



European
Commission

Horizon 2020
European Union funding
for Research & Innovation

Big Data technologies and extreme-scale analytics



Multimodal Extreme Scale Data Analytics for Smart Cities Environments

D4.2: Security assurance and acceleration in E2F2C framework – initial version[†]

Abstract: This deliverable is the initial version of the security assurance and acceleration in the E2F2C framework and it describes the initial version of the work conducted in Task 4.3, which is part of the WP4. Security assurance is achieved by two components. The first component is named EdgeSec Virtual Private Network (EdgeSec VPN). EdgeSec VPN secures the communications between the different components of the MARVEL platform within the different layers (i.e., Edge, Fog, Cloud). The second component, namely EdgeSec Trusted Execution Environment (EdgeSec TEE), enables confidential and secure execution of applications that process sensitive data. Acceleration is achieved through GPURegex, which is a component that offers GPU-accelerated stream processing. More specifically, GPURegex accelerates the pattern matching procedure. In this deliverable, we discuss about these tools, their relation to the MARVEL project and how they address the objectives and KPIs defined.

Contractual Date of Delivery	30/06/2022
Actual Date of Delivery	30/06/2022
Deliverable Security Class	Public
Editor	<i>Eva Papadogiannaki, Sotiris Ioannidis (FORTH)</i>
Contributors	FORTH, TAU, UNS
Quality Assurance	<i>Borja Sáez (IFAG)</i> <i>Dragana Bajovic (UNS)</i>

[†] The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957337.

The *MARVEL* Consortium

Part. No.	Participant organisation name	Participant Short Name	Role	Country
1	FOUNDATION FOR RESEARCH AND TECHNOLOGY HELLAS	FORTH	Coordinator	EL
2	INFINEON TECHNOLOGIES AG	IFAG	Principal Contractor	DE
3	AARHUS UNIVERSITET	AU	Principal Contractor	DK
4	ATOS SPAIN SA	ATOS	Principal Contractor	ES
5	CONSIGLIO NAZIONALE DELLE RICERCHE	CNR	Principal Contractor	IT
6	INTRASOFT INTERNATIONAL S.A.	INTRA	Principal Contractor	LU
7	FONDAZIONE BRUNO KESSLER	FBK	Principal Contractor	IT
8	AUDEERING GMBH	AUD	Principal Contractor	DE
9	TAMPERE UNIVERSITY	TAU	Principal Contractor	FI
10	PRIVANOVA SAS	PN	Principal Contractor	FR
11	SPHYNX TECHNOLOGY SOLUTIONS AG	STS	Principal Contractor	CH
12	COMUNE DI TRENTO	MT	Principal Contractor	IT
13	UNIVERZITET U NOVOM SADU FAKULTET TEHNICKIH NAUKA	UNS	Principal Contractor	RS
14	INFORMATION TECHNOLOGY FOR MARKET LEADERSHIP	ITML	Principal Contractor	EL
15	GREENROADS LIMITED	GRN	Principal Contractor	MT
16	ZELUS IKE	ZELUS	Principal Contractor	EL
17	INSTYTUT CHEMII BIOORGANICZNEJ POLSKIEJ AKADEMII NAUK	PSNC	Principal Contractor	PL

Document Revisions & Quality Assurance

Internal Reviewers

1. Borja Sáez, IFAG
2. Dragana Bajovic, UNS

Revisions

Version	Date	By	Overview
2.2	29/06/2022	Eva Papadogiannaki (FORTH)	Final document
2.1.1	21/06/2022	Eva Papadogiannaki (FORTH)	Minor revision after IR
2.1.0	16/06/2022	Eva Papadogiannaki (FORTH)	Revision after IR (round 1)
2.0.2	14/06/2022	Dragana Bajovic (UNS)	IR (round 1)
2.0.1	09/06/2022	Borja Saez (IFAG)	IR (round 1)
2.0.0	07/06/2022	Editors & Contributors	Document ready for IR
1.0.0	27/04/2022	Eva Papadogiannaki (FORTH)	Final ToC revision
0.1.3	21/04/2022	Dragana Bajovic (UNS)	Final ToC comments
0.1.2	13/04/2022	FORTH	ToC revision
0.1.1	05/04/2022	Dragana Bajovic (UNS)	Comments on the ToC
0.1.0	21/03/2022	Eva Papadogiannaki (FORTH)	ToC

Disclaimer

The work described in this document has been conducted within the MARVEL project. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957337. This document does not reflect the opinion of the European Union, and the European Union is not responsible for any use that might be made of the information contained therein.

This document contains information that is proprietary to the MARVEL Consortium partners. Neither this document nor the information contained herein shall be used, duplicated or communicated by any means to any third party, in whole or in parts, except with prior written consent of the MARVEL Consortium.

Table of Contents

LIST OF TABLES.....	6
LIST OF FIGURES.....	7
LIST OF ABBREVIATIONS.....	8
EXECUTIVE SUMMARY	10
1 INTRODUCTION.....	11
1.1 PURPOSE AND SCOPE.....	11
1.2 RELATION TO OTHER WORK PACKAGES, DELIVERABLES AND ACTIVITIES	11
1.3 CONTRIBUTION TO WP4 AND PROJECT OBJECTIVES	12
1.4 STRUCTURE OF THE REPORT	13
2 SECURE COMMUNICATION ON THE EDGE (EDGESEC VPN).....	14
2.1 BACKGROUND.....	14
2.1.1 <i>Virtual Private Networks</i>	14
2.1.2 <i>Secure Peer-to-Peer Communications</i>	14
2.1.3 <i>State-of-the-Art</i>	15
2.2 EARLY DEPLOYMENT AND INTEGRATION	16
2.2.1 <i>Development</i>	16
2.2.2 <i>Early Deployment</i>	16
2.2.3 <i>Integration with MARVEL</i>	17
2.3 USE CASES AND RELATED COMPONENTS	18
2.3.1 <i>Related Components</i>	18
2.3.2 <i>EdgeSec VPN in UNS Use Case</i>	19
2.3.3 <i>EdgeSec VPN in GRN Use Cases</i>	20
2.3.4 <i>EdgeSec VPN in MT Use Cases</i>	21
2.4 EARLY EXPERIMENTAL RESULTS.....	22
2.4.1 <i>Testbed Setup</i>	22
2.4.2 <i>Experiments</i>	23
2.4.3 <i>Results</i>	25
2.5 KPIS.....	30
2.5.1 <i>Project-related KPIS</i>	30
2.5.2 <i>Component-related KPIS</i>	31
3 TRUSTED EXECUTION ON THE EDGE (EDGESEC TEE).....	33
3.1 BACKGROUND.....	33
3.1.1 <i>Trusted Execution Environments</i>	33
3.1.2 <i>Containers</i>	34
3.1.3 <i>State-of-the-Art</i>	35
3.2 EARLY DEPLOYMENT AND INTEGRATION	36
3.2.1 <i>Development</i>	36
3.2.2 <i>Early Deployment</i>	37
3.2.3 <i>Integration with MARVEL</i>	38
3.3 USE CASES AND RELATED COMPONENTS	38
3.3.1 <i>Related Components</i>	39
3.3.2 <i>EdgeSec TEE in MARVEL Use Cases</i>	39
3.4 EARLY EXPERIMENTAL RESULTS.....	39
3.4.1 <i>Testbed Setup</i>	39
3.4.2 <i>Experiments</i>	39
3.4.3 <i>Results</i>	39
3.5 KPIS.....	39
3.5.1 <i>Project-related KPIS</i>	40
3.5.2 <i>Component-related KPIS</i>	40
4 GPU-ACCELERATED STREAM PROCESSING ON THE EDGE (GPUREGEX).....	42

4.1	BACKGROUND.....	42
4.1.1	<i>GPU-Accelerated Stream Processing</i>	42
4.1.2	<i>GPU-Accelerated Pattern Matching</i>	42
4.1.3	<i>State-of-the-Art</i>	43
4.2	EARLY DEPLOYMENT AND INTEGRATION	44
4.2.1	<i>Implementation</i>	44
4.2.2	<i>Early Deployment</i>	46
4.2.3	<i>Integration with MARVEL</i>	47
4.3	USE CASES AND RELATED COMPONENTS	47
4.3.1	<i>Related Components</i>	48
4.3.2	<i>GPURegex in MARVEL Use Cases</i>	48
4.4	EARLY EXPERIMENTAL RESULTS.....	48
4.4.1	<i>Testbed Setup</i>	48
4.4.2	<i>Experiments</i>	49
4.4.3	<i>Results</i>	50
4.5	KPIs.....	52
4.5.1	<i>Project-related KPIs</i>	52
4.5.2	<i>Component-related KPIs</i>	52
5	CONCLUSIONS	53
6	REFERENCES.....	54

List of Tables

Table 1: Modified Dockerfile with proxy environment variables	19
Table 2: Packet size 100 bytes	27
Table 3: Packet size 500 bytes	27
Table 4: Packet size 1000 bytes	28
Table 5: Project-related KPIs that concern EdgeSec VPN	31
Table 6: Component-related KPIs that concern EdgeSec VPN	31
Table 7: Project-related KPIs that concern EdgeSec TEE	40
Table 8: Component-related KPIs that concern EdgeSec TEE	41
Table 9: Processing throughput of GPURegex and GNU Grep (measured in Mbits/second)	52
Table 10: Processing time of GPURegex and GNU Grep (measured in microseconds)	52
Table 11: Project-related KPIs that concern GPURegex	52
Table 12: Component-related KPIs that concern GPURegex	52

List of Figures

Figure 1: Conceptual architecture of the whole MARVEL platform	12
Figure 2: The n2n architecture	17
Figure 3: EdgeSec VPN forms a full mesh topology	18
Figure 4: EdgeSec VPN in UNS Use Case	20
Figure 5: EdgeSec VPN in GRN Use Cases	21
Figure 6: EdgeSec VPN in MT Use Case	22
Figure 7: Virtual Machines in Proxmox.....	23
Figure 8: The output of the Super Node execution script	23
Figure 9: The output of the first Edge Node	24
Figure 10: The output of the second Edge Node.....	25
Figure 11: Ping connectivity experiment	25
Figure 12: First edge node pings the public IP of the second edge node with payload 100 bytes.....	26
Figure 13: First edge node pings the VPN IP of the second edge node with payload 100 bytes.....	26
Figure 14: First edge node pings the public IP of the second edge node with payload 500 bytes.....	26
Figure 15: First edge node pings the VPN IP of the second edge node with payload 500 bytes.....	26
Figure 16: First edge node pings the public IP of the second edge node with payload 1000 bytes.....	27
Figure 17: First edge node pings the VPN IP of the second edge node with payload 1000 bytes.....	27
Figure 18: Python-based http server on second edge node	28
Figure 19: Request to the http server using the public IP	28
Figure 20: Tcpcdump showing the content of the web page in clear text	29
Figure 21: Request to the http server using the VPN IP	29
Figure 22: Tcpcdump showing the content of the web page is encrypted.....	30
Figure 23: An Intel SGX application is divided into an untrusted and trusted part. Privileged system code does not have access to the trusted part of the Intel SGX application at any time	34
Figure 24: Differences between VMs and containers	35
Figure 25: SCONe offers secure containers (with Docker).....	36
Figure 26: An overview of EdgeSec TEE.....	37
Figure 27: An example python command within the image of EdgeSec TEE	38
Figure 28: Commands to install Python libraries (e.g., joblib, numpy, scikit-learn, scipy and threadpoolctl) within EdgeSec TEE.....	38
Figure 29: Successful installation of Python libraries within the docker image of EdgeSec TEE (i.e., joblib, numpy, scikit-learn, scipy and threadpoolctl).....	38
Figure 30: High-level overview of GPURegex in MARVEL.....	44
Figure 31: Construction of the state transition table	45
Figure 32: Architectural comparison of an integrated GPU, packed with the main processor in the same CPU die versus a discrete, dedicated GPU	46
Figure 33: An overview of GPURegex.....	46
Figure 34: An example run of GPURegex inside the container destined for Intel CPUs	47
Figure 35: Overview of AAC.....	48

List of Abbreviations

AAC	Automated Audio Captioning
AS	Authenticator Server
API	Application Programming Interface
CPU	Central Processing Unit
DFA	Deterministic Finite Automaton
DoA	Description of Action
DRAM	Dynamic Random Access Memory
EC	European Commission
EPC	Enclave Page Cache
IDS	Intrusion Detection System
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPsec	Internet Protocol Security
ISA	Instruction Set Architecture
ISP	Internet Service Provider
GPGPU	General Purpose Graphics Processing Unit
GPU	Graphics Processing Unit
GRE	Generic Routing Encapsulation
KPI	Key Performance Indicator
L2TP	Layer 2 Tunnelling Protocol
MEE	Memory Encryption Engine
OS	Operating System
PPTP	Point-to-Point Tunnelling Protocol
PRM	Processor Reserved Memory
RNA	Ribonucleic Acid
RTT	Round-Trip Time
SGX	Software Guard Extensions
SIMD	Single Instruction Multiple Data
SOCKS	Socket Secure
SSL	Secure Sockets Layer
TCB	Trusted Computing Base
TEE	Trusted Execution Environment

TLS	Transport Layer Security
VM	Virtual Machine
VPN	Virtual Private Network
WP	Work Package

DRAFT

Executive Summary

The goal of this deliverable is to present and describe the initial versions of the components that participate in the whole MARVEL platform and enable security and acceleration features in the complete Edge-to-Fog-to-Cloud continuum. This deliverable has been developed within the scope of WP4 “MARVEL E2F2C distributed ubiquitous computing framework”, and more specifically, within the scope of Task 4.3 “Security and acceleration in the complete E2F2C” of the MARVEL project under Grant Agreement No. 957337.

The deliverable demonstrates the initial versions (development until M18) of the security components, namely EdgeSec VPN and EdgeSec TEE, and the initial version of the component offering the acceleration feature, namely GPURegex. EdgeSec VPN secures the communications between the different components of the MARVEL platform within the different layers (i.e., Edge, Fog, Cloud). EdgeSec Trusted Execution Environment (EdgeSec TEE), enables confidential and secure execution of applications that process sensitive data. Acceleration is achieved through GPURegex, which is a component that offers GPU-accelerated stream processing. More specifically, GPURegex accelerates the pattern matching procedure. In this deliverable, we discuss about these tools, their relation to the MARVEL project and how they address the objectives and KPIs defined.

D4.5, which is the following and ultimate version of this deliverable, will present the final versions of the components EdgeSec VPN (i.e., security), EdgeSec TEE (i.e., security) and GPURegex (i.e., acceleration).

1 Introduction

1.1 Purpose and Scope

This deliverable, entitled D4.2 “Security assurance and acceleration in the E2F2C framework – initial version” presents the work performed in the context of Task 4.3 “Security and acceleration in the complete E2F2C”, within the scope of WP4 “MARVEL E2F2C distributed ubiquitous computing framework” and the MARVEL project under Grant Agreement No. 957337. As this is the initial version of the deliverable, we detail the components that have been developed and become available through the MARVEL image registry before M18.

The components that are demonstrated in this deliverable and offer security features to the E2F2C framework are (i) EdgeSec VPN and (ii) EdgeSec TEE. EdgeSec VPN secures the data transfer over the network, while EdgeSec TEE offers confidential computing for python applications that process sensitive user data. In the context of the MARVEL project, EdgeSec VPN will be used to encrypt any data that is transferred between the MARVEL components to meet the requirements of a strict communication security. As EdgeSec TEE will be fully integrated into the MARVEL platform after M18, its complete utilisation will be further discussed in the following version of this deliverable (D4.5).

The component that enables acceleration in the E2F2C framework is GPURegex. More specifically, what GPURegex offers is the performance acceleration of the pattern matching procedure. GPURegex can be applied in numerous cases since pattern matching is the core operation of several and diverse applications, from network packet processing, database search to RNA structure alignments. In the context of the MARVEL project, GPURegex will be probably used for keyword searching against captions exported from audio and video captures.

Thus, in this deliverable, we aim to discuss the details of the development, deployment, integration, and performance results of the three components that bring security assurance (i.e., EdgeSec VPN, EdgeSec TEE) and acceleration (i.e., GPURegex) in the E2F2C framework. During the first months of the project, EdgeSec was referred to as a complete and interdependent component with two separate functionalities (i.e., VPN and TEE). Currently, we believe that it is better to refer to these two distinct functionalities with two separate names, in order to be more coherent; thus, we discuss the two functionalities in two separate sections (Sections 2 and 3).

1.2 Relation to other Work Packages, Deliverables and Activities

From the Task 4.3 description below, we can understand that the security and acceleration features are principally expected in the edge layer, some steps after the data collection from the sensors, to securely and promptly transfer those data to the subsequent layers and participating components. The description of Task 4.3 follows:

*This task will explore remote attestation, a well-known technique, for verifying the state of remote computing devices and for verifying the trustworthiness of the data collected and shared by remote sensors. Moreover, trusted execution environments (such as Intel SGX) which can enable remote attestation and further provide full memory encryption will be explored as well (Sect. 1.4.1.5). These environments contain secure elements that lie in the hardware chip, and support, at least, advanced cryptographic functions and physically protected storage of private and secret keys. This will allow building a **multi-layer architecture** that will provide **security, trust and privacy** in the **edge device** itself. Finally, the utilisation of **GPU accelerated streaming processing** in edge devices will also be explored in this task (Sect. 1.4.1.5). This low-end GPU acceleration in the*

processing of streaming data is able to accelerate light computations as a pre-processing phase, right before offloading tasks to the cloud.

The work conducted in Task 4.3 aims to offer security and acceleration in the complete E2F2C framework, where specifically EdgeSec VPN aims to participate in the MARVEL platform as a more holistic element, securing every step of data transmission with end-to-end network packet encryption.

Furthermore, this deliverable has close relation with Task 3.4, where EdgeSec VPN participates in the distribution of the AI tasks. Also, it can interact with Task 3.3, where GPURegex will accelerate the processing of audio/video captions that will be resulted from the components that participate there. Finally, Task 4.3 and the corresponding deliverables (i.e., D4.2 and D4.5) have also close relation to WP1 by addressing the project objectives and the respecting KPIs.

Figure 1 presents the conceptual architecture of the whole MARVEL platform, where the placement of the components EdgeSec VPN, EdgeSec TEE, and GPURegex is highlighted using black, dashed rectangles. In the following sections, we will refer to this figure to help the reader recognise the relation of the three components with the other MARVEL components.

