An Artificial Intelligent Aided Unified Network for Secure Beyond 5G Long Term Evolution [GA: 101096456]

Deliverable 6.8

Italian in-lab testbed dataset 2

Programme: HORIZON-JU-SNS-2022-STREAM-A-01-06

Start Date: 01 January 2023

Duration: 36 Months

NANCY project has received funding from the Smart Networks and Services Joint Undertaking (SNS JU) under the European Union's Horizon Europe research and innovation programme under Grant Agreement No 101096456.

Document Control Page

Deliverable Name	Italian in-lab testbed dataset 2
Deliverable Number	D6.8
Work Package	WP6
Associated Task	T6.7 Italian in-lab testbed
Dissemination Level	Public
Due Date	30 November 2024 (M23)
Completion Date	29 November 2024
Submission Date	30 November 2024
Deliverable Lead Partner	ITL
Deliverable Author(s)	Antonella Clavenna (ITL), Simone Gentile (CRAT), Alvise Rigo (VOS), Daniel Casini (SSS)
Version	1.0

Document History

Version	Date	Change History	Author(s)	Organisation
0.1	04 October 2024	Initial version	Antonella Clavenna	ITL
0.2	14 November 2024	ToC, Sections assignments, Introduction and description of the scenarios	Antonella Clavenna	ITL
0.3	20 November 2024	Contributions from CRAT and VOS	Simone Gentile Alvise Rigo	CRAT VOS
0.4	22 November 2024	Contributions from SSS	Daniel Casini	SSS
0.5	22 November 2024	Datasets related to “normal conditions” uploaded on IEEE dataport (link inserted)	Antonella Clavenna	ITL
0.6	25 November 2024	Datasets related to “normal conditions” uploaded on Zenodo (link inserted)	Antonella Clavenna	ITL
0.7	25 November 2024	Latency (RTT) and Anomaly detection Data description added; data uploaded on Zenodo and IEEE	Antonella Clavenna	ITL
1.0	29 November 2024	Internal review comments integrated. Deliverable ready for quality check revision.	Antonella Clavenna	ITL

Internal Review History

Name	Organisation	Date
Marco Tambasco	TEI	27 November 2024
Mauro Marinoni	SSS	27 November 2024

Quality Manager Revision

Name	Organisation	Date
Anna Triantafyllou, Dimitrios Pliatsios	UOWM	30 November 2024

Table of Contents

List of Figures

List of Tables

List of Acronyms

Acronym	Explanation
5G	Fifth Generation
ARM	Advanced RISC Machine
B5G	Beyond Fifth Generation
BS	Base Station
CN	Core Network
CoMP	Coordinated Multi Point
CRAT	Consortium for Research in Automation and Telecommunications
DL	Down Link
HD	High Definition
HLS	HTTP Live Streaming
HTTP	Hypertext Transfer Protocol
ISA	Instruction Set Architecture
JSON	JavaScript Object Notation
MAC	Medium Access Control
MEC	Multi-access Edge Computing
NR	New Radio
PAPI	Performance Application Programming Interface
PCAP	Packet Capture
PDU	Protocol Data Unit
QoE	Quality of Experience
QoS	Quality of Service
RAN	Radio Access Network
RTMP	Real-Time Messaging Protocol
RTP	Real-time Transport Protocol
RTSP	Real-Time Streaming Protocol
SD	Standard Definition
SSS	Sant'Anna Higher School of Pisa (Scuola Superiore Sant’Anna di Pisa)
TCP	Transmission Control Protocol
UE	User Equipment
UHD	Ultra High Definition
UL	Uplink
USB	Universal Serial Bus
VNF	Virtual Network Function
VOS	Virtual Open Systems
VTU	Video streaming and Transcoding Unit
WebRTC	Web Real-Time Communication
WP	Work Package

Executive summary

The content of Deliverable D6.8 deals with the generation of a second set of data collected in the Italian in-lab testbed, namely "dataset 2", focusing on Edge segment of the network and considering the presence of some of the NANCY components, to assess their impact with respect to the baseline defined through the first collected dataset (D6.6 "Italian in-lab testbed dataset 1"), assessment that will be carried out and completed, together with the technology evaluation, in subsequent project activities.
More specifically, D6.8 aims to collect the metrics considering an Edge server based on the ARMv8 CPU architecture, and some components specifically developed by the partners for the NANCY project; these components are integrated in the Italtel environment and are part of the testbed topology setup; they are:

