020.

[7] J. Yang, L. Jiang, X. Bai, H. Peng, and Q. Dai, "A high-performance round-robin regular expression matching architecture based on fpga," in *2018 IEEE Symposium on Computers and Communications (ISCC)*, pp. 1–7, IEEE, 2018.

[8] NVIDIA, "Nvidia bluefield-1 smartnic." https://store.mellanox.com/products/nvidia-mbf1l516a-cscat-bluefield-smartnic-dpu-100gbe-dual-port-qsfp28-pcie-4-0-x16-g-serie
html.

[9] I. Levy, "Bluefield smartnic modes." https://community.mellanox.com/s/article/BlueField-SmartNIC-Modes, 2019.

[10] NVIDIA, "Modes of operation." https://docs.mellanox.com/display/BlueFieldSWv35111601/Modes+of+Operation.

[11] A. Yemelianov, "Introduction to dpdk: Architecture and principles." https://blog.selectel.com/introduction-dpdk-architecture-principles/.

[12] A. Toonk, "Building a high performance - linux based traffic generator with dpdk." https://toonk.io/building-a-high-performance-linux-based-traffic-generator-with-dpdk/index.html.

[13] Napatech, "Snort dpdk daq." https://github.com/napatech/daq_dpdk_multiqueue.

[14] V. Varghese, "Dpdk suricata." https://github.com/vipinpv85/DPDK_SURICATA-4_1_1.

[15] OvS, "Open vswitch with dpdk." https://docs.openvswitch.org/en/latest/intro/install/dpdk/.

[16] D. P. García, "A brief introduction to xdp and ebpf." https://blogs.igalia.com/dpino/2019/01/07/a-brief-introduction-to-xdp-and-ebpf/.

[17] A. Toonk, "Building an xdp (express data path) based bgp peering router." https://toonk.io/building-an-xdp-express-data-path-based-bgp-peering-router/index.html.

[18] D. P. García, "The express data path." https://blogs.igalia.com/dpino/2019/01/10/the-express-data-path/.

[19] Suricata, "Suricata docs - ebpf and xdp." https://suricata.readthedocs.io/en/latest/capture-hardware/ebpf-xdp.html.

[20] J. F. Kim and A. Kit, "Accelerating solution development with doca on nvidia bluefield dpus." https://developer.nvidia.com/blog/accelerating-solution-development-with-doca-on-bluefield-dpus/.

[21] NVIDIA, "Bluefield dpu sw manual - deep packet inspection." https://docs.mellanox.com/display/BlueFieldSWv35011563/Deep+Packet+Inspection.

[22] j. NVIDIA, "Nvidia developer forums - doca sdk documentation." https://forums.developer.nvidia.com/t/doca-sdk-documentation/174726/5.

[23] Cisco, "cisco-system-traffic-generator trex-core," June 2021.

[24] Scapy, "Scapy packet builder," June 2021.

[25] Cisco, "Cisco trex manual," June 2021.

[26] Cisco, "Cisco trex manual - stateless," June 2021.

[27] Cisco, "Cisco trex manual - astf," June 2021.

[28] Cisco, "Cisco trex manual - ui," June 2021.

[29] F. Erlacher and F. Dressler, "How to test an ids? genesids: An automated system for generating attack traffic," in *Proceedings of the 2018 Workshop on Traffic Measurements for Cybersecurity*, pp. 46–51, 2018.

[30] P. E. Threats, "Et open ruleset." https://rules.emergingthreats.net/OPEN_download_instructions.html.

[31] Suricata, "Setting up ips/inline for linux." https://suricata.readthedocs.io/en/suricata-6.0.0/setting-up-ipsinline-for-linux.html.

[32] M. Khalil and F. U. Khan, "Exploration of tcp parameters for enhanced performance in a datacenter environment," in *2018 4th International Conference on Electrical, Electronics and System Engineering (ICEESE)*, pp. 65–69, IEEE, 2018.

[33] Cisco, "Cisco trex - realistic traffic generator." https://github.com/cisco-system-traffic-generator/trex-core.

[34] D. Nanni, "How to measure packets per second or throughput on high speed network interface." https://www.xmodulo.com/measure-packets-per-second-throughput-high-speed-network-interface.html.

[35] c. lev, "Nvidia mellanox bluefield-2 smartnic hands-on tutorial: "rig for dive" — part vii/b: To offload or not to offload? continued," May 2021.

[36] "Napatech." https://www.napatech.com/.

[37] FD.io, "Vpp." https://wiki.fd.io/view/VPP.

[38] Intel, "Hyperscan." https://github.com/intel/hyperscan.

A Throughput Measurement Script

```
#!/bin/bash

INTERVAL="1"  # update interval in seconds

if [ -z "$1" ]; then
        echo
        echo usage: $0 [network-interface] [duration]
        echo
        echo "e.g. $0 p0 60 | tee -a 1500_bf_udp_alert_1_bps"
        echo
        exit
fi

IF=$1

for i in $(seq $2); do
        R1=`cat /sys/class/net/$1/statistics/rx_bytes`
        sleep $INTERVAL
        R2=`cat /sys/class/net/$1/statistics/rx_bytes`
        RBPS=`expr $R2 - $R1`
        RRBPS=`expr $RBPS \* 8`
        echo "$RRBPS"
done
```

Listing 2: Bash script to measure the incoming bandwidth in bits per second

B DPU and memory utilization measurement script

```
#!/bin/bash

if [ -z "$1" ]; then
        echo
        echo "usage: $0 [mtu (1500/9000)] [udp/tcp] [alert/drop] [size
            (1/10/25...)]"
        echo
        echo "e.g. $0 1500 udp alert 1"
        echo
        exit
fi

MTU=$1
PROTO=$2
ACTION=$3
SIZE=$4

OUTFILE=${MTU}_${PROTO}_${ACTION}_${SIZE}
echo $OUTFILE

sar -u 1 60 | awk '{ print $3+$5 }' | tee -a "${OUTFILE}_cpu" &
sar -r 1 60 | awk '{ print $5 }' | tee -a "${OUTFILE}_mem" &
wait
```

Listing 3: Bash script to measure the DPU and memory utilization