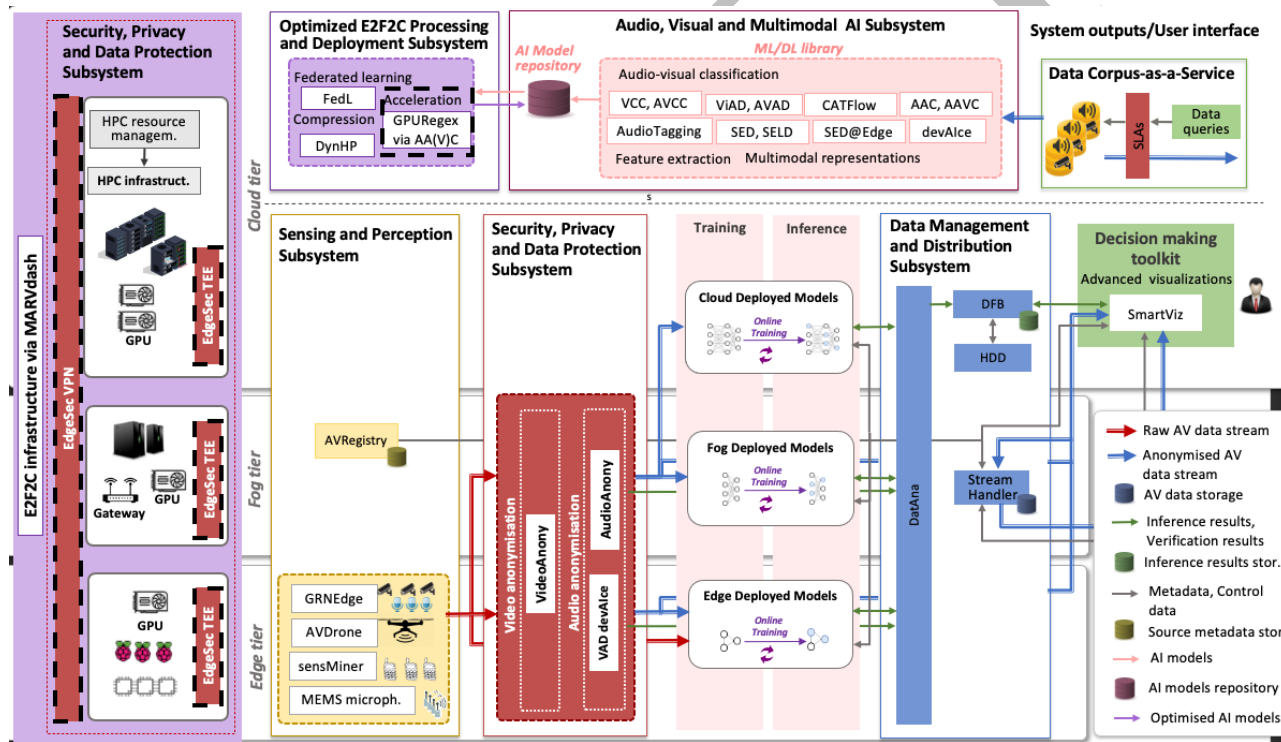


Figure 1: Conceptual architecture of the whole MARVEL platform

1.3 Contribution to WP4 and Project Objectives

This deliverable is the main outcome of Task 4.3, which is entitled “Security and acceleration in the complete E2F2C”. Task 4.3 is part of WP4 “MARVEL E2F2C distributed ubiquitous computing framework”. The description of WP4 is presented in the next paragraph.

WP4 develops the MARVEL E2F2C framework in order to fully harness, in resource-optimised and secure way, the edge (including data capturing), fog and cloud resources to effectively orchestrate and distribute computational and AI-related tasks (Pillar III). A major consideration towards this goal is the ability to perform a significant part of the processing at the Edge or Fog. Therefore, AI-enabled devices that can run (edge-optimised) light-weight DL models play a key role. Moreover, security and acceleration

methods will be enforced at all layers so as to end up with a robust, trustworthy and fast decision support toolkit.

The objectives of WP4 are presented in the DoA as follows:

To address these challenges, WP4 objectives are to: (i) offer GPU stream processing acceleration in edge devices and nodes; (ii) define and deploy a security strategy, including privacy-aware algorithms, at the edge; (iii) describe innovations performed by analogue and digital microphones that are based on MEMS technology; (iv) utilise the openSMILE platform for audio analysis and feature extraction; (v) develop advanced visualisation techniques to support both real-time and long term decision-making.

Thus, this deliverable contributes to WP4 and project objectives by offering (i) GPURegex for processing acceleration and (ii) the combination of EdgeSec VPN and EdgeSec TEE as a security strategy with privacy-aware technologies (i.e., encrypted data transfer across the network and Trusted Execution Environments for confidential execution).

1.4 Structure of the Report

The structure of this deliverable is outlined as follows:

- In the first section, we provide an introduction of this deliverable, highlighting its purpose and scope, its relation to the other work packages, deliverables and activities within the context of the MARVEL project, its contribution to WP4, and the total objectives of the project (i.e., Section 1).
- In the second section, we present the first security feature offered through the component called EdgeSec VPN (i.e., Section 2), which offers encryption of data transmitted over the network traffic.
- Then, in the third section, we present EdgeSec TEE (i.e., Section 3), which is the second component that offers security and privacy in the complete E2F2C2, by enabling confidential processing of code and sensitive data into isolated and encrypted memory regions.
- In the fourth section, we present details about the component, namely GPURegex, that offers processing acceleration of pattern matching applications, using GPUs or other hardware devices that enable SIMD processing (i.e., Section 4).
- In the final section, we conclude this deliverable, by summarising its contents and results (i.e., Section 5).

While EdgeSec aims to offer a unified security assurance in the context of the whole MARVEL platform, the two involved components, namely EdgeSec VPN and EdgeSec TEE are distinct and operate independently. Thus, we present the two components in two distinct sections, so as not to perplex the two technologies.

2 Secure Communication on the Edge (EdgeSec VPN)

In this section, EdgeSec VPN is presented and described. EdgeSec VPN is the first component that brings security and privacy features to the complete E2F2C framework, developed within the MARVEL project. More specifically, EdgeSec VPN is based on the technology of peer-to-peer VPNs.

In the following sections, we provide some background information regarding VPNs, secure peer-to-peer communications, and the related state-of-the-art. We describe the basic development and deployment details. Then, we locate the relevant project and component KPIs. Finally, we discuss about their correlation with EdgeSec VPN and how they can be realised within the context of MARVEL.

2.1 Background

In this section, background information related to EdgeSec VPN is presented. More specifically, we discuss the basics of Virtual Private Networks (VPNs), secure peer-to-peer communications and the state-of-the-art.

2.1.1 Virtual Private Networks

A virtual private network (VPN) is basically a connection over the Internet from a specific device to a target network. The term VPN is associated with encrypted transmitted data. The encryption, although it is not a key characteristic of a VPN connection, it ensures that sensitive data is safely transmitted and prevents unauthorised entities from eavesdropping on the traffic. Corporate environments usually make use of VPNs.

The term VPN is used to describe the communication among a closed user group, using a shared network infrastructure. The available public infrastructure is used for the realisation of the private network operation. Addressing, connectivity, access control, etc. are offered in the same way a conventional private network would offer them. The major advantage of VPN is cost savings since it eliminates the need for long-distance leased lines, operational support, etc. The main disadvantage is that its performance depends on factors that cannot be controlled by the use of a VPN, especially if this VPN uses the internet for the communication of its participants. A number of different VPN protocols have been created, such as PPTP, L2TP, GRE, IPSec, SOCKS. These protocols enable authentication and encryption, two very desirable characteristics of a virtual private network [1].

Another definition sets VPN as an extension of private networks across public networks, with additional authentication and encryption to network traffic. Internet Protocol Security (IPsec) is one of the VPN protocols that is quite popular. It is standardised by the Internet Engineering Task Force (IETF), and it offers protection on the Internet Protocol (IP) layer [2].

2.1.2 Secure Peer-to-Peer Communications

Secure peer-to-peer communication is the goal in many different contexts involving dissimilar technologies.

Edge computing is an emerging paradigm that promises data processing that is done at or near the collectors of data, limiting the possibility for a central cloud computing infrastructure to run beyond its capability. In such a framework, security aspects have been raised [3]. Technologies such as Pseudonymity, Unobservability, Unlinkability, and Anonymity are used for privacy preservation [4]. On the other hand, Confidentiality, Integrity, Availability, Access control, and authentication are the factors for the security evaluation of a system [5]. An ideally reliable system offering edge computing does not disclose a user's identity, behaviour, and location.

These kinds of requirements place barriers to the broader deployment of such systems. However, the goal is to collect and process a large amount of data without revealing a user's private information.

A security architecture that includes an authenticator server (AS) is proposed in [6] as the solution to systems that use cellular networks and multi-access edge computing. Strong authentication, confidentiality, and access control are the security goals of the architecture. AS is considered always secure and is responsible for the provision of required keys for the communication between the sensor nodes and the smartphones (end users). This architecture allows the use of asymmetric or symmetric cryptography in the communications among the different architecture elements.

2.1.3 State-of-the-Art

There are two common VPN solutions: i) IPSec and ii) OpenVPN.

OpenVPN has become the de facto standard in VPNs today. It uses SSL/TLS for key exchange and encryption. OpenVPN is open-source and according to an independent review in 2017 by Cryptography Engineering [7], there were no major vulnerabilities. OpenVPN is fully functional on three major operating systems (Windows, macOS, and Linux). A plethora of ciphers and encryption methods can be used.

IPSec is actually a set of protocols working together [8]. Layer 2 Tunnelling Protocol (L2TP) is used for the tunnelling of the VPN, transferring the messages' payload. Regarding security protocols, IPSec offers encrypting and negotiating keys. In that way, additional security is offered at the IP layers in the form of encryption with additional complexity. Another way of achieving security is certificates and pre-shared private keys. There are studies that recognise complexity as the main drawback of IPSec [9].

A third VPN solution named WireGuard is presented in [10]. This third solution could avoid the complexity of IPSec and perform better than OpenVPN. WireGuard makes use of asymmetric key cryptography and state-of-the-art cryptographic algorithms and protocols such as NOISE, BLAKE2 and Curve25519. It seems that the potentials of this new VPN solution are high. In any case, according to its claims, it is "faster, simpler, and leaner" than the other VPN solutions.

The authors of [11] made a comparison experiment in order to help system administrators to solve the problem of how to choose the best VPN solution, based on their system requirements. The research question was how the performance differs among the state-of-the-art VPN solutions. The results were values in Mbits/sec regarding packets sent from iPerf to the target server. The results showed:

- Any VPN solution is slower than no VPN implementation.
- IPSec shows fast throughput due to the compression of data when this is an option (Linux).
- WireGuard is the best performer in Windows.
- OpenVPN is the slowest in every operating system.

2.2 Early Deployment and Integration

EdgeSec VPN is based on the open-source software n2n¹. The initial version of EdgeSec VPN is uploaded on the MARVEL Docker² image registry and can be downloaded from the MARVEL platform³. In this section, we will describe the current development, deployment, and integration status of the component namely EdgeSec VPN with respect to the whole MARVEL platform.

2.2.1 Development

For the development of EdgeSec VPN, three virtual machines (VM) are used. The first VM has the role of the Super Node whereas the other two VMs have the role of edge nodes. The operating system of the VMs is Ubuntu Linux version 18.04 LTS the size of the RAM is 4GB and the disk is 40GB.

EdgeSec VPN is provided through the n2n software and it is not meant to be self-contained meaning that it is possible to route traffic across n2n and non-n2n networks. The component is containerised with Docker for easier deployment. For EdgeSec VPN, the first layer of Docker image is based on the Ubuntu:18.04 image.

The resulted EdgeSec VPN image can be found on the MARVEL docker image registry and can be downloaded or shared upon request.

2.2.2 Early Deployment

The architecture of the EdgeSec VPN adopts the architecture of n2n, as shown in Figure 2. There are two key components: edge nodes and Super Nodes. The edge nodes are the peers participating in the network. The Super Nodes are used by the edge nodes for discovering other edge nodes. The Super Nodes are also used for routing the traffic when the nodes are behind symmetrical firewalls. The n2n and therefore the EdgeSec VPN, is a peer-to-peer VPN that works on the second layer of the OSI model, allowing the peers to cross NAT and firewalls and be reachable. Edge nodes that participate in the same virtual network form a community. Super Nodes are able to serve more than one community and a single computer can join multiple communities. Within a community, encryption of the packets is feasible with the use of an encryption key. Edge nodes establish direct communication among themselves via UDP however when this is not possible, due to special NAT circumstances, then the Super Node can facilitate the relay of the packets.

EdgeSec VPN is containerised with Docker to simplify the deployment. The relevant Docker file consists of the following commands:

```
1. FROM ubuntu:18.04
2. RUN apt-get update && apt-get install -y build-essential
   net-tools autoconf pkg-config
3. RUN mkdir -p /usr/ipsec
4. WORKDIR /usr/ipsec
5. COPY ./ .
6. RUN ./autogen.sh
7. RUN ./configure
8. RUN make
```

¹ <https://www.ntop.org/products/n2n/>

² <https://www.docker.com>

³ https://marvel-platform.eu/image/edgesec_vpn


```

9.      RUN make install
10.     EXPOSE 4194/udp
11.     CMD ["sh", "init_script.sh"]

```

After the initial build of the image, we need to start a Super Node in a machine that has port 4194 exposed to internet with the following command:

```

1. sudo docker run -it -p 139.91.58.106:4194:4194/udp --name
   n2n_supernode marvel-ipsec supernode -p 4194 -c community.list
   -f

```

Then, we will be able to connect the respecting edge nodes with the following command:

```

1. sudo docker run -it --privileged --net=host --name n2n_edge
   marvel-ipsec edge -d n2n0 -c community -k pass1 -l
   139.91.58.106:4194 -f

```

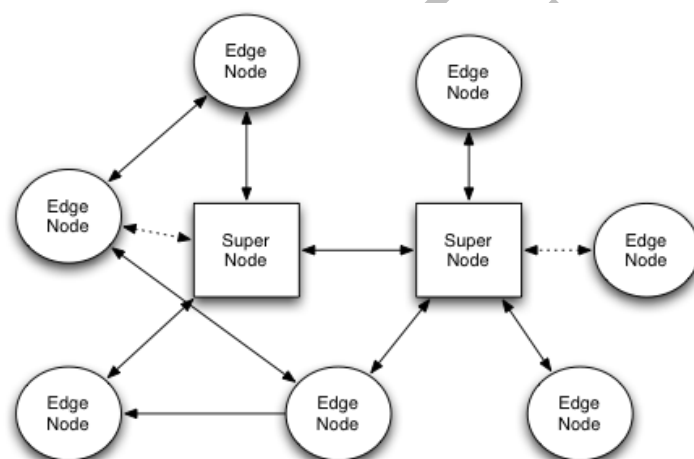


Figure 2: The n2n architecture⁴

2.2.3 Integration with MARVEL

In MARVEL, several components are required to be deployed across three different layers, (i) edge, (ii) fog, and (iii) cloud. In order to manage the deployment of components with Kubernetes⁵, all three layers must be part of the Kubernetes cluster. Kubernetes by design requires that all pods can communicate with other pods on any node without NAT which comes in direct contradiction with the actual setup of having remote nodes. The EdgeSec VPN provides the solution here, due to the fact that it brings together all the participating nodes as if they were under the same local network making any NAT or firewall transparent to the communication between them.

Essentially all participating computing devices form a full mesh network where every device has a direct connection with every other device as depicted in Figure 3. For example, a drone that is located in the edge layer can ping within one hop a server located at the fog as well as a

⁴ <https://www.ntop.org/products/n2n/>

⁵ <https://kubernetes.io>

server located at the cloud. The EdgeSec VPN is instantiated as a Docker container within the computing device leveraging the microservices approach.

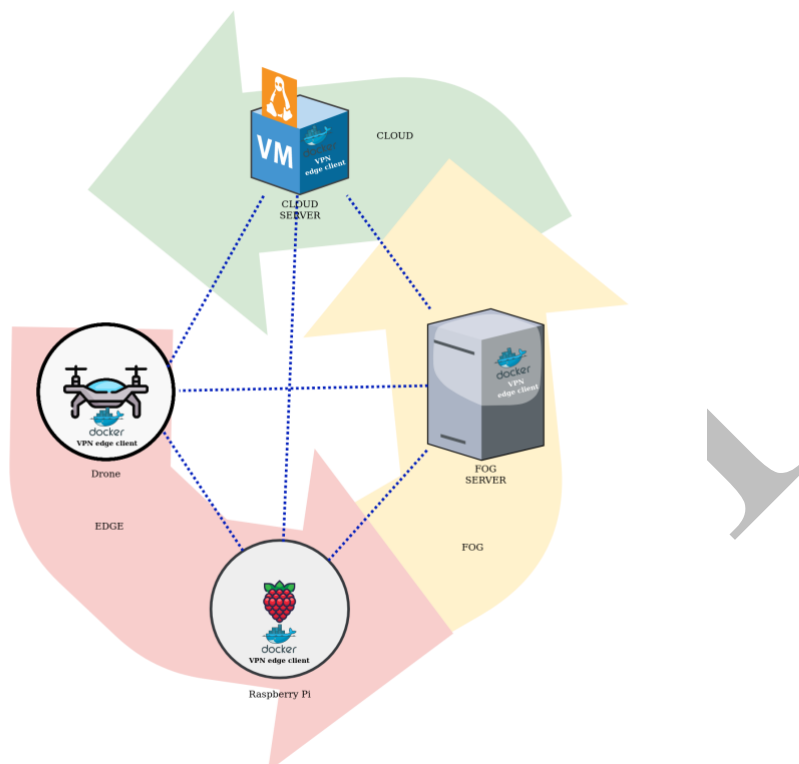


Figure 3: EdgeSec VPN forms a full mesh topology

2.3 Use Cases and Related Components

This section describes the MARVEL use cases that EdgeSec VPN participates in. In addition, an overview of the related components is presented. EdgeSec VPN is also presented in Figure 1 as part of the subsystem named “Security, Privacy, and Data Protection”. As shown in the figure, EdgeSec VPN secures the communications between every component that either participates in the edge, fog or cloud layer of MARVEL.

2.3.1 Related Components

The majority of MARVEL components will be deployed within Kubernetes. However not all components that are going to be deployed within Kubernetes are also meant to run at the cloud layer where Kubernetes is. This raised the need for having nodes that exist at the fog and at the edge layer to join the Kubernetes cluster at the cloud. Kubernetes by design requires that all pods can communicate with other pods on any node without NAT which comes in direct contradiction with the actual setup of having remote nodes. The EdgeSec VPN provides the solution here, due to the fact that it brings together all the participating nodes as if they were under the same local network making any NAT or firewall transparent to the communication between them.