- the "**Anomaly detection application module**", provided by CRAT partner and focusing on detecting anomalous utilization of computing and network resources;
- the "**VOSySmonitor and vManager**", a novel virtualization technology designed by NANCY and provided by VOS to host offloaded VNFs, which can be deployed in a bare-metal fashion ensuring "application isolation";
- the "**Malicious traffic generation application**" and "**PAPI extension for ARM performance counter interaction**", provided by SSS for the technology validation in the context of the Italian testbed set-up;
- "**Italtel VTU application**", provided by ITL, can convert audio and video streams from one format to another, at multiple encodings schemes, changing resolution, bitrate, and video parameters.

The dataset contains time series, collected by transmitting video content through the Italtel VTU application. The datasets collected are related to the observation of some of the resources involved.

Specifically, the experiments for collecting the dataset are structured in three different scenarios, respectively:

I. **Scenario 1 - ARMv8-based Edge Host: Normal Condition**
II. **Scenario 2 - ARMv8-based Edge Host: Attack Condition, Separate (isolated) Partitions**
III. **Scenario 3 - ARMv8-based Edge Host: Attack Condition, Same Partition (computational resources shared)**

The datasets related to the scenarios can be found on IEEE DataPort at:

- **Permalink:** http://ieee-dataport.org/documents/nancy-sns-ju-project-italtel-italian-lab-testbed-dataset-2
- **DOI Link:** https://dx.doi.org/10.21227/v3q2-b941

1. Introduction

The Italian in-lab testbed provides a test environment for experimenting with the Edge part of the network, and to support the testing and validation of technologies and applications developed in NANCY to make the most of the capabilities that the Edge can provide (e.g., to increase the QoS of applications in terms of reduced latency, reduced bandwidth consumption, improved privacy).

Specifically, since the ability to promptly detect anomalies (e.g., abnormal or unusual patterns of behavior related to resource consumption such as CPU, memory, and bandwidth) is essential to help to detect potential security threats, attacks, or malicious activities that might compromise performance, user experience, etc., the testbed experiments the “Anomaly detection application module” (by CRAT), developed in the context of the NANCY project and focusing on detecting anomalous utilization of computing and network resources at the Edge, where the attack surface expands.

Moreover, the testbed experiments a novel virtualization technology, designed by VOS in the context of the NANCY project, that exploits the ARM Trust-zone hardware-enforced isolation to guarantee application isolation. This novel virtualization technology is based on VOSySmonitor, a multi-core software layer, which allows the co-execution of a safety-critical Real-Time Operating System (RTOS) and a noncritical General-Purpose Operating System (GPOS) on the same hardware ARMv8-A platform [1].

1.1. Purpose of the Deliverable

D6.8 “Italian in-lab testbed dataset 2” is the second deliverable of T6.7 “Italian in-lab testbed” and it documents the datasets that were generated using the Italian in-lab testbed.

The process of dataset generation is similar to the one outlined in D6.6 “Italian in-lab testbed dataset 1”. The main distinction lies in the CPU architecture of the Edge server, which is based on the ARMv8 processor. Additionally, it features several components developed by the NANCY project.

In summary, in the Italian in-lab testbed a MEC-assisted 5G network scenario with a video streaming application for generating traffic is provided. The recorded video selected for both scenarios is “Big Buck Bunny”, which can be found at: "BigBuckBunny.mp4".

The streaming was made considering:

- different bands (N3 and N78)
- different resolutions (480p, 720p, 1080p)
- different bit rate (3Mbps, 6Mbps, 10 Mbps)
- H.264 and VP8 encoding

Finally, different scenarios were set up that include both “normal conditions” and “under attack conditions”.

The testbed topology, the integrated NANCY components, and the scenarios associated with data collection are described in detail in the respective sections below.