EdgeSec VPN becomes the underlying network that allows each remote node to join the Kubernetes cluster at the cloud. This implies that all components that are deployed in

Kubernetes essentially utilise the EdgeSec VPN as they traverse the tunnel created by the EdgeSec VPN.

2.3.2 EdgeSec VPN in UNS Use Case

For the UNS1 – Drone Experiment use case, we had to deal with the fact that all the involved computing devices are accessing the internet via proxy. In order to build the docker image on the UNS premises, we had to include the appropriate environment variables in the Dockerfile. The modified Dockerfile with the required proxy environment variables follows:

Table 1: Modified Dockerfile with proxy environment variables

```
1. FROM ubuntu:18.04
2. ENV http_proxy 'http://proxy.uns.ac.rs:8080'
3. ENV https_proxy 'http://proxy.uns.ac.rs:8080'
4. RUN apt-get update && apt-get install -y build-essential
   net-tools autoconf pkg-config
5. RUN mkdir -p /usr/ipsec
6. WORKDIR /usr/ipsec
7. COPY ./ .
8. RUN ./autogen.sh
9. RUN ./configure
10. RUN make
11. RUN make install
12. EXPOSE 4194/udp
13. CMD ["sh", "init_script.sh"]
```

The infrastructure of UNS consists of a server that is located at the fog layer, a Raspberry Pi that is located at the edge layer and an Intel NUC that is mounted on a drone also located at the edge. The proxy that is used in the UNS infrastructure created unexpected communication issues between the edge nodes and the Super Node that is located at the cloud in PSNC's infrastructure. This communication is necessary for nodes to announce themselves and discover other nodes.

The adopted solution was to instantiate a secondary Super Node at the fog layer and configure the edge nodes to connect to this secondary Super Node. Having a secondary Super Node on the premises of UNS allows us to bypass the proxy issue. The secondary Super Node is able to communicate directly with the main Super Node at the Cloud after configuring the necessary port forwarding at the fog layer required by a Super Node. The two Super Nodes form a special community, called federation. When a Super Node is part of a federation, it propagates its knowledge about all the edges, to the other Super Nodes in the federation (Figure 4).

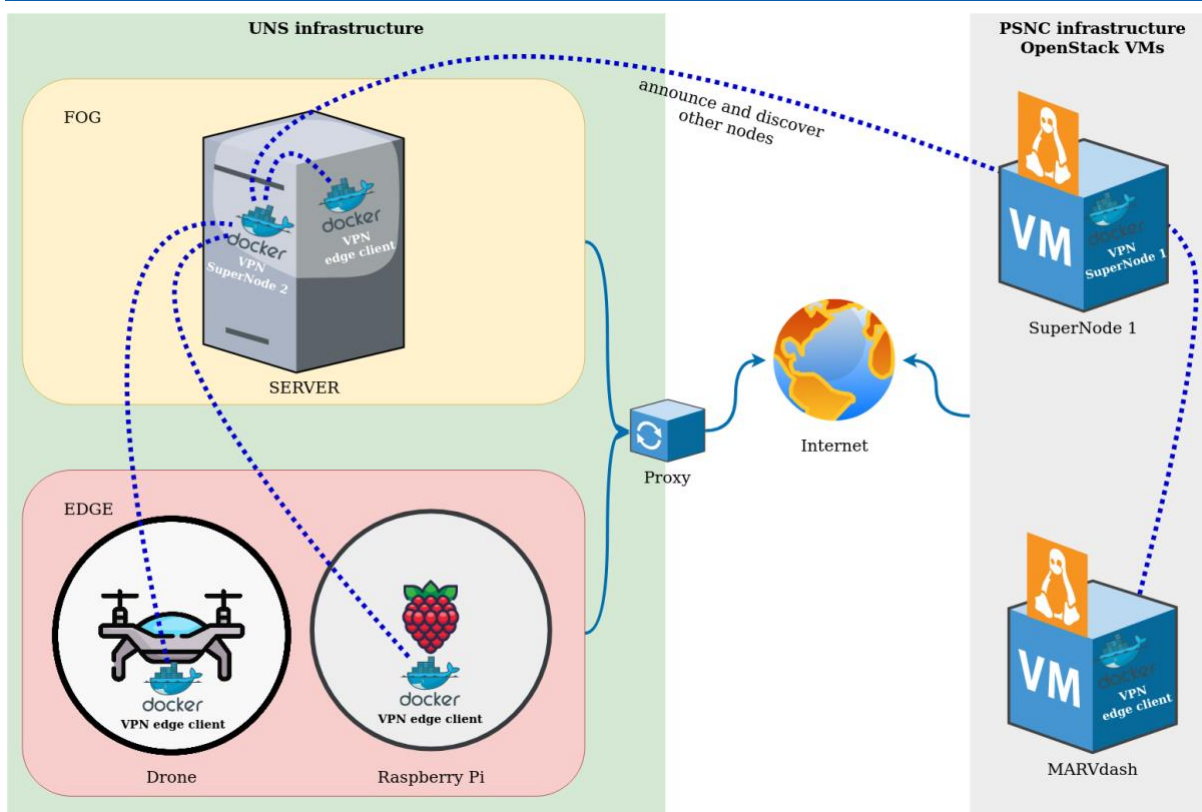


Figure 4: EdgeSec VPN in UNS Use Case

2.3.3 EdgeSec VPN in GRN Use Cases

For the GRN3 – “Traffic Conditions and Anomalous Events” and GRN4 – “Junction Traffic Trajectory Collection” use cases, no proxy was present in the infrastructure and therefore no extra modification was required to the initial Dockerfile. The infrastructure of GRN consists of a server that is located at the fog layer and a workstation located at the edge layer. The communication between those nodes at the GRN and the Super Node at the cloud in PSNC’s infrastructure is unhindered and therefore no additional Super Node was required. Nodes are able to directly announce themselves and discover other nodes via the Super Node.

The communication between the participating nodes is limited to the traffic that matches the network subnet defined by the EdgeSec VPN. This means that all unrelated traffic such as browsing the internet or downloading updates, etc. is not routing through the VPN, thus limiting the overhead of the VPN channel (Figure 5).

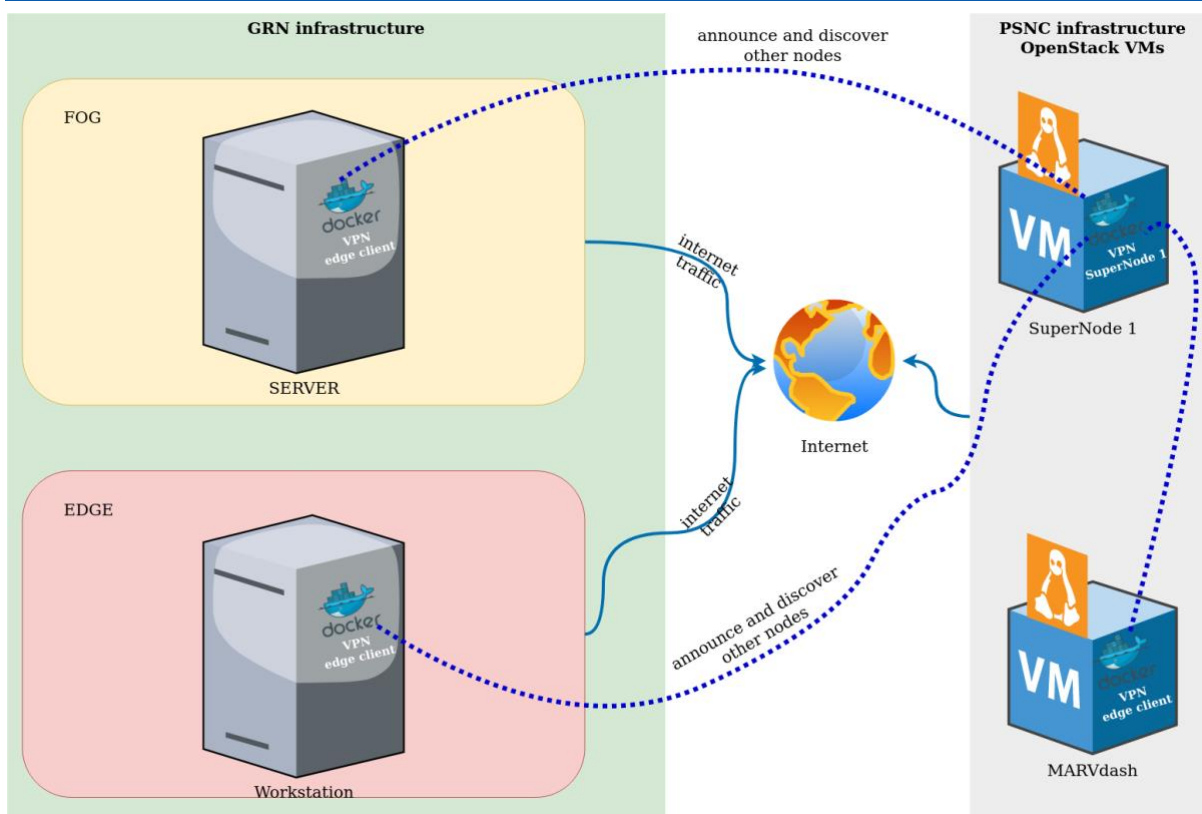


Figure 5: EdgeSec VPN in GRN Use Cases

2.3.4 EdgeSec VPN in MT Use Cases

For the MT use cases, i.e., MT1 – “Monitoring of Crowded Areas” and MT3 – “Monitoring of Parking Places”, no proxy was present in the infrastructure and therefore no extra modification was required to the initial Dockerfile. The fog layer of the infrastructure for the MT use cases is actually hosted in FBK’s infrastructure. FBK is hosting two workstations that are located at the fog layer. Only one of them is meant to be part of the EdgeSec VPN. The communication between the workstation at the FBK and the Super Node at the cloud in PSNC’s infrastructure is unhindered and therefore no additional Super Node was required. The workstation is able to directly announce itself and discover other nodes via the Super Node.

The aforementioned setup introduced new requirements in order to comply with security policies present at the FBK network. The first requirement is to route all the traffic via the EdgeSec VPN and not only the traffic that matches the network subnet defined by the EdgeSec VPN. The second requirement is that EdgeSec VPN should not interfere with the communication between the two workstations internally.

In order to address the first requirement, a new VM has been instantiated in the cloud. The role of this newly created VM is to act as a gateway to the internet for the workstation at the FBK. This gateway will route all the traffic originating from the workstation through PSNC’s infrastructure. Additionally, custom static routes are added to the routing table of the workstation, forcing the traffic to be routed through the VPN gateway. The second requirement is addressed by carefully modifying the routing table of the workstation without interfering with the internal communication of the two workstations.

The final outcome of the aforementioned setup is depicted in Figure 6. Workstation 1 and Workstation 2 are connected internally via a switch. Workstation 1, accesses internet via the

main router of the FBK's network whereas Workstation 2 accesses internet via the VM hosting the VPN gateway at the cloud in PSNC's infrastructure.

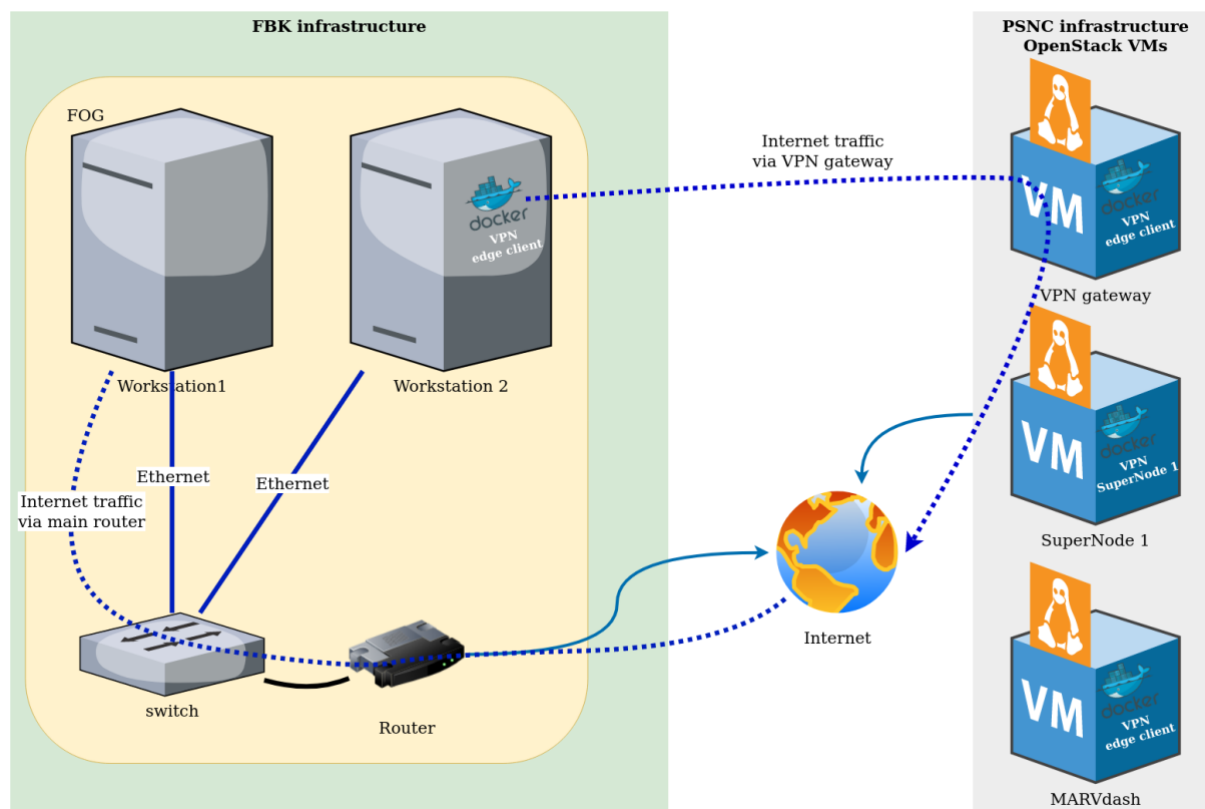


Figure 6: EdgeSec VPN in MT Use Case

2.4 Early Experimental Results

This section demonstrates the experiments that were performed to explore the capabilities of EdgeSec VPN.

2.4.1 Testbed Setup

The testbed that was used is the same as the one used for the development of EdgeSec VPN. Three VMs were created using Proxmox as hypervisor (Figure 7). The first VM has the role of the Super Node whereas the other two VMs have the role of edge nodes. The operating system of the VMs is Ubuntu Linux version 20.04 LTS, the size of the RAM is 4GB, and the disk is 50GB. All the machines have direct access to internet without the presence of NAT.

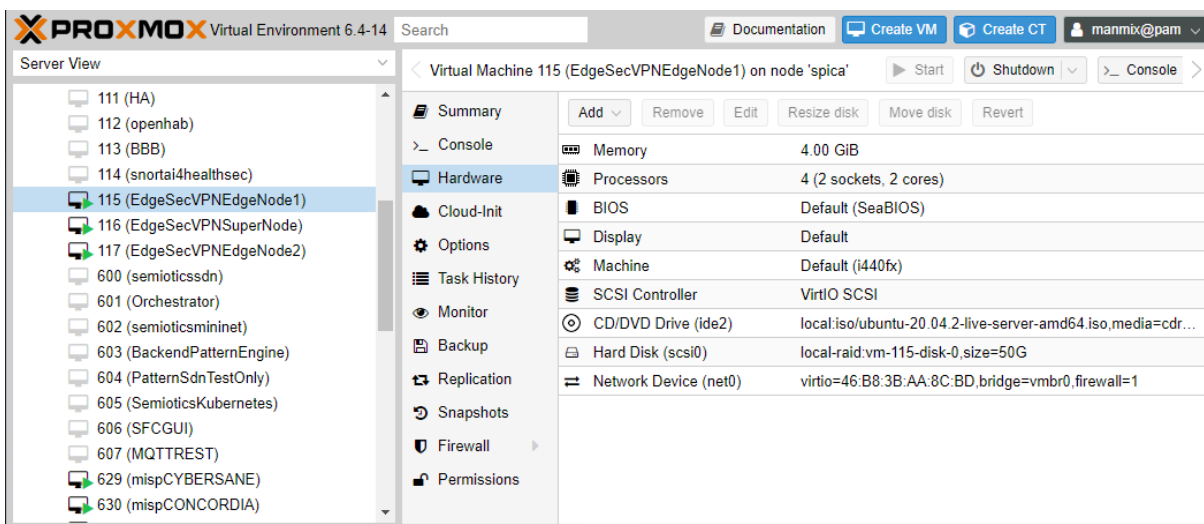


Figure 7: Virtual Machines in Proxmox

2.4.2 Experiments

At the first VM, we execute the Super Node script. The appropriate command follows:

```
1. sudo docker run -it -p 139.91.58.106:4194:4194/udp --name
   n2n_supernode marvel-ipsec supernode -p 4194 -c community.list
   -f -v
```

The IP address is 139.91.58.106, the port for the incoming connections is 4194, the name for the container identification is “n2n_supernode” and the name of the docker image is “marvel-ipsec”. Finally, the “community.list” corresponds to a file with the names of the communities that the Super Node will serve. In this case, the file contains the community “community”. The result of the command is presented in Figure 8.

```

vagrant@supernode:~/ipsec$ ./start_supernode.sh
05/Jun/2022 17:16:23 [sn_utils.c:400] added allowed community 'community' [total: 1]
05/Jun/2022 17:16:23 [sn_utils.c:427] assigned sub-network 192.168.50.0/24 to community 'marvelnet'
05/Jun/2022 17:16:23 [sn_utils.c:445] loaded 1 fixed-name communities from community.list
05/Jun/2022 17:16:23 [sn_utils.c:448] loaded 0 regular expressions for community name matching from community.list
05/Jun/2022 17:16:23 [sn_utils.c:120] started shared secrets calculation for edge authentication
05/Jun/2022 17:16:23 [sn_utils.c:136] calculated shared secrets for edge authentication
05/Jun/2022 17:16:23 [sn_utils.c:145] calculating dynamic keys
05/Jun/2022 17:16:23 [supernode.c:588] WARNING: using default federation name; FOR TESTING ONLY, usage of a custom federation name (-F) is highly recommended!
05/Jun/2022 17:16:23 [sn_utils.c:120] started shared secrets calculation for edge authentication
05/Jun/2022 17:16:23 [sn_utils.c:136] calculated shared secrets for edge authentication
05/Jun/2022 17:16:23 [supernode.c:604] supernode is listening on UDP 4194 (main)
05/Jun/2022 17:16:23 [supernode.c:613] supernode opened TCP 4194 (aux)
05/Jun/2022 17:16:23 [supernode.c:620] supernode is listening on TCP 4194 (aux)
05/Jun/2022 17:16:23 [supernode.c:629] supernode is listening on UDP 5645 (management)
05/Jun/2022 17:16:23 [supernode.c:641] dropping privileges to uid=65534, gid=65534
05/Jun/2022 17:16:23 [sn_utils.c:807] successfully created resolver thread
05/Jun/2022 17:16:23 [supernode.c:659] supernode started
05/Jun/2022 17:16:28 [sn_utils.c:1725] locked community 'community' to unencrypted headers
05/Jun/2022 17:16:28 [sn_utils.c:1218] assign IP 192.168.50.133/24 to tap adapter of edge
05/Jun/2022 17:16:28 [sn_utils.c:1097] created edge FA:03:ED:A2:24:67 ==> 139.91.58.111:41408
05/Jun/2022 17:16:31 [sn_utils.c:1097] created edge 6E:D4:B2:CA:C3:CE ==> 139.91.58.105:44983

```

Figure 8: The output of the Super Node execution script

At the next virtual machine, we execute the script for the edge node. The command and its description follow:

```
1. sudo docker run -it --privileged --net=host --name n2n_edge
marvel-ipsec edge -d n2n0 -c community -k pass1 -l
139.91.58.106:4194 -f
```

More specifically, “*n2n_edge*” is the name for the container identification, “*marvel-ipsec*” is the docker image, “*community*” is the community that the node will join, “*pass1*” is the password, “*139.91.58.106*” is the IP address and “*4194*” is the port number of the Super Node.

The result of the first edge node command is presented in Figure 10.

```
11. edgenode1
vagrant@edgenode1:~/ipsec$ ./start_edge.sh
05/Jun/2022 17:16:28 [edge_utils.c:3774] adding supernode = 139.91.58.106:4194
05/Jun/2022 17:16:28 [edge.c:1045] WARNING: switching to AES as key was
05/Jun/2022 17:16:28 [edge.c:1085] starting n2n edge 3.0.0 Mar 10 20
05/Jun/2022 17:16:28 [edge.c:1091] using compression: none.
05/Jun/2022 17:16:28 [edge.c:1092] using AES cipher.
05/Jun/2022 17:16:28 [edge_utils.c:392] number of supernodes in the list
05/Jun/2022 17:16:28 [edge_utils.c:394] supernode 0 => 139.91.58.106:4194
05/Jun/2022 17:16:28 [edge_utils.c:483] successfully created resolver thread
05/Jun/2022 17:16:28 [edge.c:1122] automatically assign IP address by supernode
05/Jun/2022 17:16:28 [edge.c:1194] send REGISTER_SUPER to supernode [139.91.58.106:4194] asking for
IP address
05/Jun/2022 17:16:28 [edge.c:1194] received REGISTER_SUPER_ACK from supernode for IP address assignme
nt
05/Jun/2022 17:16:28 [edge.c:1194] create local tap device IP: 192.168.50.133, Mask: 255.255.0.0,
WARNING: n2n has not been compiled with libcap-dev; some commands
05/Jun/2022 17:16:28 [edge.c:1194] dropping privileges to uid=65534, gid=65534
05/Jun/2022 17:16:28 [edge.c:1329] edge started
05/Jun/2022 17:16:28 [edge_utils.c:1132] successfully joined multicast group 224.0.0.68:1968
05/Jun/2022 17:16:28 [edge_utils.c:2730] [OK] edge <<< =====>>> supernode
```

Figure 9: The output of the first Edge Node

At the last VM, we start the script for the edge node, one more time. The IP address of the edge node was automatically assigned by the Super Node. The command for the second Edge Node and its description follows:

```
1. sudo docker run -it --privileged --net=host --name
n2n_edge marvel-ipsec edge -d n2n0 -a 192.168.50.200 -c
community -k pass1 -l 139.91.58.106:4194 -f
```

Again, “*n2n_edge*” is the name for the container identification, “*marvel-ipsec*” is the docker image, “*community*” is the community that the node will join, “*pass1*” is the password, “*139.91.58.106*” is the IP address and “*4194*” is the port number of the Super Node.

Finally, the result of the command is presented in Figure 10.


```

7. edgenode2
vagrant@edgenode2:~/ipsec$ ./start_edge.sh
05/Jun/2022 17:16:31 [edge_utils.c:3774] adding supernode = 139.91.58.106:4194
05/Jun/2022 17:16:31 [edge.c:1045] WARNING: switching to AES as key was
05/Jun/2022 17:16:31 [edge.c:1085] starting n2n edge 3.0.0 Jun  5 2022
05/Jun/2022 17:16:31 [edge.c:1091] using compression: none.
05/Jun/2022 17:16:31 [edge.c:1092] using AES cipher.
05/Jun/2022 17:16:31 [edge_utils.c:392] number of supernodes in the list
05/Jun/2022 17:16:31 [edge_utils.c:394] supernode 0 => 139.91.58.106:4194
05/Jun/2022 17:16:31 [edge_utils.c:483] successfully created resolver thread
05/Jun/2022 17:16:31 [edge.c:1116] use manually set IP address
05/Jun/2022 17:16:31 [edge.c:1211] created local tap device IP: 192.168.50.200. mask: 255.255.255.0,
MAC: 6E:D4:B2:CA:C3:CE
05/Jun/2022 17:16:31 [edge.c:1211] WARNING: n2n has not been compiled with libcap-dev; some commands
may fail
05/Jun/2022 17:16:31 [edge.c:1211] dropping privileges to uid=65534, gid=65534
05/Jun/2022 17:16:31 [edge.c:1211] edge started
05/Jun/2022 17:16:31 [edge.c:1132] successfully joined multicast group 224.0.0.68:1968
05/Jun/2022 17:16:31 [edge_utils.c:2730] [OK] edge <<< =====>>> supernode

```

Figure 10: The output of the second Edge Node

In order to test the connectivity with and without the deployment of EdgeSec VPN, we use the ping command. First, we send 100 ping packets from the first edge node to the second edge node using the public IPs and then we do the same using the VPN IPs (Figure 11). In both cases, we have 0% packet loss.

```

18. edgenode1
vagrant@edgenode1:~$ ping 139.91.58.105 -c 100 -q
PING 139.91.58.105 (139.91.58.105) 56(84) bytes of data.

--- 139.91.58.105 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 101335ms
rtt min/avg/max/mdev = 0.428/0.552/2.396/0.196 ms
vagrant@edgenode1:~$ ping 192.168.50.200 -c 100 -q
PING 192.168.50.200 (192.168.50.200) 56(84) bytes of data.

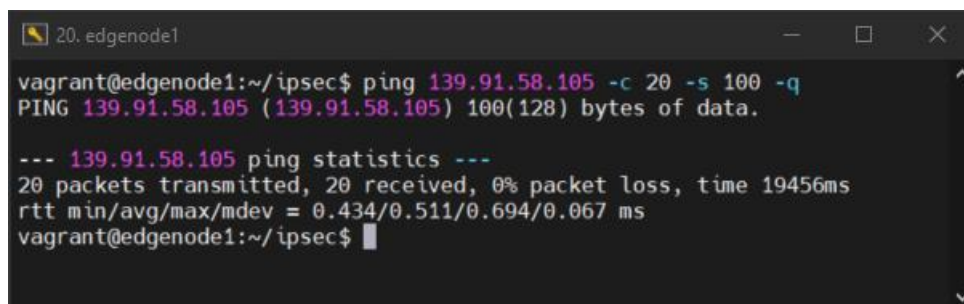
--- 192.168.50.200 ping statistics ---
100 packets transmitted, 100 received, 0% packet loss, time 99320ms
rtt min/avg/max/mdev = 0.857/1.054/1.934/0.152 ms
vagrant@edgenode1:~$

```

Figure 11: Ping connectivity experiment

2.4.3 Results

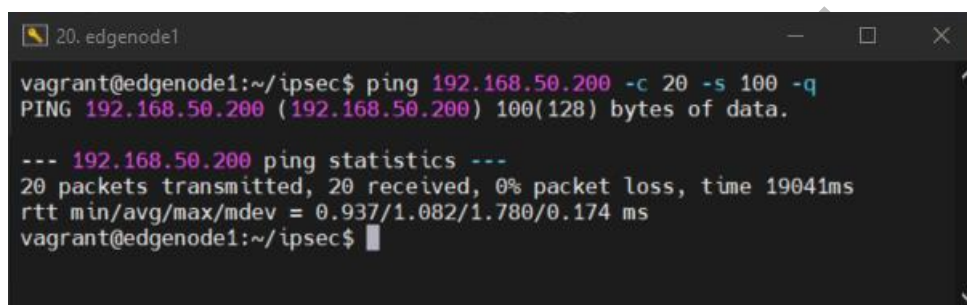
In order to test the overhead that is introduced by the deployment of EdgeSec VPN (in terms of network traffic latency), we use the ping command again. First, we send 20 ping packets from the first edge node to the second edge node using the public IPs and then we do the same using the VPN IPs. We repeat the same test with payload 100 bytes, 500 bytes, and 1000 bytes (Figure 12, Figure 13, Figure 14, Figure 15, Figure 16, Figure 17). The ICMP header and IP version 4 header add extra 28 bytes to the packet.



```
20. edgenode1
vagrant@edgenode1:~/ipsec$ ping 139.91.58.105 -c 20 -s 100 -q
PING 139.91.58.105 (139.91.58.105) 100(128) bytes of data.

--- 139.91.58.105 ping statistics ---
20 packets transmitted, 20 received, 0% packet loss, time 19456ms
rtt min/avg/max/mdev = 0.434/0.511/0.694/0.067 ms
vagrant@edgenode1:~/ipsec$
```

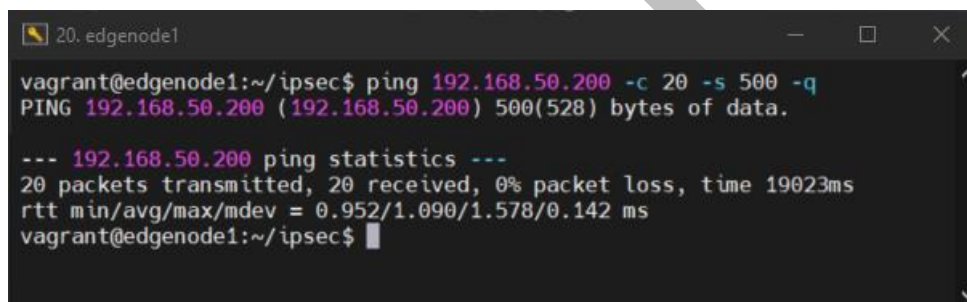
Figure 12: First edge node pings the public IP of the second edge node with payload 100 bytes



```
20. edgenode1
vagrant@edgenode1:~/ipsec$ ping 192.168.50.200 -c 20 -s 100 -q
PING 192.168.50.200 (192.168.50.200) 100(128) bytes of data.

--- 192.168.50.200 ping statistics ---
20 packets transmitted, 20 received, 0% packet loss, time 19041ms
rtt min/avg/max/mdev = 0.937/1.082/1.780/0.174 ms
vagrant@edgenode1:~/ipsec$
```

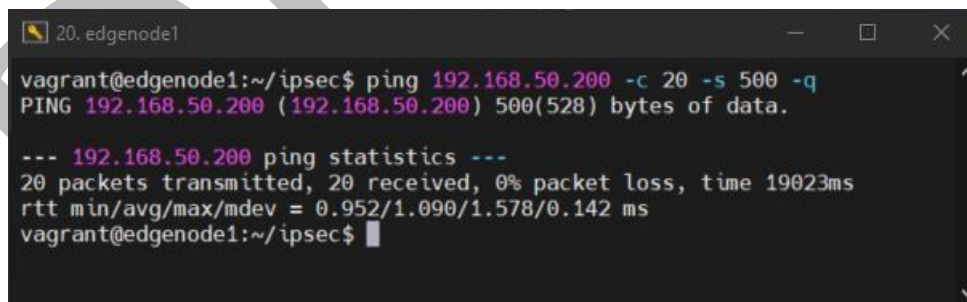
Figure 13: First edge node pings the VPN IP of the second edge node with payload 100 bytes



```
20. edgenode1
vagrant@edgenode1:~/ipsec$ ping 192.168.50.200 -c 20 -s 500 -q
PING 192.168.50.200 (192.168.50.200) 500(528) bytes of data.

--- 192.168.50.200 ping statistics ---
20 packets transmitted, 20 received, 0% packet loss, time 19023ms
rtt min/avg/max/mdev = 0.952/1.090/1.578/0.142 ms
vagrant@edgenode1:~/ipsec$
```

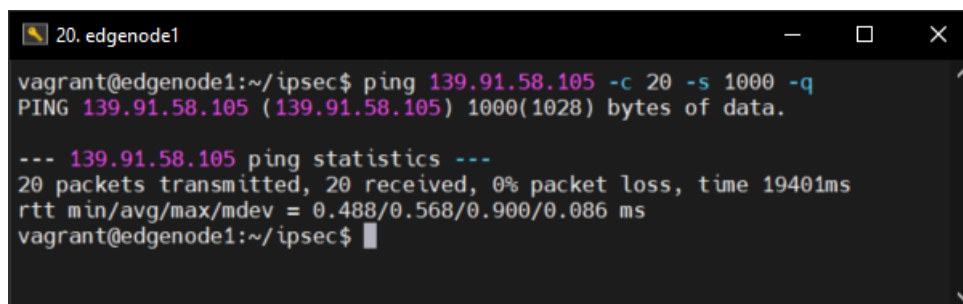
Figure 14: First edge node pings the public IP of the second edge node with payload 500 bytes



```
20. edgenode1
vagrant@edgenode1:~/ipsec$ ping 192.168.50.200 -c 20 -s 500 -q
PING 192.168.50.200 (192.168.50.200) 500(528) bytes of data.

--- 192.168.50.200 ping statistics ---
20 packets transmitted, 20 received, 0% packet loss, time 19023ms
rtt min/avg/max/mdev = 0.952/1.090/1.578/0.142 ms
vagrant@edgenode1:~/ipsec$
```

Figure 15: First edge node pings the VPN IP of the second edge node with payload 500 bytes



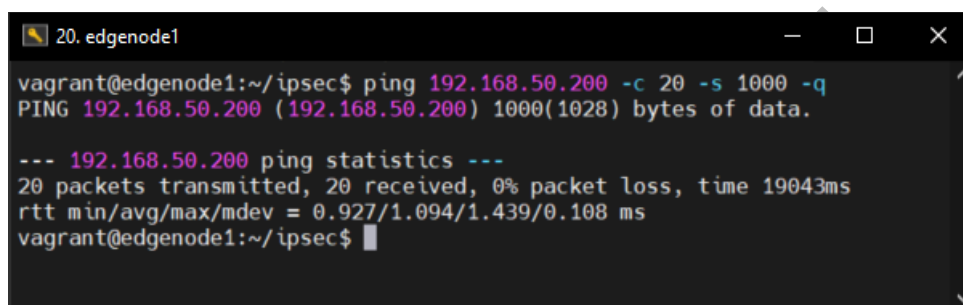
```

20. edgenode1
vagrant@edgenode1:~/ipsec$ ping 139.91.58.105 -c 20 -s 1000 -q
PING 139.91.58.105 (139.91.58.105) 1000(1028) bytes of data.

--- 139.91.58.105 ping statistics ---
20 packets transmitted, 20 received, 0% packet loss, time 19401ms
rtt min/avg/max/mdev = 0.488/0.568/0.900/0.086 ms
vagrant@edgenode1:~/ipsec$

```

Figure 16: First edge node pings the public IP of the second edge node with payload 1000 bytes



```

20. edgenode1
vagrant@edgenode1:~/ipsec$ ping 192.168.50.200 -c 20 -s 1000 -q
PING 192.168.50.200 (192.168.50.200) 1000(1028) bytes of data.

--- 192.168.50.200 ping statistics ---
20 packets transmitted, 20 received, 0% packet loss, time 19043ms
rtt min/avg/max/mdev = 0.927/1.094/1.439/0.108 ms
vagrant@edgenode1:~/ipsec$

```

Figure 17: First edge node pings the VPN IP of the second edge node with payload 1000 bytes

Based on the ping measurements, we created the corresponding tables to compare the network traffic before and after the EdgeSec VPN is deployed (Table 2, Table 3, Table 4). We measure the Round-trip time (RTT) which is the duration in milliseconds (ms) it takes for a network request to be transmitted from the first edge node to the second edge node plus the duration of the return.