1.2. Relation with other Deliverables

The relation of this deliverable with other deliverables is illustrated in Figure 1. Firstly, D6.8 “Italian in-lab testbed dataset 2” is related to D6.4 “In-lab testbeds definition”, from which it takes the requirements and the activities’ plan for the Italian testbed, as well as D6.6 “Italian in-lab testbed dataset 1". In more detail, D6.4 defines the network architecture, as well as the topology that was implemented, while D6.6 collects "Italian in-lab testbed dataset 1". Additionally, D6.8 "Italian in-lab testbed dataset 2" is associated with D2.1 "NANCY Requirements Analysis", D3.1 "NANCY Architecture Design", D2.3 "Network information framework" and D6.1 "B-RAN and 5G End-to-end Facilities Setup".

D6.8 receives input from D2.1 – "NANCY Requirements Analysis", D3.1 – "NANCY Architecture Design", and D6.1" B-RAN and 5G End-to-end Facilities Setup", as regards the latter, in relation to NANCY architecture refinement from deployment and integration perspective.

As a final step, the outcomes of D6.8 – "Italian in-lab testbed dataset 1" will subsequently contribute to Work Package 2 "Usage Scenario and B-RAN Modelling, Network Requirements, and Research Framework", to part of D6.9 "Outdoor Demonstration Planning, evaluation methodology and KPIs" and to part of D6.10 "NANCY Pilots' documentation and evaluation" which are part of the results of Work Package 6 "System Integration, Validation and Demonstration".

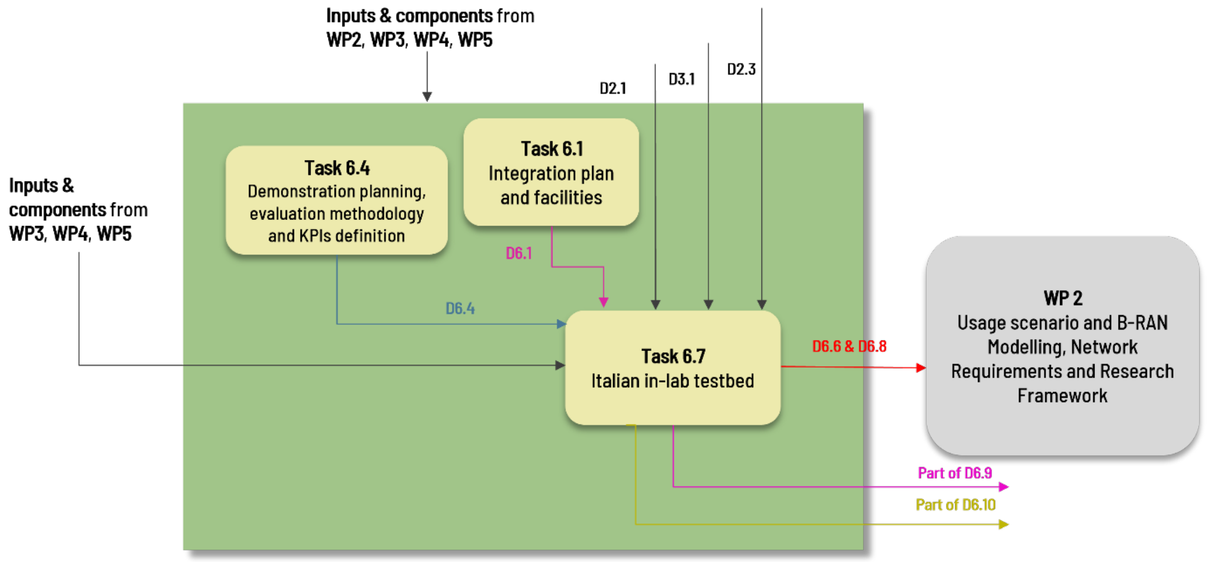

Inputs & components from WP2, WP3, WP4, WP5
D2.1
D3.1
D2.3
Inputs & components from WP3, WP4, WP5
Task 6.4 Demonstration planning, evaluation methodology and KPIs definition
Task 6.1 Integration plan and facilities
D6.1
D6.4
Task 6.7 Italian in-lab testbed
D6.6 & D6.8
WP 2 Usage scenario and B-RAN Modelling, Network Requirements and Research Framework
Part of D6.9
Part of D6.10