Table 2: Packet size 100 bytes

RTT	Ping the public IP	Ping the VPN IP	Overhead
Minimum	0.434ms	0.937ms	0.503ms
Average	0.511ms	1.082ms	0.571ms
Maximum	0.694ms	1.780ms	1.086ms
Standard deviation	0.067ms	0.174ms	0.107ms

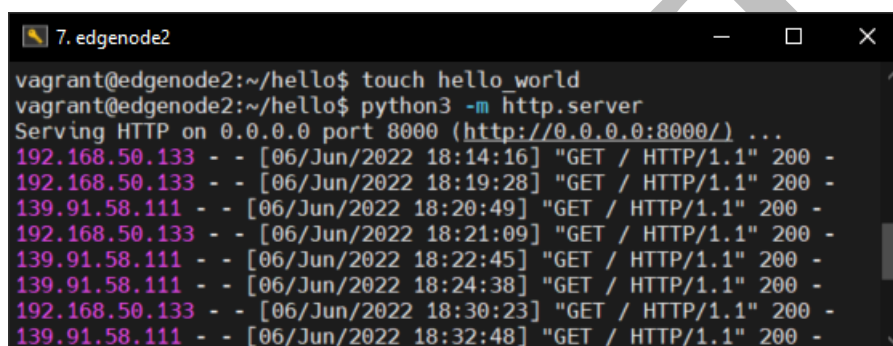
Table 3: Packet size 500 bytes

RTT	Ping the public IP	Ping the VPN IP	Overhead
Minimum	0.455ms	0.952ms	0.497ms
Average	0.539ms	1.090ms	0.551ms
Maximum	0.870ms	1.578ms	0.708ms
Standard deviation	0.092ms	0.142ms	0.05ms

Table 4: Packet size 1000 bytes

RTT	Ping the public IP	Ping the VPN IP	Overhead
Minimum	0.488ms	0.927ms	0.439ms
Average	0.568ms	1.094ms	0.526ms
Maximum	0.900ms	1.439ms	0.539ms
Standard deviation	0.086ms	0.108ms	0.022ms

Additionally, we tried to verify that data in transit is encrypted. To that end, we run a simple python-based http server on the second edge node (Figure 18) and we attempt to request the contents of the webpage from the first edge node using both the public as well as the VPN IP.



```

7. edgenode2
vagrant@edgenode2:~/hello$ touch hello_world
vagrant@edgenode2:~/hello$ python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
192.168.50.133 - - [06/Jun/2022 18:14:16] "GET / HTTP/1.1" 200 -
192.168.50.133 - - [06/Jun/2022 18:19:28] "GET / HTTP/1.1" 200 -
139.91.58.111 - - [06/Jun/2022 18:20:49] "GET / HTTP/1.1" 200 -
192.168.50.133 - - [06/Jun/2022 18:21:09] "GET / HTTP/1.1" 200 -
139.91.58.111 - - [06/Jun/2022 18:22:45] "GET / HTTP/1.1" 200 -
139.91.58.111 - - [06/Jun/2022 18:24:38] "GET / HTTP/1.1" 200 -
192.168.50.133 - - [06/Jun/2022 18:30:23] "GET / HTTP/1.1" 200 -
139.91.58.111 - - [06/Jun/2022 18:32:48] "GET / HTTP/1.1" 200 -

```

Figure 18: Python-based http server on second edge node

Initially, we make the request using the public IP (Figure 19) and we are able to receive the response.



```

32. edgenode1
vagrant@edgenode1:~$ curl http://139.91.58.105:8000
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Directory listing for /</title>
</head>
<body>
<h1>Directory listing for /</h1>
<hr>
<ul>
<li><a href="hello_world">hello_world</a></li>
</ul>
<hr>
</body>
</html>
vagrant@edgenode1:~$

```

Figure 19: Request to the http server using the public IP

With the help of tcpdump command, we are able to verify that data in transit is not encrypted when using the public IP (Figure 20).

```

vagrant@edgenode1:~$ sudo tcpdump -A -i ens18 host 139.91.58.105
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on ens18, link-type EN10MB (Ethernet), capture size 262144 bytes
18:24:38.015533 IP ubuntu.ics.forth.gr.8000 > edgenode1.ics.forth.gr.40916: Flags [ACK], seq 2875250044, length 155
E.....@.0....[:i[:i...@]. .....P.....
..#...|HTTP/1.0 200 OK
Server: SimpleHTTP/0.6 Python/3.8.10
Date: Mon, 06 Jun 2022 18:24:38 GMT
Content-type: text/html; charset=utf-8
Content-Length: 344

18:24:38.015625 IP edgenode1.ics.forth.gr.40916 > ubuntu.ics.forth.gr.8000: Flags [ACK], seq 2875250051, win 501, options [nop,nop,TS val 2345739228, length 0]
E..4..@.0....[:o[:i...@]. .....
..#..
18:24:38.015671 IP ubuntu.ics.forth.gr.8000 > edgenode1.ics.forth.gr.40916: Flags [FP.], seq 156:500, ack 83, win 509, options [nop,nop,TS val 2345739229, ecr 2875250044], length 344
E.....@.0....[:i[:o.@.....
..#...|!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Directory listing for /</title>
</head>
<body>
<h1>Directory listing for /</h1>
<hr>
<ul>
<li><a href="hello_world">hello_world</a></li>
</ul>
<hr>
</body>
</html>

```

Figure 20: Tcpcdump showing the content of the web page in clear text

We then make the request using the VPN IP (Figure 21) and we are again able to receive the response.

```

vagrant@edgenode1:~$ curl http://192.168.50.200:8000
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Directory listing for /</title>
</head>
<body>
<h1>Directory listing for /</h1>
<hr>
<ul>
<li><a href="hello_world">hello_world</a></li>
</ul>
<hr>
</body>
</html>
vagrant@edgenode1:~$

```

Figure 21: Request to the http server using the VPN IP

Again, with the help of tcpdump command we are able to verify that data in transit is now encrypted when using the VPN IP (Figure 22).

transferred encrypted) and EdgeSec TEE that offers trusted and protected execution in environments that cannot be trusted, when sensitive data needs to be processed.

Table 5: Project-related KPIs that concern EdgeSec VPN

KPI ID	KPI Description	Strategy	Related Task	Related Component
KPI-O1-E3-2	The end-to-end data flow from the edge to the cloud, will be 100% encrypted.	No unencrypted data will ever be transmitted by/to any processing layer (E2F2C).	T4.3	EdgeSec VPN
KPI-O3-E3-1	Realise a secure computing framework at all the processing layers.	Ensure that there are no weak links in the E2F2C chain, every layer and communication channel between them shall be secure.	T4.3	EdgeSec VPN, EdgeSec TEE
iKPI-2.2	At least three (3) different cyber threats avoided due to E2F2C.	Cyber threats that could otherwise potentially be exploited to harm the system, but their impact is minimised due to the security features offered by E2F2C.	T4.3	EdgeSec VPN, EdgeSec TEE

2.5.2 Component-related KPIs

The component-related KPIs that concern EdgeSec VPN are presented in Table 6. More specifically, with EdgeSec VPN, MARVEL aims to minimise the effort for the end user to configure the component. In addition, another goal is to leave a minimum imprint in the network traffic after deploying EdgeSec VPN. Finally, MARVEL aims to effectively avoid at least three distinct cyber threats while providing encryption for data in transit.

Considering the fact that EdgeSec VPN is offered through a Docker container, the time that is required by the end user in order to setup the underlying environment and configure EdgeSec VPN is significantly reduced. In addition, setting up EdgeSec VPN does not require advanced knowledge of networking systems, since everything is offered within the container image. The only manual configuration that is required by the end user is to provide an IP address and a port number both corresponding to the Super Node. Thus, the end user is only required to run the EdgeSec VPN container.

Regarding the scalability KPI, as already shown by the ping measurements in Table 2, Table 3, and Table 4 the deployment of the EdgeSec VPN has a minimum increase in network traffic latency.

EdgeSec VPN by design removes ISP and any other middle-man from the list of components that need to be trusted. Threats such as ISP snooping, attacks over insecure wireless networks as well as compromised networking equipment are avoided.

Finally, we have demonstrated in Figure 22, that data in transit is encrypted, thus addressing the last KPI that refers to the Communication Security.

Table 6: Component-related KPIs that concern EdgeSec VPN

KPI	Metric	Expected Result	Relevant Project KPI
Usability	Effort needed by end user	Least possible manual tuning for component users	<i>KPI-O1-E3-2,</i>

Scalability	Network performance metrics (latency)	Zero or barely noticeable increase in network traffic latency due to the deployment of EdgeSec	<i>KPI-O3-E3-1,</i> <i>iKPI-2.2</i>
Effectiveness for Avoiding Cyber Threats	Number of threats avoided	3 distinct cyber threats avoided	
Communication Security	Amount of encrypted data in transit	100% of data will be end-to-end encrypted	

DRAFT

3 Trusted Execution on the Edge (EdgeSec TEE)

In this section, EdgeSec TEE is presented and described. EdgeSec TEE is the second component that brings security and privacy features to the complete E2F2C framework, developed within the MARVEL project. More specifically, EdgeSec TEE is based on the technology of Trusted Execution Environments that enable the confidential processing of sensitive data and execution of parts of code that should not be exposed. EdgeSec TEE is available through a container that can operate on top of modern Intel processors since it is dependent on the Intel SGX technology.

In the following sections, we provide some background information regarding TEEs, containers, and the state-of-the-art on these two topics. Then, we describe the basic development and deployment details, while we locate the related project and component KPIs and we discuss about their correlation with EdgeSec TEE and how they can be realised within the context of MARVEL. Due to the very specific technical requirements of the EdgeSec TEE underlying technology, the component will be fully integrated after M18, so the final version of this deliverable (i.e., D4.5) will contain details about the results of EdgeSec TEE.

3.1 Background

In this section, background information regarding Trusted Execution Environments (TEEs) and containers will be briefly presented. This information will allow the reader to be introduced with the basics of the main technologies that will be used and examined in this section. Finally, the state-of-the-art in secure containers will be discussed.

3.1.1 Trusted Execution Environments

A Trusted Execution Environment (TEE) is an area within the main processor that allows secure, protected, and confidential execution. TEEs guarantee that the code itself and the data that needs to be processed are located inside protected and isolated execution environments that enable confidentiality and integrity, even if processed in untrusted environments. An untrusted environment could be, amongst others, the operating system, the hypervisor, the drivers, the management stack, the system's memory, and I/O devices. Furthermore, in cases of outsourced applications, even in a seemingly healthy environment, there is always the possibility of an honest-but-curious cloud provider, willing to learn and extract information regarding the users or the system utilisation. Several vendors incorporate hardware technologies that can be utilised to implement TEEs; as for instance, AMD's Secure Processor⁶, ARM's TrustZone⁷, MultiZone Security Trusted Execution Environment from RISC-V⁸, and Intel's Software Guard Extensions (SGX)⁹. In the context of MARVEL, FORTH chooses to employ the Intel SGX technology since it is a mature, well-documented technology, which is also widely examined by the research community.

Intel SGX is a hardware-assisted mechanism in the form of an instruction set architecture (ISA) extension to the Intel architecture. It allows secure attestation and sealing to application software that is executed in a secure environment. This secure environment is called "enclave". The main purpose of these extensions is the protection of selected code parts and data from disclosure or modification in untrusted environments. The enclaves are protected by the CPU that

⁶ <https://www.amd.com/en/technologies/pro-security>

⁷ <https://www.arm.com/technologies/trustzone-for-cortex-a>

⁸ <https://hex-five.com/multizone-security-tee-riscv/>

⁹ <https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/overview.html>

is in charge of any access to the enclave memory or other protected areas of execution. Any instruction that reads or writes to the enclave and is not part of the enclave itself, is prohibited. Assuming an untrusted or even a malicious operating system, hypervisor or firmware, Intel SGX is able to protect the confidentiality of the enclave pages. An Intel SGX application is divided into the following parts (i.e., untrusted and trusted), as depicted in Figure 23.

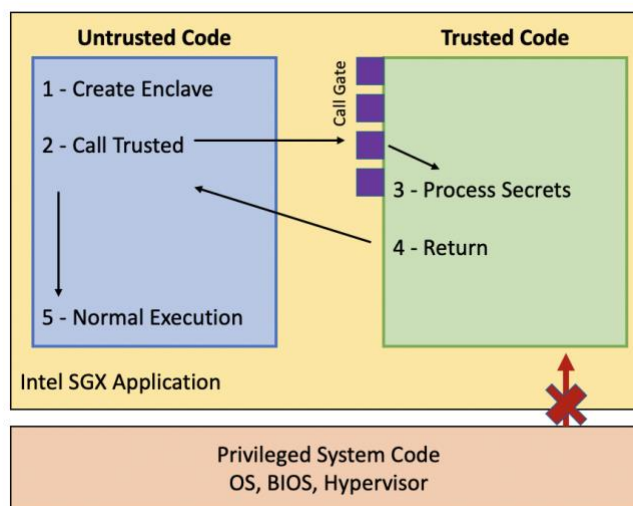


Figure 23: An Intel SGX application is divided into an untrusted and trusted part. Privileged system code does not have access to the trusted part of the Intel SGX application at any time

The code and data that are part of the enclave are stored in a DRAM subset, namely the Processor Reserved Memory (PRM). PRM has a contiguous range, which is not accessible by any system software or other peripherals. Moreover, the contents of the enclaves are stored in the Enclave Page Cache (EPC), a subset of PRM. Software that is not part of the enclave is not able to access the EPC. For Intel Skylake CPUs, the EPC size is between 64 MB and 128 MB and SGX provides a paging mechanism for swapping pages between the EPC and untrusted DRAM. The data of the enclave that has to be written to the disk is encrypted and checked for its integrity. Every time that data are transferred from the cache to the main memory, they get encrypted via an extra on-chip memory encryption engine (MEE). The enclave is decrypted only within the CPU itself, and it is accessible only for code and data that are part of the same enclave. This provides protection to the code from being accessed and examined by other code. Between enclaves, SGX enables local attestation. Additionally, in the case of a third-party application or software, SGX allows remote attestation to ensure that the application is uncompromised and therefore can be trusted. SGX enables the remote system to establish a connection with the enclave, using an end-to-end encrypted channel.

3.1.2 Containers

A container is a standalone system entity consisted of software code that is packed with all the required packages and libraries (i.e., dependencies) enabling an application to be executed consistently and self-sufficiently across different computing environments. Concisely, a container image includes everything needed to enable an application run, offered as an executable package of software¹⁰. At runtime, container images become containers enabling execution.

¹⁰ <https://www.docker.com/resources/what-container/>

Docker is the most popular platform that enables the creation and management of containers. It is a runtime environment that uses Docker images to deploy applications or software in a containerised fashion, built on top of popular operating systems and distributions. The orchestration of different containers in operation can be handled by a container orchestrator, like Kubernetes. Kubernetes is an open-source project that enables the deployment, scalability, and management automation of multiple containers with multiplexed applications.

Containers are offered like a virtualised operating system and share similar functionalities with VMs; yet, they should not be confused. Their differences are illustrated in Figure 24. To begin with, VMs run on top of a hypervisor and each VM is comprised of its own guest operating system with any necessary libraries and files. Taking into consideration that on a single physical machine several VMs can be present, there is a significant consumption of resources and overheads. A container, on the other hand, shares the same host OS or kernel, something that makes it lighter, with essentially less overhead.

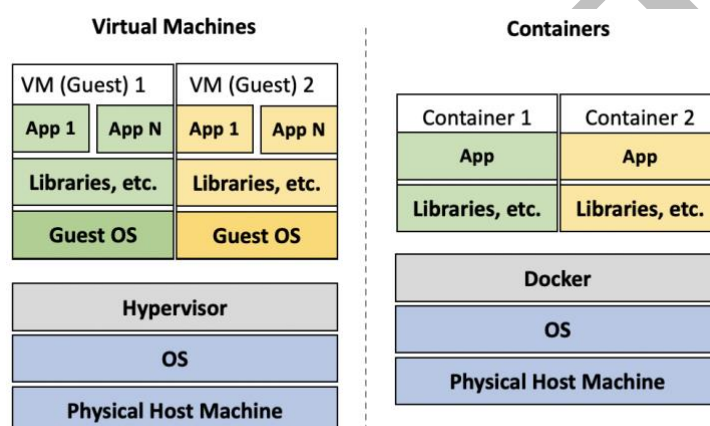


Figure 24: Differences between VMs and containers

3.1.3 State-of-the-Art

Trusted Execution Environments (TEEs) have been widely examined by the research community with the aim to enable the confidentiality of code execution and user data in environments that should not be trusted. As the need for lower costs, higher performance, and scalability rises, outsourcing applications to the cloud has become common. As already mentioned, TEEs, such as Intel SGX, can guarantee data and code protection. Thus, a significant number of works focus on proposing the exploitation of this technology for outsourced applications in the cloud. For instance, VC3 [1] and Opaque [12] offer privacy-preserving data analytics in the cloud using Intel SGX. EnclaveDB [13] is a database engine that can guarantee confidentiality, integrity, and freshness for data and queries. In addition, EndBox [14], ShieldBox [15], and SafeBricks [16] focus on securing middlebox functionality using Intel SGX, while TrustAV [17] is a cloud-based malware scanning solution based on Intel SGX. Andromeda [18] is a framework that provides secure enclaves for the Android OS to mitigate attacks that target sensitive or critical code, data and communication channels. Finally, there are works that enable the execution of unmodified applications within enclaves [19][20][21][22].

SCONE [23] is based on the Intel SGX technology and offers a secure container mechanism for Docker. The design of SCONE leads to (i) a small trusted computing base (TCB) and (ii) a low-performance overhead. More specifically, SCONE offers a secure C standard library interface that transparently encrypts and decrypts the data I/O. To reduce the performance impact of thread synchronisation and system calls within SGX enclaves, SCONE supports user-

level threading and asynchronous system calls. A high-level overview of the secure containerisation with Docker with respect to SCONE is shown in Figure 25.

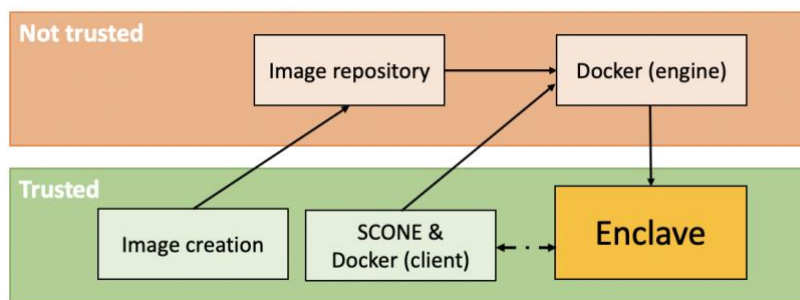


Figure 25: SCONE offers secure containers (with Docker)

Except for SCONE, there are some alternative secure container designs, such as Haven [24] that follows an alternative design. In TensorSCONE [25], authors integrate TensorFlow¹¹ with SCONE to enable secure executions of machine learning computations in untrusted infrastructures. Yet, in TensorSCONE, the GPU utilisation is not supported as in the traditional TensorFlow framework. Finally, Graviton [26] is an architecture for supporting trusted execution environments on GPUs, since it enables applications to offload security- and performance-sensitive kernels and data to a GPU, executing the kernels in isolation from other code running on the GPU.

3.2 Early Deployment and Integration

EdgeSec TEE is based on the SCONE confidential computing technology. The initial version of EdgeSec TEE mechanism is based on SCONE, which is appropriately configured for python applications to run inside Intel SGX enclaves. The initial version of EdgeSec TEE is uploaded on the MARVEL docker image registry and can be downloaded¹² or shared upon request. In this section, we will describe the current development, deployment, and integration status of the component namely EdgeSec TEE with respect to the whole MARVEL platform.