Figure 1: D6.8 Relations to other deliverables and WPs/Tasks

1.3. Structure of the Deliverable

The structure of D6.8 "Italian in-lab testbed dataset 2" is presented as follows:

- **Section 1** – **Introduction**: This section includes a brief introduction to this deliverable's purpose and how it is related to other deliverables. Also, this section outlines the deliverable's structure.
- **Section 2** – **Italian In-lab Testbed Description**: This section describes the Italian in-lab testbed, focusing on the topology, and the utilized hardware and software components, both generic and especially developed in the NANCY project.
- **Section 3** – **Datasets Generation Process and Structure Description**: This section includes the experimental scenarios carried out to generate the datasets. Moreover, it presents the datasets' structure and the naming convention.
- **Section 4** – **Conclusion**: This section summarizes and concludes the deliverable.

2. Italian in-lab Testbed Description

This section presents the Italian in-lab testbed for Usage Scenario 1: “Fronthaul network of fixed topology – Direct Connectivity & Coordinated Multi Point (CoMP)” for B-RAN modelling, performance measurement, and theoretical framework validation. The implementation of the testbed, in an indoor environment, located in Italtel’s premises in Milano, provides regulated and measured experimental conditions, enabling data collection and analysis. The section details the testbed design principles and its implementation. Figure 3 depicts the testbed topology, the main functional blocks and overall architecture, outlining the NANCY components that were integrated, while Figure 4 shows the physical testbed.

Considering the reference testbed scenario depicted below, the data were collected at three different points of the network, namely the UE, gNB, and edge host (i.e., point A, point B, and point E as shown in Figure 2 below.

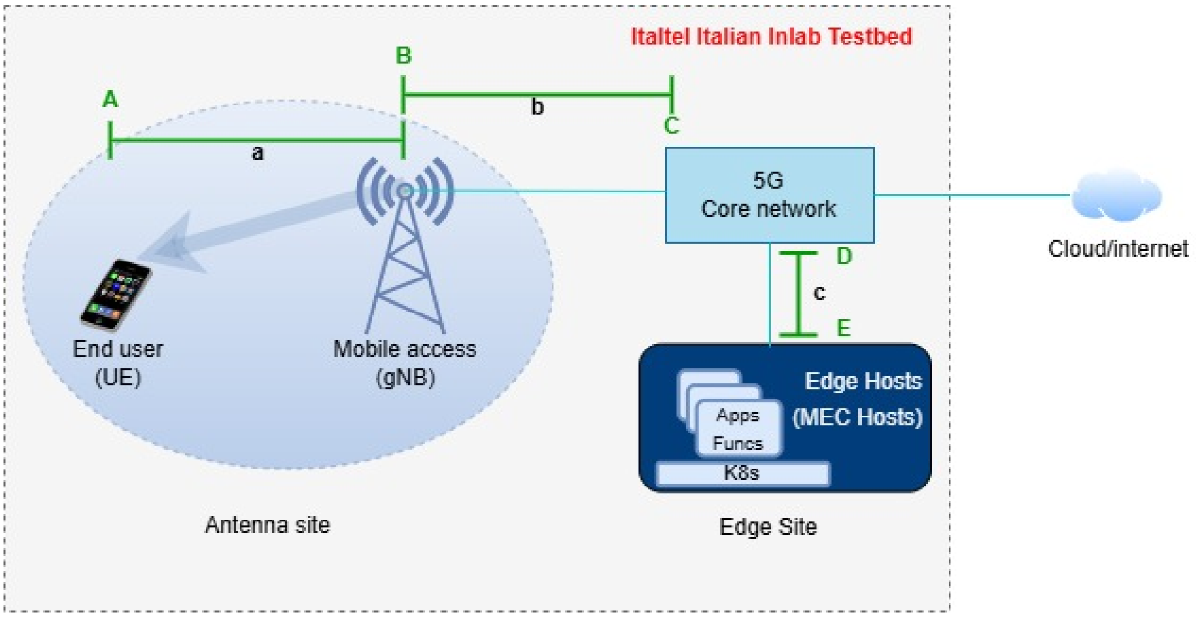


Figure 2: Italian in-lab testbed for dataset generation

2.1. Testbed Topology

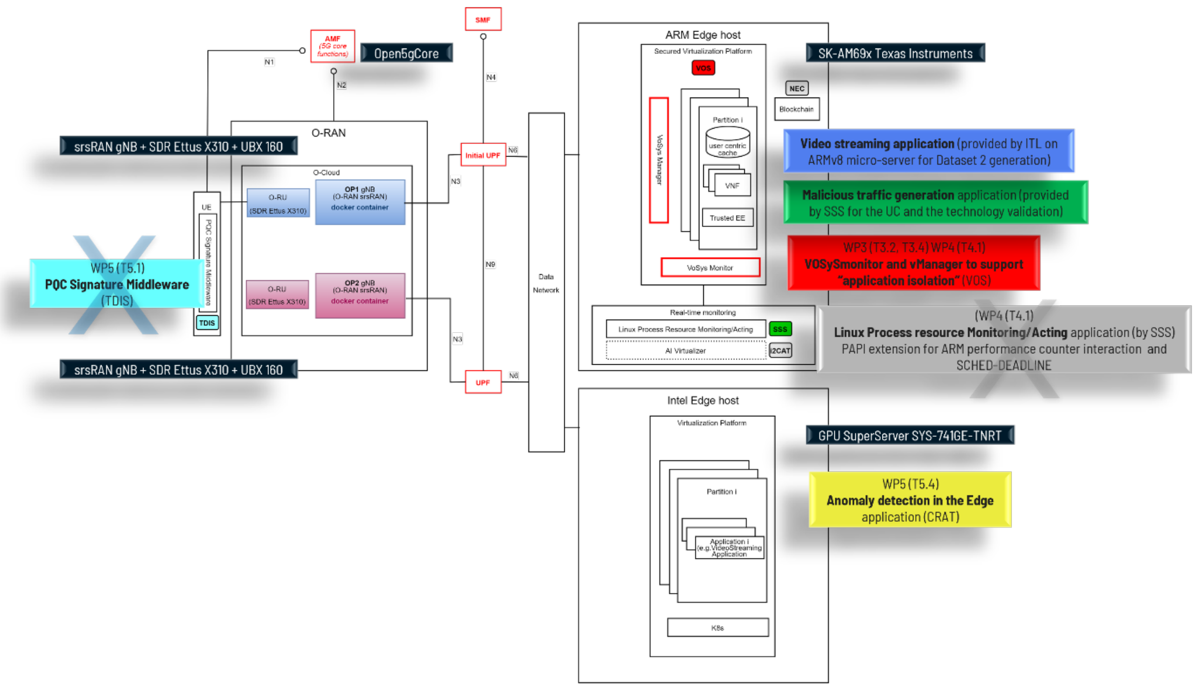

SMF
AMF (5G core functions)
Open5gCore
N1
N2
N4
srsRAN gNB + SDR Ettus X310 + UBX 160
O-RAN
O-Cloud
UE
O-RU (SDR Ettus X310)
OP1 gNB (O-RAN srsRAN) docker container
O-RU (SDR Ettus X310)
OP2 gNB (O-RAN srsRAN) docker container
PQC Signature Middleware
TDIS
WP5 (T5.1) PQC Signature Middleware (TDIS)
srsRAN gNB + SDR Ettus X310 + UBX 160
Initial UPF
N6
N3
N9
N3
UPF
N6
Data Network
ARM Edge host
Secured Virtualization Platform
VOS
VoSys Manager
Partition i
user centric cache
VNF
Trusted EE
VoSys Monitor
NEC
Blockchain
SK-AM69x Texas Instruments
Video streaming application (provided by ITL on ARMv8 micro-server for Dataset 2 generation)
Malicious traffic generation application (provided by SSS for the UC and the technology validation)
WP3 (T3.2, T3.4) WP4 (T4.1) VOSySmonitor and vManager to support "application isolation" (VOS)
Real-time monitoring
Linux Process Resource Monitoring/Acting
SSS
AI Virtualizer
i2CAT
(WP4 (T4.1) Linux Process resource Monitoring/Acting application (by SSS) PAPI extension for ARM performance counter interaction and SCHED-DEADLINE
Intel Edge host
Virtualization Platform
Partition i
Application i (e.g.VideoStreaming Application
K8s
GPU SuperServer SYS-741GE-TNRT
WP5 (T5.4) Anomaly detection in the Edge application (CRAT)