3.2.1 Development

For the development of EdgeSec TEE, a VM is utilised. Intel SGX can be virtualised (i) when the host system supports the Intel SGX technology, (ii) when SGX is enabled either explicitly in the BIOS or via the software enabling procedure, and (iii) when Linux kernel version 5.13 or later is used both in the host and the guest VMs¹³. Both host and guest operating systems are Ubuntu Linux (version 20.04.4 LTS) with a Linux kernel 5.13.4. The CPU is an Intel(R) Core (TM) i7-7700 CPU and a KVM hypervisor is used. The Docker version is 20.10.12.

EdgeSec TEE is provided through the SCONE functionality that supports the execution of Python applications inside the Intel SGX enclaves. SCONE follows precisely the traditional Docker workflow. The Docker workflow enables the creation of Docker images that include the Python engine together with the Python applications. What SCONE does, is to perform the encryption of the Python programs and guarantee their confidentiality and integrity when executed. After the Python engine starts inside an Intel SGX enclave, the SCONE runtime

¹¹ <https://www.tensorflow.org>

¹² <https://marvel-platform.eu/image/docker-sgx>

¹³ Virtualizing Intel SGX: <https://www.intel.com/content/www/us/en/developer/articles/technical/virtualizing-intel-software-guard-extensions-with-kvm-and-qemu.html>

transparently attests the Python engine together with the supporting filesystem and only then, SCONE runtime can get the encryption key in order to initiate the secure execution. SCONE support the execution of Python programs inside SGX enclaves, while there are available several Docker images for the different Python versions and engines.

For EdgeSec TEE, Python 3.7.3 is used (within an Alpine Linux 3.10 OS version). The resulted EdgeSec TEE image can be found on the MARVEL docker image registry and can be downloaded¹⁴ or shared upon request.

3.2.2 Early Deployment

EdgeSec TEE requires an Intel-SGX enabled machine and the installation of the Docker software. Each application that is secured with EdgeSec TEE lays on top of a machine that supports Intel SGX, as it is illustrated in Figure 26. In order to use EdgeSec TEE and take full advantage of the security characteristics that it offers, an application developer needs to follow the following steps: (i) the developer needs to get access to infrastructure that is Intel SGX-enabled, (ii) the developer downloads the EdgeSec TEE's docker image that is uploaded to the MARVEL registry by FORTH, (iii) the developer launches the EdgeSec TEE container from this image, (iv) the developer copies the python application inside the container's file system and installs any required python library or package, (v) the developer executes the python application that is secured by SCONE during the total execution time. In the following paragraphs, the steps after downloading the EdgeSec TEE image are discussed in more detail.

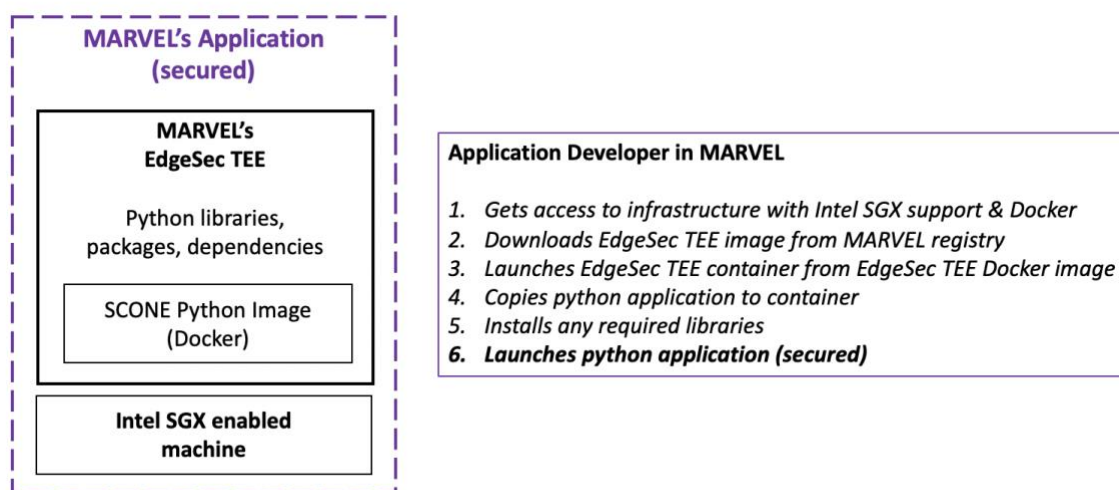


Figure 26: An overview of EdgeSec TEE

Once downloaded from the MARVEL docker image registry, the EdgeSec TEE component can be deployed by the following certain steps:

```
1. docker login registry.marvel-platform.eu
2. docker pull registry.marvel-platform.eu/docker-sgx:0
3. docker run -it registry.marvel-platform.eu/docker-sgx:0
   /bin/sh
```

¹⁴ <https://marvel-platform.eu/login?next=/image/docker-sgx>

An example of a Python command (i.e., *print*) executed within the image of EdgeSec TEE that is uploaded on the MARVEL image registry is shown in Figure 27.

```
vagrant@scone:~$ docker run -it registry.marvel-platform.eu/docker-sgx:0 /bin/sh
/# python3
Python 3.7.3 (default, May 3 2019, 11:24:39)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello MARVEL")
Hello MARVEL
>>>
```

Figure 27: An example python command within the image of EdgeSec TEE

As already mentioned, the Python version that is supported by EdgeSec TEE is 3.7.3 (in an environment of Alpine Linux 3.10). The installation of packages and libraries within the container is possible. For instance, to install the popular Python libraries namely *scipy* and *scikit-learn*, the following commands can be used (i.e., Figure 28). As shown in Figure 29, the installation of the Python libraries is successful, enabling the operation of Python programs to be performed exactly as in traditional setups (i.e., without the support of Intel SGX, SCONE, and EdgeSec TEE). Within the container of EdgeSec TEE, however, the Python application is executed within Intel SGX enclaves, which offer security, code integrity, and data confidentiality. At this point, we have to accentuate that these packages (e.g., *scipy* and *scikit-learn*) are not installed inside the image that is uploaded to the MARVEL registry, and if necessary, they have to be installed following the commands shown in Figure 28 by the developer or user of EdgeSec TEE. Other packages and libraries can be installed following a similar procedure.

```
/ # apk --no-cache add lapack libstdc++ && apk --no-cache add --virtual .builddeps g++ gcc gfortran
musl-dev lapack-dev && pip install scipy scikit-learn && apk del .builddeps && rm -rf /root/.cache
```

Figure 28: Commands to install Python libraries (e.g., *joblib*, *numpy*, *scikit-learn*, *scipy* and *threadpoolctl*) within EdgeSec TEE.

```
Successfully built scipy scikit-learn numpy
Installing collected packages: numpy, scipy, joblib, threadpoolctl, scikit-learn
Successfully installed joblib-1.1.0 numpy-1.21.6 scikit-learn-1.0.2 scipy-1.7.3 threadpoolctl-3.1.0
```

Figure 29: Successful installation of Python libraries within the docker image of EdgeSec TEE (i.e., *joblib*, *numpy*, *scikit-learn*, *scipy* and *threadpoolctl*)

3.2.3 Integration with MARVEL

Up until this point, EdgeSec TEE can be used and tested by any partner that participates in the MARVEL project by downloading the EdgeSec TEE image that exists in the MARVEL registry, following the instructions that were described in the previous section (Section 3.2). Since the integration process is still work in progress, a detailed and final integration plan cannot be completely outlined in the current deliverable. The integration of EdgeSec TEE with the whole MARVEL platform will be fully discussed in the following and final version of this deliverable (i.e., D4.5).

3.3 Use Cases and Related Components

This section is destined to present the MARVEL uses cases that EdgeSec TEE will participate in. In addition, we will offer an overview of the related components that will connect with EdgeSec TEE. EdgeSec TEE is also presented in Figure 1 as part of the subsystem named “Security, Privacy, and Data Protection”. As shown in the figure, EdgeSec TEE can potentially

participate in any layer as long as the hardware and software infrastructure facilitate its deployment requirements. The requirements include but are not limited to an Intel SGX-enabled CPU and a component implemented using the Python programming language.

3.3.1 Related Components

EdgeSec TEE will be fully integrated into the whole MARVEL platform after M18. Thus, at this point, the interconnection of EdgeSec TEE and other MARVEL components is not finalised. Details about the related components of EdgeSec TEE will be shared in the following version of the deliverable (i.e., D4.5).

3.3.2 EdgeSec TEE in MARVEL Use Cases

Similarly, as EdgeSec TEE will be integrated and tested as part of the MARVEL platform after M18, we will provide more details about it and the respecting MARVEL use cases in the following version of the deliverable (i.e., D4.5).

3.4 Early Experimental Results

This section will demonstrate the experiments that were performed to explore the capabilities of EdgeSec TEE. Since EdgeSec TEE will be fully integrated within the MARVEL platform after M18, the experimental results have not been explored yet. This section will be fully described in the following and final version of the deliverable (i.e., D4.5).

3.4.1 Testbed Setup

This section will be fully explored after M18. Yet, in Section 3.2, we describe the testbed setup for the early deployment of EdgeSec TEE before M18. More specifically, for the development and early deployment of EdgeSec TEE, we use a VM that has the following characteristics. The CPU that was used in the experiments performed is an Intel Core i7-7700 operating at 3.6 GHz and the main memory is 4GBytes. The L3 cache (i.e., 16MiB) and the memory controller are shared across the CPU cores and the integrated GPU. Each CPU core is equipped with 64KiB of L1 cache and 512KiB of L2 cache. The docker version is 20.10.12 (build e91ed57). The docker image of EdgeSec TEE has a Linux kernel (version 5.13) with an Alpine Linux 3.10 operating system.

3.4.2 Experiments

Since the experiments of EdgeSec TEE as part of the MARVEL platform will be presented in the following deliverable (i.e., D4.5), we will be able to discuss them in full detail there. Yet in Section 3.2.2, we present some snapshots of EdgeSec TEE enabling the secure execution of a simple python program.

3.4.3 Results

Similarly, this section will be fully explored after M18, when EdgeSec TEE will be fully deployed and connected with other components to participate in MARVEL use cases.

3.5 KPIs

In this section, the Key Performance Indicators (KPIs) that are related to EdgeSec TEE, Task 4.3 and by extension to D4.2 will be presented and discussed.

3.5.1 Project-related KPIs

Table 7 enumerates the project-related KPIs that concern the component EdgeSec TEE, which is implemented in Task 4.3 in the context of the MARVEL platform.

More specifically, the goals of the project-related KPI with identifier *KPI-O3-E3-1* will be achieved by: (i) attesting edge devices to ensure that no untrusted components flood the system with fraud data, (ii) encrypting collected data and establishing secure communication channels between each processing layer, (iii) exploring in-chip memory encryption technologies that will further enhance the security of the processing devices. Secure computing will be achieved using the EdgeSec TEE component in conjunction with EdgeSec VPN. Both components have been uploaded to the MARVEL registry (v0). At its current state, EdgeSec TEE enables programming using the python language and related libraries, such as scikit. In the future, we plan to provide more flexibility in terms of programming languages and libraries supported if it is necessary by other components or the pilot use cases.

Regarding the project-related KPI with identifier *iKPI-2.2*, the system will be protected from the harmful effects of at least three attacks. As already discussed in Section 2.5, FORTH currently provides EdgeSec VPN that secures the network communications and EdgeSec TEE that offers trusted and protected execution.

Table 7: Project-related KPIs that concern EdgeSec TEE

KPI ID	KPI Description	Strategy	Related Task	Related Component
KPI-O3-E3-1	Realise a secure computing framework at all the processing layers.	Ensure that there are no weak links in the E2F2C chain, every layer and communication channel between them shall be secure.	T4.3	EdgeSec VPN, EdgeSec TEE
iKPI-2.2	At least three (3) different cyber threats avoided due to E2F2C.	Cyber threats that could otherwise potentially be exploited to harm the system, but their impact is minimised due to the security features offered by E2F2C.	T4.3	EdgeSec VPN, EdgeSec TEE

3.5.2 Component-related KPIs

The component-related KPI that concerns EdgeSec TEE is presented in Table 8. More specifically, with EdgeSec TEE, MARVEL aims to effectively avoid at least three distinct cyber threats. As already mentioned, EdgeSec TEE offers security and privacy guarantees in environments that should not be trusted, since it is based on the Intel SGX technology that offers a combo of cybersecurity, encryption, and verification capabilities. Some of the attacks that Intel SGX targets for mitigation are software and physical attacks, memory mapping attacks, and cache timing attacks amongst many others [27]. For instance, attacking the system memory (DRAM) of a machine is a serious and common threat, while it has been shown that an adversary with physical access to a machine can potentially read and/or modify memory contents. With Intel SGX, an autonomous hardware unit called the Memory Encryption Engine (MEE) is offered, which is able to protect the confidentiality, integrity, and freshness of the CPU-DRAM traffic over some memory range [28]. In addition, even though CPU side-channel attacks are out of the threat model scope of the Intel SGX technology, Intel stays up-to-date and

constantly provides guidelines to help developers harden their code¹⁵, while they also offer bug bounty competitions to encourage researchers to find and disclose new side-channel vulnerabilities¹⁶. Thus, EdgeSec TEE greatly contributes to the realisation of this specific KPI, by effectively avoiding physical and cyber threats.

Table 8: Component-related KPIs that concern EdgeSec TEE

KPI	Metric	Expected Result	Relevant Project KPI
Effectiveness for Avoiding Cyber Threats	Number of threats avoided	3 distinct cyber threats avoided	<i>KPI-O3-E3-1,</i> <i>iKPI-2.2</i>

¹⁵ <https://www.intel.com/content/dam/develop/external/us/en/documents/180204-sgx-sdk-developer-guidance-v1-0.pdf>

¹⁶ <https://www.intel.com/content/www/us/en/security-center/bug-bounty-program.html>

4 GPU-accelerated Stream Processing on the Edge (GPURegex)

In this section, GPURegex is presented, described, and evaluated based on the performance achieved as a standalone component. GPURegex is the component that brings performance acceleration features to the complete E2F2C framework, developed within the MARVEL project. More specifically, GPURegex is based on the Single Instruction – Multiple Data (SIMD) type of parallel processing, taking advantage of modern processors (either CPUs or accelerators), and OpenCL, a framework for writing programs that uniformly execute across heterogeneous platforms. The component is offered as an OpenCL program that is able to be executed on top of a hardware device when the proper runtime and libraries are installed. GPURegex offers the pattern matching functionality, accelerated.

In the following sections, we provide some background information regarding GPU-accelerated stream processing and pattern matching, as well as the state-of-the-art on these two topics. Then, we describe the development and deployment details, while we locate the related project and component KPIs. Finally, we discuss about their correlation with GPURegex and how they can be realised within the context of MARVEL.

GPURegex will be used to accelerate the searching of keywords against audio and video captions (offered by the AAC tool of TAU). Since the input will be only available after M18, we allow the evaluation of the tool using several public datasets with audio or video captions identified and shared by TAU. We expect that these data formats will fairly resemble the actual data resulted from AAC.

4.1 Background

In this section, basic background information regarding GPU acceleration of stream processing and pattern matching will be presented. In addition, we will discuss the state-of-the-art that concerns GPU-acceleration of stream processing and pattern matching.

4.1.1 GPU-Accelerated Stream Processing

Processing in a streaming fashion is a programming technique that facilitates parallel processing. Image, video, and signal processing were originally suited for stream processors, while presently general-purpose computing can be achieved when the nature of the processing is computationally heavy, not memory intensive. GPUs, due to their architectural design are ideal for fast and highly-parallelised computations, since their numerous, powerful cores enable this kind of execution.

The continuous processing of data streams is useful in numerous applications, such as network traffic inspection and log processing, AI and real-time predictions and continuous monitoring of healthcare/transportation/manufacturing data amongst others. Thus, accelerating stream processing for near real-time scenarios' requirements is crucial but feasible – realised by streaming processors and SIMD-enabled hardware, like GPUs.

4.1.2 GPU-Accelerated Pattern Matching

Pattern matching is the procedure of identifying if a certain keyword (i.e., pattern) is part of an expression (i.e., input). Due its computational characteristics and requirements, pattern matching processing can be parallelised (depending on the algorithm used) allowing streaming capabilities.

GPURegex is based on the Aho-Corasick algorithm [29]. Aho-Corasick is one of the most widely used algorithms for string pattern matching and is the optimal solution for multiple patterns searching, since it enables simultaneous pattern matching. This simultaneous matching

can be achieved when the set of patterns is preprocessed. In the preprocessing phase, one automaton is being built, which will be eventually used in the matching phase. Each character of the text-based input is processed only once during the matching phase. The Aho-Corasick algorithm has the property that, theoretically, the processing time does not explicitly depend on the number of patterns. Given a set of patterns, the algorithm constructs a pattern matching machine (i.e., the automaton), that matches all patterns against the input one byte at a time. The implementation of GPURegex is based on the Aho-Corasick algorithm due to the processing nature, which enables the easy parallelisation of the underlying computations.

In the following sections, we describe how Aho-Corask is used and implemented for GPURegex.

4.1.3 State-of-the-Art

Pattern matching is the core operation of several network packet processing applications, such as firewalls, intrusion detection, L7 filtering, and traffic classification. Thus, it is very common when searching for pattern matching applications, to encounter works in the domain of network inspection. However, pattern matching is not only destined for network processing applications. For instance, pattern matching is used for system log processing, continuous monitoring of healthcare/transportation/manufacturing data and bioinformatics, like RNA structure alignments. Thus, there are numerous and diverse research domains that are benefitted from optimised versions of traditional pattern matching algorithms.

In the meantime, GPUs have become very popular due to a substantial performance boost that provide to many individual network traffic inspection applications that are based on the parallelisation of the pattern matching computations. Related works include but are not limited to GPU-accelerated intrusion detection [30][31][32][33], cryptography [34], and IP routing [35]. For instance, Gnort accelerates the pattern matching engine of the Snort IDS using a discrete GPU. Similarly, Kargus performs load balancing in pattern matching workloads and is compatible with Snort IDS [36]. MIDeA offers a multi-parallel intrusion detection architecture tailored to multi-queue NICs, multiple CPUs, and multiple GPUs [33]. DFC offers accelerated string matching tailored to packet processing by reducing memory accesses and cache misses [37].

In addition, there have been proposed several programmable network traffic processing frameworks, such as Snap [38] and GASPP [39], that manage to simplify the development of GPU-accelerated network traffic processing applications.