Figure 3: Italian in-lab testbed topology for dataset 2 generation scenarios

Figure 3 shows the Italian in-lab testbed topology and the integrated NANCY components. The NANCY components marked with a large "X" will be integrated later in the testbed and are, therefore, not used in the data collection scenarios considered in the present deliverable D6.8. In more detail, ITL provided the Video streaming application (Italtel VTU) integrated on the ARM Edge host provided by VOS; captured metrics are sent to the Anomaly detection application (developed by CRAT) and integrated on the Intel-based Edge host (captured metrics, i.e., the data collected from the actual scenario, are used for its training).

Based on this configuration, an initial set of data, constituting dataset 2, was collected using a scenario similar to the one set up for D6.6 "Italian in-lab testbed dataset 1", under normal conditions of operation. The distinctive feature of the current configuration is that the ITL VTU application is hosted on the ARMv8-based Edge server, in one of the two available partitions (i.e., isolated compartments) provided through "VOSySmonitor and vManager" NANCY component, developed by VOS.

Subsequently, a second set of data was collected by running, for the same scenario, a "Malicious traffic generation" application (i.e., under attack conditions). This application is provided by SSS both for the purpose of implementing the collection scenarios of the dataset 2 and for the technology validation, and it is hosted on the ARMv8-based Edge server in the other partition (i.e., the second isolated compartment available on the ARMv8-based Edge server).

Finally, for the same scenario, a third set of data was collected activating the aforementioned "Malicious traffic generation" application, hosted on the ARMv8-based Edge server, but this time integrated into the same partition as the one that hosts the ITL VTU application.

Again, captured metrics under both "attack conditions" were sent to the "Anomaly detection" application (by CRAT) on the Intel-based Edge host, to potentially identify the attack condition and, if this is the case, to send the related event to the relevant NANCY components (not part of this testbed).

2.2. Hardware Components

As depicted in Figure 4, for the purpose of collecting the second set of data "Data set 2", the Italian in-lab testbed consists of:

- **ACME server**, Multi-access Edge Computing (MEC) Host, **integrating a GPU (NVIDIA P4),** as HW accelerator to increase performance (portable): an Intel-based Edge server for hosting Open5gs (5g Core), Media server, ITL video streaming app
- a **Graphics Processing Unit (GPU) SuperServer SYS-741GE-TNRT** + 2 **GPU NVIDIA A40 (Supermicro®**): a Server for hosting 5g-srsRAN (gNB)
- 2x **Ettus Research USRP X310** + 2x **UBX160 board** (USRP X310 High-Performance Software Defined Radio – Ettus Research): 5g Radio Units
- **Google Pixel7 Android 5G**, **Motorola Moto G100**: Commercial smartphones
- **Texas Instruments SK-AM69**: ARMv8-based Edge server (by VoS) for hosting NANCY components