Other works take advantage of the shared integrated GPU that is packed with the main processor in the same die in order to accelerate or offload network packet processing applications [40][41][42]. In APUNet, authors propose the utilisation of integrated GPUs to accelerate packet processing workloads without paying the overheads of memory transactions between the host and discrete GPUs [40]. Papadogiannaki et al. [41] have proposed a scheduling approach that, based on performance policies (such as high throughput or low power consumption), determines the most suitable combination of heterogeneous devices (i.e., CPU, integrated or discrete GPUs) for efficient execution of network packet processing workloads (such as DPI or network packet encryption). In NBA, authors extend the functionality of a network router to leverage hardware accelerators for network packet processing load balancing [43]. In addition, there are works that perform GPU-accelerated pattern matching for metadata searching to enable encrypted network traffic analysis and inspection [44][45].

4.2 Early Deployment and Integration

GPURegex is a real-time high-speed pattern matching engine that leverages the parallelism properties of general-purpose GPUs (GPGPUs) to accelerate the process of string and/or regular expression matching. The initial version of GPURegex is uploaded on the MARVEL docker image registry and can be downloaded or it can be shared upon request. More specifically, GPURegex is available for two different hardware setups: (i) a hardware setup with an Intel CPU¹⁷, and (ii) a hardware setup with an Intel CPU and an integrated (on chip) GPU¹⁸. In the first hardware setup, the drivers that are installed in the container are destined for OpenCL-enabled CPUs, while in the second hardware setup, the drivers that are installed in the container are destined for shared, integrated, OpenCL-enabled GPUs, like Intel HD Graphics.

In this section, we will describe the current implementation, deployment and integration status of the component namely GPURegex with respect to the whole MARVEL platform.

4.2.1 Implementation

As already discussed, pattern matching includes intensive computations and can be significantly accelerated using the right hardware architectures and an algorithm implementation with operations that can be parallelised. GPURegex is implemented based on the Aho-Corasick algorithm and pre-compiled DFA automata. Thus, GPURegex is able to perform simultaneous multi-pattern matching within a single pass of the input. A high-level overview of GPURegex as it is used in MARVEL can be found in Figure 30. In this section, we discuss the implementation details of the component GPURegex.

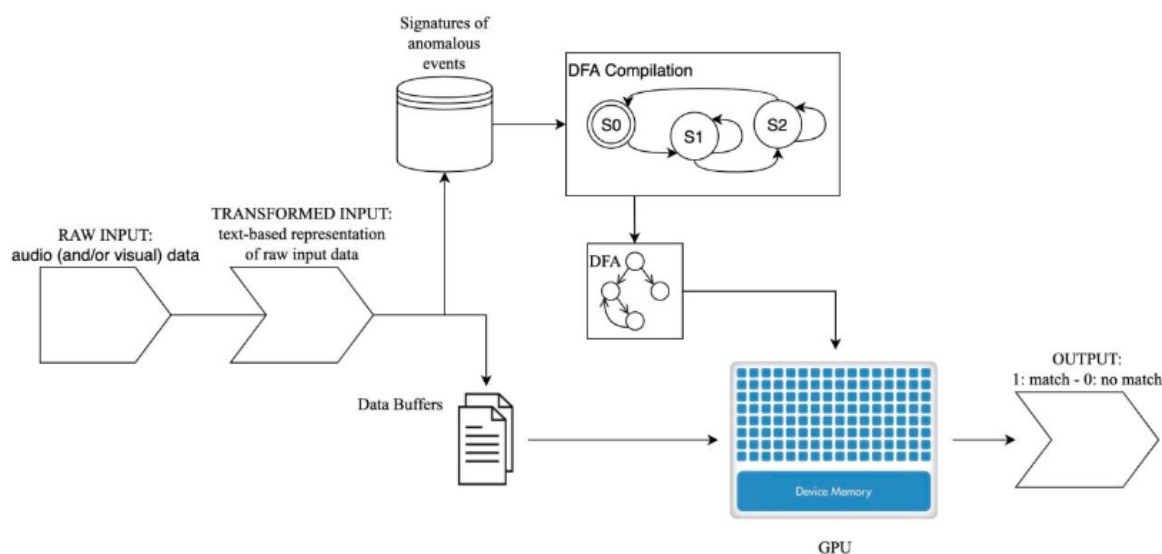


Figure 30: High-level overview of GPURegex in MARVEL

GPURegex supports string searching and regular expression matching operations. For the development of GPURegex, all the GPU-accelerated pattern matching operations are implemented using the OpenCL library, provided through a C API. GPURegex enables the processing of the incoming input, which is text-based, and when the processing is over it returns the reported matches. In the context of MARVEL, GPURegex will be used to accelerate the

¹⁷ <https://marvel-platform.eu/login?next=/image/gpuregex-intel-cpu>

¹⁸ <https://marvel-platform.eu/login?next=/image/gpuregex-intel-gpu>

intensive pattern matching operations for fast event detection, even though traditionally, this component has been specifically designed to accelerate the pattern matching procedure of security applications such as network intrusion detection systems, load balancers, and firewalls [31][41][44][45].

The pattern matching procedure is described as follows. Initially, the patterns are compiled into DFA state machines and state transition tables. A GPURegex user can compile a single pattern to a single DFA or combine different patterns into a single one. The compilation process is performed once, before the execution of the pattern matching engine, and thus, it can be performed offline by the CPU, during the initialisation phase of the user application without adding any runtime overheads. Depending on the GPU architecture, the state table is copied to the memory of the discrete GPU or mapped to the memory space that is shared between the CPU and the integrated GPU. During the pattern matching phase, each thread searches against a different portion of the input data. The algorithm processes the input one character (one character corresponds to one byte) at a time and for each consumed byte, the matching algorithm switches the current state according to the state transition table. The size of the state transition table is *number of states X alphabet size*. For instance, to support the standard ASCII character-set for both patterns and input, the alphabet size is 128. The *number of states* is completely dependent on the patterns and the resulted DFA state machine. A simplified example is shown in Figure 31.

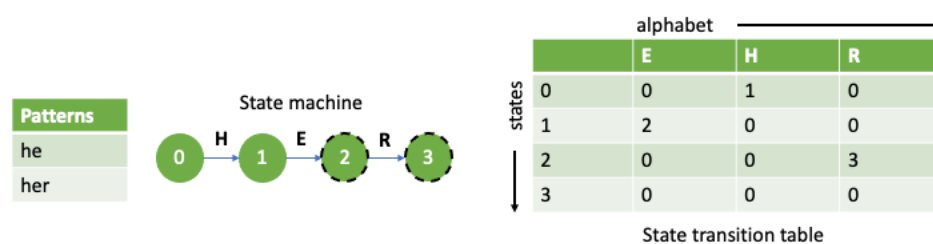


Figure 31: Construction of the state transition table

As already mentioned, for a GPU to process data, we need first to copy the data to the corresponding GPU memory space from the main memory. When the GPU is discrete, these data transfers add a time overhead due to the PCIe bus, which is relatively slow and it is becoming an overhead when the application is memory intensive and requires constant memory transfers (e.g., CPU-GPU-CPU). An illustration of the memory transfers' requirements in an integrated-GPU format versus a discrete-GPU format is presented in Figure 32. Thus, in the very first version of GPURegex (GPURegex v.0), we introduce an implementation specifically for OpenCL-enabled integrated GPUs (such as Intel HD Graphics) that share the same memory space with the main processor¹⁹. In addition, we implemented another version of GPURegex to target OpenCL-enabled processors (such as Intel CPUs) for cases where a shared GPU is not present in the hardware setup that is available²⁰.

¹⁹ <https://marvel-platform.eu/login?next=/image/gpuregex-intel-gpu>

²⁰ <https://marvel-platform.eu/login?next=/image/gpuregex-intel-cpu>

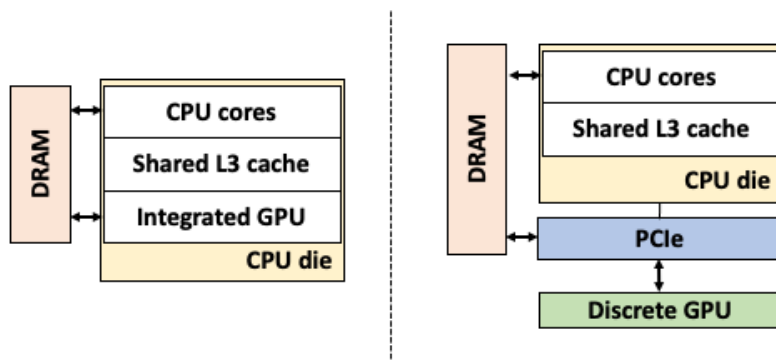


Figure 32: Architectural comparison of an integrated GPU, packed with the main processor in the same CPU die versus a discrete, dedicated GPU

4.2.2 Early Deployment

GPURegex can be deployed to any OpenCL-enabled processor or hardware accelerator, such as dedicated, discrete GPUs or shared, integrated GPUs. In the first version of GPURegex that has been uploaded to the MARVEL registry and is available to any MARVEL partner, FORTH introduces an implementation for integrated GPUs²¹ and an implementation for main processors²² for hardware setups that do not offer a GPU.

As illustrated in Figure 33, a GPURegex Docker container can be deployed on top of any OpenCL-enabled hardware device. OpenCL drivers are required to be installed in the specific docker container before the execution of GPURegex. Each vendor (e.g., Intel, NVIDIA) and each hardware device (e.g., CPU, discrete GPU, integrated GPU) is supported by vendor and device specific OpenCL drivers. For instance, the OpenCL drivers that are destined for Intel integrated GPUs are different to those that are destined for NVIDIA GPUs. Thus, normally, a new OpenCL driver must be installed for every hardware device change. As already stated, GPURegex is available via two images, uploaded to the MARVEL image registry (i.e., Intel CPU and Intel HD Graphics GPU).

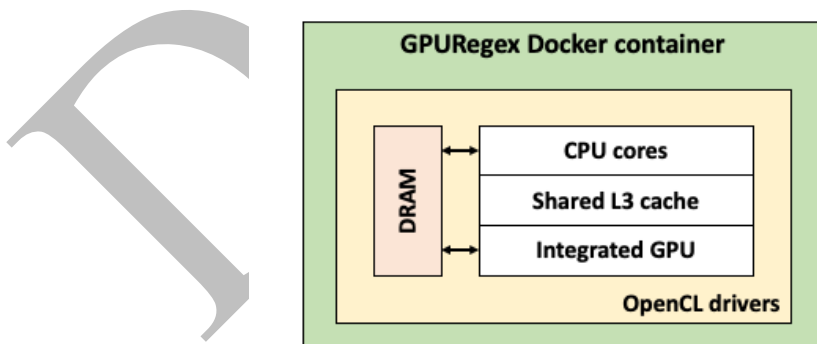


Figure 33: An overview of GPURegex

Once downloaded from the MARVEL docker image registry, the GPURegex component can be deployed by the following certain steps:

²¹ <https://marvel-platform.eu/login?next=/image/gpuregex-intel-gpu>

²² <https://marvel-platform.eu/login?next=/image/gpuregex-intel-cpu>

1. `docker login registry.marvel-platform.eu`
2. `docker pull registry.marvel-platform.eu/gpuregex-intel-cpu:1`
3. `docker run -it registry.marvel-platform.eu/gpuregex-intel-cpu:1 /bin/sh`

An example of GPURegex’s execution in the container loaded by the image uploaded on the MARVEL image registry is shown in Figure 34. We can see that GPURegex returns the input lines that contain patterns that match against them. In this specific example, patterns “pattern01” and “pattern02” are matched, while pattern “pattern00” is not contained in the input file. The contents of the pattern file, namely “patterns_demo”, and the contents of the input file, namely “input_demo”, are presented in the same figure. GPURegex is compiled using the command: `$ make`, and executed using the command: `$./bin/gpuregex -p patterns_demo -i input_demo`, where `-p` accepts the pattern file name and `-i` accepts the input file name.

```

vagrant@scone:~$ docker run -it gpuregex-intel-cpu:2 /bin/bash
root@f1af7b1c4fe9:/usr/gpuregex# ./bin/gpuregex -p patterns_demo -i input_demo
max state: 9
max states: 10
trc: 10
this line contains pattern pattern01
this line contains pattern pattern02
root@f1af7b1c4fe9:/usr/gpuregex# cat patterns_demo
pattern00
pattern01
pattern02
root@f1af7b1c4fe9:/usr/gpuregex# cat input_demo
this line does not contain a pattern
this line contains pattern pattern01
this line neither contains a pattern
this line contains pattern pattern02
root@f1af7b1c4fe9:/usr/gpuregex#

```

Figure 34: An example run of GPURegex inside the container destined for Intel CPUs

4.2.3 Integration with MARVEL

Up until this point, GPURegex can be used and tested by any partner that participates in the MARVEL project by downloading the GPURegex images that exist in the MARVEL registry, following the instructions that were described in the previous section (Section 4.2.2). Since GPURegex - in the context of the MARVEL project - will participate in a pipeline receiving input from a component that can only be deployed after M18 (i.e., AAC from TAU), the entire deployment and integration details of GPURegex will be fully outlined in the following deliverable (i.e., D4.5). Thus, in these sections, FORTH describes the testbed setup that is located at FORTH premises and is used for the development of the GPURegex that has been uploaded to the MARVEL registry.

4.3 Use Cases and Related Components

In this section, we briefly describe the component Automated Audio Captioning (AAC) that is planned to pair with GPURegex in a processing pipeline, in the context of the MARVEL project. Yet, since AAC will be ready after M18, the entire deployment and integration details of GPURegex will be fully outlined in the following deliverable (i.e., D4.5). As a component of the whole MARVEL platform, GPURegex is presented in Figure 1. As part of the subsystem named “Optimised E2F2C Processing and Deployment”, GPURegex’s relation with the AAC component is shown. In this conceptual architecture figure, GPURegex is placed within the cloud layer, but can also be trained or operate in different layers of the platform. More details will be described in the following version of this deliverable (D4.5).

4.3.1 Related Components

AAC is a cross-modal translation task in artificial intelligence that connects audio processing and natural language processing. The automated audio captioning component generates a descriptive textual description to describe the content of an audio clip. These textual descriptions can be used to assist the decision-making process in other components. The AAC component can collaborate and enhance the predictive behaviour of other components by providing high-level information about the audio content.

The input to the AAC component is an audio signal, and the output is a sentence that describes the audio signal, for instance, “Birds chirp while people talk in the background”. The output is a properly formulated sentence that not only describes the sound events but also the spatial-temporal relation between different objects as well as the activities involved. In the learning stage, the component learns the mapping between the input audio signal and the corresponding captions, and at the inference stage, the component predicts the captions for audio input. The learned knowledge depends on the data utilised during the learning stage. The system development requires an audio dataset consisting of audio samples and the corresponding manually generated reference captions. The current state-of-the-art for the AAC system is based on relatively large neural networks with encoder-decoder architectures or transformer-based sequence-to-sequence architectures. An overview of the AAC component is presented in Figure 35.

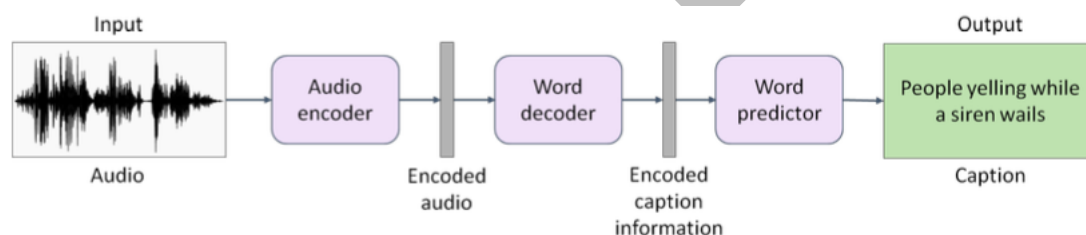


Figure 35: Overview of AAC

4.3.2 GPURegex in MARVEL Use Cases

As already discussed, GPURegex in the context of the MARVEL project will participate in a pipeline and will receive input from the AAC component that can only be deployed after M18. Thus, the entire deployment and integration details of GPURegex will be fully outlined in the following deliverable (i.e., D4.5). In this deliverable, we test GPURegex using several public datasets that contain captions from audio and video sources. These datasets are close to the input that GPURegex will receive from AAC. So, we expect that the differences of this early experimental results will not be significant to those of the actual deployment of GPURegex in the context of MARVEL.

4.4 Early Experimental Results

In this section, FORTH will present the early experimental results of GPURegex in detail.

4.4.1 Testbed Setup

GPURegex in its current form, can be executed on top of an Intel CPU and on top of an Intel integrated GPU, and it is available in the MARVEL image registry via two different images “gpuregex-intel-cpu” and “gpuregex-intel-gpu”. Since a CPU, rather than a graphics processor, is the most typical device in a hardware setup of an experimental environment, in this section we present the experimental results of “gpuregex-intel-cpu”.

For the performance evaluation of GPURegex, we use a VM that has the following characteristics. The CPU that was used in the experiments performed is an Intel Core i7-7700 operating at 3.6 GHz and the main memory is 4GBytes. The L3 cache (i.e., 16MiB) and the memory controller are shared across the CPU cores and the integrated GPU. Each CPU core is equipped with 64KiB of L1 cache and 512KiB of L2 cache.

The docker image has a Linux kernel (version 5.13) with an Ubuntu 16.04.7 LTS operating system. The OpenCL version installed is 2.1, with platform name “Intel(R) CPU Runtime for OpenCL (TM) Applications” and driver version 18.1.0.0920.

4.4.2 Experiments

As already mentioned, GPURegex in the context of the MARVEL project will participate in a pipeline that will receive input from the AAC component that can only be deployed after M18. Since there is no available input for GPURegex yet, in this deliverable we will use several public datasets that include captions extracted from audio and video (in several formats: e.g., csv, json, yaml) and could resemble the output of AAC. TAU specifically proposed the following captioning datasets: the Clotho dataset²³, the AudioCaps dataset²⁴, and the MACS dataset²⁵. Another dataset that was proposed by TAU, was the Audio Caption Hospital dataset²⁶. Unfortunately, processing was not possible due to the character encoding that was not part of the ASCII character-set that GPURegex supports (256 ASCII character-set).

The Clotho dataset consists of three main csv files, namely `clotho_captions_development.csv` (1.3MBytes), `clotho_captions_evaluation.csv` (354KBytes), `clotho_captions_validation.csv` (360KBytes). Each file contains a number of lines and each line contains five captions extracted by one audio/video file. A line looks like the following example:

```
“Distorted_AM_Radio_noise.wav, A muddled noise of broken channel of the TV, A television blares the rhythm of a static TV., Loud television static dips in and out of focus, The loud buzz of static constantly changes pitch and volume., heavy static and the beginnings of a signal on a transistor radio”
```

For each one of these files, there is a respecting metadata file that contains -- amongst others - - the keywords that were exported by those captions: “Distorted_AM_Radio_noise.wav, **noise ; radio**, ...”. To construct our pattern file, we extract the unique keywords and we save them into a pattern file. The pattern file consists of 6623 unique fixed strings. Searching for the patterns against the input file `clotho_captions_development.csv` results in a total of 3840 matches (out of the 3840 total sentences). This means that all 3840 input lines contain at least one of the patterns.

The AudioCaps dataset consists of three csv files, namely `train.csv` (3.5MBytes), `test.csv` (393KBytes), `val.csv` (168KBytes). Each file contains a number of lines and each line contains a single caption for a single audio capture. A line looks like the following example: “91139, r1nicOVtvkQ, 130, A woman talks nearby as water pours” (the four columns correspond to the following: `audiocap_id`, `youtube_id`, `start_time`, `caption`). To construct our pattern file, we extract the unique keywords and save them into a pattern file. The pattern file consists of 7414 unique fixed strings. Searching for the patterns against the input

²³ <https://zenodo.org/record/4783391#.YovyVS0RpTa>

²⁴ <https://github.com/cdjkim/audiocaps>

²⁵ <https://zenodo.org/record/5114771#.YovyXC0RpTa>

²⁶ <https://zenodo.org/record/4671263#.YovyWS0RpTa>

file train.csv results in a total of 49839 matches (out of the 49839 total sentences). This means that all 49839 input lines contain at least one of the patterns.

The MACS dataset contains a single file, namely MACS.yaml (2.7MBytes). An example of the entries that are contained in the file follows:

```
files:
- filename: airport-barcelona-0-0-a.wav
  annotations:
  - annotator_id: 233
    sentence: a person whistling and singing
    tags:
    - adults_talking
    - music
  - annotator_id: 105
    sentence: people are talking whistling and singing
    tags:
    - adults_talking
```

To construct our pattern file, we extract the tags in the file and keep the unique keywords. In addition, if there is the character “_” between two different words, we split the tags and keep them both. This results in a pattern file with 16 unique keywords (fixed strings). Concerning the input file, we extracted only the “sentence” fields from the yaml file, in order to avoid patterns matching against the tags themselves (the exported file size is 933KBytes). As is, the patterns that match against the input lines result in 14573 matches (out of the 17275 total sentences). Since the keywords contain words like “talking” or “voices”, we add a pre-processing step to keep the origins, like “talk” or “voice”, respectively. This action results in a total of 15284 matches. This means that 15284 input lines include at least one of the patterns.

4.4.3 Results

Since the related KPIs address the performance metrics of GPURegex to be throughput and execution time, in this section we will perform two corresponding benchmarks that concern those two metrics. To begin with, we evaluate the processing performance of GPURegex using the three public datasets shared by TAU (i.e., Clotho, AudioCaps, and MACS), using the corresponding pattern files and inputs. We compare the performance achieved by the GPURegex executed on top of the CPU versus the GNU grep utility that is also based on the Aho-Corasick algorithm (using the -F option that enables fixed strings searching) and runs on the CPU, as well. In addition, we build several pattern files (with different state transition counts) and input files (with different sizes) and we execute GPURegex to present how the processing performance achieved is based on these parameters. We perform the same benchmarks (same pattern file, same input files) with the GNU grep utility to enable performance comparisons.

For the Clotho dataset, we use the pattern file that was constructed by the keywords that were extracted by the captions. Each one of these patterns is a fixed string, with a total of 6623 patterns. The pattern sizes are very diverse with the maximum sized pattern being 62Bytes while the shortest pattern is 1Byte. The resulting automaton has 20788 state transitions. The average GPURegex processing throughput is 3282 Mbits/second (among 30 GPURegex executions using the same pattern file and input file that is 1.3MBytes). Similarly, the average processing execution time (latency) is 3500 microseconds, whereas the CPU version of Aho-Corasick achieves processing in 12000 microseconds. This means that the speedup of

GPURegex, in this case, is more than x3, due to the parallel processing, and specifically due to the Single Instruction/Multiple Data (SIMD) model that OpenCL facilitates.

For the AudioCaps dataset, we use the pattern file that was constructed by the keywords that were extracted by the captions (found in the fourth column). Each one of these patterns is a fixed string, with a total of 7414 patterns. The pattern sizes are also diverse with the maximum sized pattern being 19Bytes while the shortest pattern is 2Bytes. The resulting automaton has 14465 state transitions. Even though the number of the patterns is larger than in the first example, their characteristics (length reaches up to 19Bytes versus the length of the 62Bytes from the first automaton) lead to a shorter state transitions table. The average GPURegex processing throughput is 3200 Mbits/second (among 30 GPURegex executions using the same pattern file and input file that is 3.5MBytes). Similarly, the average processing execution time (latency) is 2000 microseconds, whereas the CPU version of Aho-Corasick achieves processing in 28000 microseconds. This means that the speedup of GPURegex, in this case, is more than x10. This essential performance difference appears due to the large input file size. GPURegex handles the input files, optimally, with respect to maximising processing performance and enabling full data parallelisation and SIMD processing.

For the MACS dataset, we use the pattern file that was constructed by the tags that were extracted by the sentences and were part of the yaml file provided in the dataset. Each one of these patterns is a fixed string, with a total of 16 patterns. The maximum sized pattern is 12Bytes while the shortest pattern is 3Bytes. The resulting automaton has 68 state transitions, due to the short pattern file. The average GPURegex processing throughput is 6748 Mbits/second (among 30 GPURegex executions using the same pattern file and input file that is 933KBytes). Similarly, the average processing execution time (latency) is 626 microseconds, whereas the CPU version of Aho-Corasick achieves processing in 3500 microseconds. This means that the speedup of GPURegex in this case is more than x5.

For the last experiment, we generate two synthetic input files and one pattern file. The synthesised pattern and input files are generated using a simple python script, which takes as arguments the number of lines to be generated and the number of characters for each line, printing the corresponding randomly selected ASCII alphanumeric characters. Both input files contain 10K lines of 100Bytes (its size is 987KBytes) and 1500Bytes (its size is 15MBytes) per line, respectively. The pattern file consists of 1000 lines with 10Bytes pattern length. For the microbenchmarks, we measure the throughput and execution time. Again, we measure the processing performance of GPURegex, comparing it to the processing performance of GNU Grep which is built for CPU execution. We present the throughput and execution time in two separate tables (i.e., Table 9 and Table 10). More specifically, using the smaller input file, GPURegex achieves an average processing throughput of 3200 Mbits/second (the average execution time is 660ms), while GNU Grep exits with an average processing throughput of 789Mbits/second (the average execution time is 10000ms). Using the larger input file, GPURegex achieves an average processing throughput of 3330 Mbits/second (the average execution time is 7200ms), while GNU Grep exits with an average processing throughput of 723Mbits/second (the average execution time is 166000ms).

In all experiments, GPURegex succeeds better performance, when compared to the GNU Grep utility.

Table 9: Processing throughput of GPURegex and GNU Grep (measured in Mbits/second)

Throughput	GPURegex		GNU Grep	
Input Length	100B (length per line)	1500B (length per line)	100B (length per line)	1500B (length per line)
10K Lines	3200 Mbits/sec	3330 Mbits/sec	789 Mbits/sec	723 Mbits/sec

Table 10: Processing time of GPURegex and GNU Grep (measured in microseconds)

Execution Time	GPURegex		GNU Grep	
Input Length	100B (length per line)	1500B (length per line)	100B (length per line)	1500B (length per line)
10K Lines	660 ms	7200 ms	10000 ms	166000 ms

4.5 KPIs

In this section, FORTH will present the relation of the GPURegex component to the project- and component-related KPIs in the context of the MARVEL project.

4.5.1 Project-related KPIs

The project-related KPI that concerns the component, namely GPURegex, which is implemented in Task 4.3 in the context of the MARVEL platform is presented in Table 11. More specifically, the KPI proposes the acceleration of the pattern matching procedure after the utilisation of the GPURegex component. The processing performance is calculated with the performance metrics of throughput and latency. As discussed in Section 4.4, the sustained processing performance achieved using the OpenCL-enabled CPU offers significant processing speedups, more than 10% of a similar CPU implementation.

Table 11: Project-related KPIs that concern GPURegex

KPI ID	KPI Description	Strategy	Related Task	Related Component
KPI-O1-E1-2	Increase of data throughput and decrease of access latency by 10%.	Access latency is defined by the response time of the overall system, while the throughput is defined as the amount of data that can be processed per unit of time.	T4.1, T4.2	GPURegex

4.5.2 Component-related KPIs

The component-related KPI that concerns GPURegex is presented in Table 12. As similarly stated in Section 4.5.1, GPURegex offers significant processing speed up. We evaluate GPURegex using the “throughput” and “execution time” metrics (i.e., latency). The actual result meets and succeeds the expected one, as described in Section 4.4 and the previous section (Section 4.5.1).

Table 12: Component-related KPIs that concern GPURegex

KPI	Metric	Expected Result	Relevant Project KPI
Efficiency	Throughput and Execution time	At least 10% processing speed-up	<i>KPI-O1-E1-2</i>

5 Conclusions

In this deliverable, entitled D4.2 “Security assurance and acceleration in the E2F2C framework – initial version” we presented the work performed in the context of Task 4.3 “Security and acceleration in the complete E2F2C”, within the scope of WP4 “MARVEL E2F2C distributed ubiquitous computing framework” and the MARVEL project under Grant Agreement No. 957337.

Specifically, the main three components are presented and discussed. The three components have been developed in the context of Task 4.3 and offer security and acceleration features in the complete E2F2C2 MARVEL framework. The security-related components are EdgeSec VPN and EdgeSec TEE. EdgeSec VPN secures the communications using end-to-end network encryption, while EdgeSec TEE shields the execution of sensitive data processing applications within trusted regions of memory. The component that offers acceleration in the pattern matching procedure is GPURegex, which takes advantage of the SIMD parallel processing architectural design and modern processors, like powerful multi-core CPUs or GPUs.

This deliverable corresponds to the initial version of “Security assurance and acceleration in the E2F2C framework”. In D4.5, the final version of this deliverable, the three components will be explored and evaluated within the context of the MARVEL project as integral parts of the whole framework and the use cases defined.

6 References

- [1] Zhang, Z., Zhang, Y. Q., Chu, X., & Li, B. (2004). An overview of virtual private network (VPN): IP VPN and optical VPN. *Photonic network communications*, 7(3), 213-225.
- [2] Hauser, F., Häberle, M., Schmidt, M., & Menth, M. (2020). P4-IPsec: site-to-site and host-to-site VPN with IPsec in P4-based SDN. *IEEE Access*, 8, 139567-139586.
- [3] Alrowaily, M., & Lu, Z. (2018, October). Secure edge computing in IoT systems: review and case studies. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)* (pp. 440-444). IEEE.
- [4] A. Pfitzmann and M. Hansen, "Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management-a consolidated proposal for terminology," Version v0, vol. 31, p. 15, 2008.
- [5] J. Zhang, B. Chen, Y. Zhao, X. Cheng, and F. Hu, "Data security and privacy-preserving in edge computing paradigm: Survey and open issues," *IEEE Access*, vol. 6, pp. 18 209–18 237, 2018
- [6] Durrezi, M., Subashi, A., Durrezi, A., Barolli, L., & Uchida, K. (2019). *Secure communication architecture for internet of things using smartphones and multi-access edge computing in environment monitoring*. *Journal of Ambient Intelligence and Humanized Computing*, 10(4), 1631-1640
- [7] Hopkins, J. and Green, M. (2019). OpenVPN 2.4 Evaluation Summary and Report. [online] Private Internet Access Blog. Available at: <https://www.privateinternetaccess.com/blog/2017/05/openvpn-2-4-evaluation-summary-report/> [Accessed 16 Aug. 2019].
- [8] Cisco. (2019). Security and VPN - Support Documentation. Retrieved from <https://www.cisco.com/c/en/us/tech/security-vpn/index.html>
- [9] Ferguson, N., & Schneier, B. (2003). A Cryptographic Evaluation of IPsec. Retrieved from <https://www.schneier.com/academic/paperfiles/paper-IPSec.pdf>
- [10] Donenfeld, J. (2018). WireGuard: Next Generation Kernel Network Tunnel [Ebook] (1st ed.). Retrieved from <http://www.wireguard.com/papers/wireguard.pdf>.
- [11] Osswald, L., Häberle, M., & Menth, M. (2020). Performance Comparison of VPN Solutions
- [12] Schuster, Felix, et al. "VC3: Trustworthy data analytics in the cloud using SGX." *2015 IEEE symposium on security and privacy*. IEEE, 2015.
- [13] Zheng, Wenting, et al. "Opaque: An oblivious and encrypted distributed analytics platform." *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 2017.
- [14] Priebe, Christian, Kapil Vaswani, and Manuel Costa. "EnclaveDB: A secure database using SGX." *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018.
- [15] Goltzsche, David, et al. "Endbox: Scalable middlebox functions using client-side trusted execution." *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2018.
- [16] Trach, Bohdan, et al. "Shieldbox: Secure middleboxes using shielded execution." *Proceedings of the Symposium on SDN Research*. 2018.
- [17] Poddar, Rishabh, et al. "{SafeBricks}: Shielding Network Functions in the Cloud." *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. 2018.
- [18] Deyannis, Dimitris, et al. "Trustav: Practical and privacy preserving malware analysis in the cloud." *Proceedings of the tenth ACM conference on data and application security and privacy*. 2020.
- [19] Deyannis, Dimitris, et al. "Andromeda: Enabling Secure Enclaves for the Android Ecosystem." *International Conference on Information Security*. Springer, Cham, 2021.
- [20] Baumann, Andrew, Marcus Peinado, and Galen Hunt. "Shielding applications from an untrusted cloud with haven." *ACM Transactions on Computer Systems (TOCS)* 33.3 (2015): 1-26.
- [21] Shinde, Shweta, et al. "Panoply: Low-TCB Linux Applications With SGX Enclaves." *NDSS*. 2017.
- [22] Tian, Hongliang, et al. "Sgxkernel: A library operating system optimized for intel SGX." *Proceedings of the Computing Frontiers Conference*. 2017.
- [23] Tsai, Chia-Che, Donald E. Porter, and Mona Vij. "{Graphene-SGX}: A Practical Library {OS} for Unmodified Applications on {SGX}." *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. 2017.
- [24] Arnautov, Sergei, et al. "{SCONE}: Secure linux containers with intel {SGX}." *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 2016.
- [25] Baumann, Andrew, Marcus Peinado, and Galen Hunt. "Shielding applications from an untrusted cloud with haven." *ACM Transactions on Computer Systems (TOCS)* 33.3 (2015): 1-26.

- [26] Kunkel, Roland, et al. "Tensorscone: A secure tensorflow framework using intel sgx." *arXiv preprint arXiv:1902.04413*(2019).
- [27] Volos, Stavros, Kapil Vaswani, and Rodrigo Bruno. "Graviton: Trusted Execution Environments on {GPUs}." *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 2018.
- [28] Costan, Victor, and Srinivas Devadas. "Intel SGX explained." *Cryptology ePrint Archive* (2016).
- [29] Gueron, Shay. "A memory encryption engine suitable for general purpose processors." *Cryptology ePrint Archive*(2016).
- [30] Aho, Alfred V., and Margaret J. Corasick. "Efficient string matching: an aid to bibliographic search." *Communications of the ACM* 18.6 (1975): 333-340.
- [31] Smith, Randy, et al. "Evaluating GPUs for network packet signature matching." *2009 IEEE International Symposium on Performance Analysis of Systems and Software*. IEEE, 2009.
- [32] Vasiliadis, Giorgos, et al. "Gnort: High performance network intrusion detection using graphics processors." *International workshop on recent advances in intrusion detection*. Springer, Berlin, Heidelberg, 2008.
- [33] Vasiliadis, Giorgos, et al. "Regular expression matching on graphics hardware for intrusion detection." *International Workshop on Recent Advances in Intrusion Detection*. Springer, Berlin, Heidelberg, 2009.
- [34] Vasiliadis, Giorgos, Michalis Polychronakis, and Sotiris Ioannidis. "MIDeA: a multi-parallel intrusion detection architecture." *Proceedings of the 18th ACM conference on Computer and communications security*. 2011.
- [35] Harrison, Owen, and John Waldron. "Practical Symmetric Key Cryptography on Modern Graphics Hardware." *USENIX Security Symposium*. Vol. 195. 2008.
- [36] Han, Sangjin, et al. "PacketShader: a GPU-accelerated software router." *ACM SIGCOMM Computer Communication Review* 40.4 (2010): 195-206.
- [37] Jamshed, Muhammad Asim, et al. "Kargus: a highly-scalable software-based intrusion detection system." *Proceedings of the 2012 ACM conference on Computer and communications security*. 2012.
- [38] Choi, Byungkwon, et al. "{DFC}: Accelerating string pattern matching for network applications." *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. 2016.
- [39] Sun, Weibin, and Robert Ricci. "Fast and flexible: Parallel packet processing with GPUs and click." *Architectures for Networking and Communications Systems*. IEEE, 2013.
- [40] Vasiliadis, Giorgos, et al. "{GASPP}: A {GPU-Accelerated} Stateful Packet Processing Framework." *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. 2014.
- [41] Go, Younghwan, et al. "{APUNet}: Revitalizing {GPU} as Packet Processing Accelerator." *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 2017.
- [42] Papadogiannaki, Eva, et al. "Efficient software packet processing on heterogeneous and asymmetric hardware architectures." *IEEE/ACM Transactions on Networking* 25.3 (2017): 1593-1606.
- [43] Giakoumakis, Giannis, et al. "Pythia: Scheduling of concurrent network packet processing applications on heterogeneous devices." *2020 6th IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2020.
- [44] Kim, Joongi, et al. "NBA (network balancing act) a high-performance packet processing framework for heterogeneous processors." *Proceedings of the Tenth European Conference on Computer Systems*. 2015.
- [45] Papadogiannaki, Eva, and Sotiris Ioannidis. "Acceleration of intrusion detection in encrypted network traffic using heterogeneous hardware." *Sensors* 21.4 (2021): 1140.
- [46] Papadogiannaki, Eva, Dimitris Deyannis, and Sotiris Ioannidis. "Head (er) Hunter: fast intrusion detection using packet metadata signatures." *2020 IEEE 25th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*. IEEE, 2020.