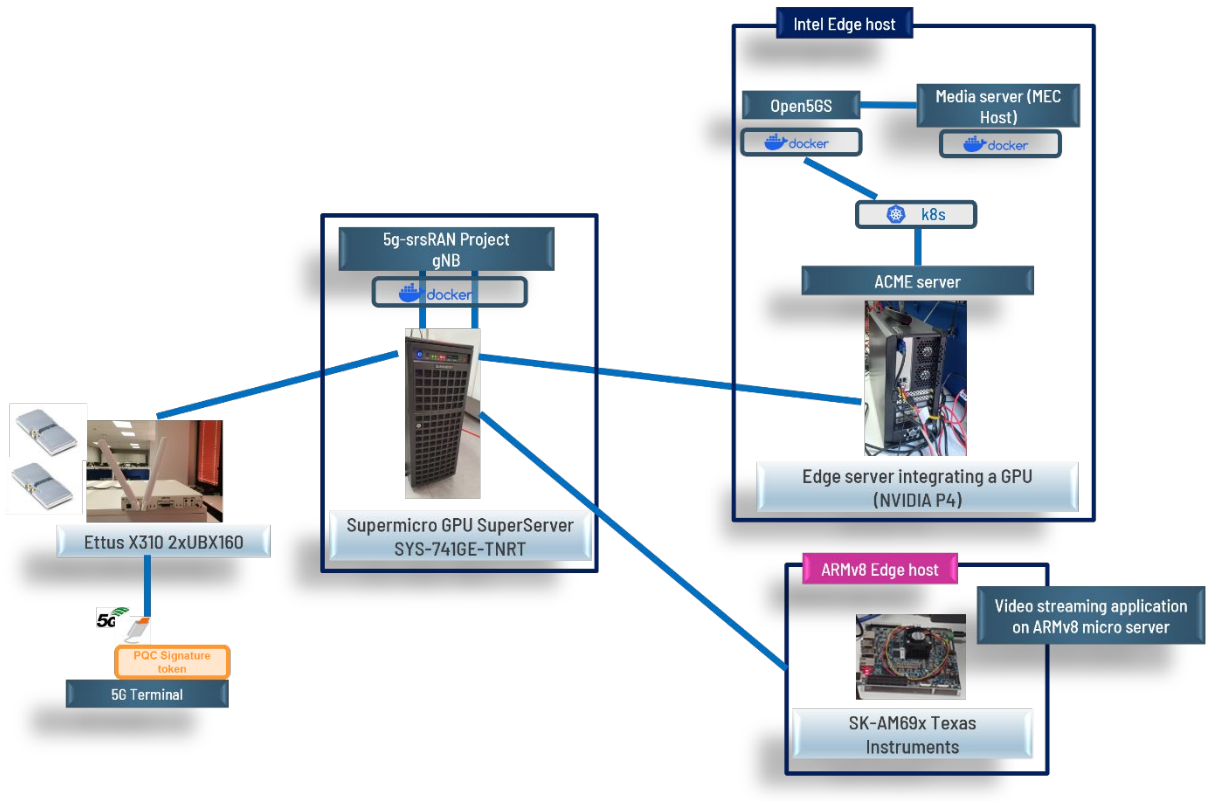


Figure 4: Italian in-lab testbed equipment

The testbed hosts a user equipment (UE), attached to a virtualized gNB instance running on a shared computing platform. The UE consists both of commercial smartphones and 5G terminals and of a radio head and a set of dedicated computing resources provided by a Supermicro® server (UE emulated). Such resources host the complete radio protocol stack and processes from heterogeneous mobile applications. Both UE and gNB use a USRP x310 board as a radio head, and the srsRAN software to implement the radio protocol stack. The gNB USRP board is attached to the computing pool via a Universal Serial Bus (USB) 3.0 connector, while the srsRAN gNB runs as containerized software instances using Docker. The UE is connected to the gNB, emulating the traffic volume generated over a cell. The testbed supports a maximum of 2 UEs and 2 gNBs.

Table 1 provides an overview of the HW devices and the respective integrated SW applications.

Table 1: HW devices and SW applications

Hardware	Description and SW applications hosted
Supermicro GPU SuperServer SYS-741GE-TNRT	Server for hosting 5G-srsRAN (gNB)
2x Ettus Research USRP X310 + 2xUBX160 board	5G Radio Units
ACME server, integrating a GPU (NVIDIA P4)	Intel-based Edge server for hosting Open5gs (5g Core), Media server, "Anomaly detection in the Edge" (by CRAT)
Google Pixel7 Android 5G, Motorola Moto G100	Commercial smartphones (UE)
Texas Instruments SK-AM69	ARMv8-based Edge server (by VOS) for hosting "VOSySmonitor and vManager" (by VOS) to support "application isolation", "Malicious traffic generation application" + "PAPI extension for ARM performance counter interaction" (by SSS), Video streaming application (by ITL)

2.3. Software applications

The following paragraphs outline the fundamental SW applications, already employed in the generation of the first round of data, D6.6 "Italian in-lab testbed dataset 1", and the other SW applications that are specifically employed in the generation of the second round of data D6.8 "Italian in-lab testbed dataset 2".

2.3.1. Fundamental SW applications

The fundamental SW applications, of general use and which are not specific NANCY components, are as follows:

1. The recorded video selected for both scenarios is "Big Buck Bunny";
2. To generate the downstream video stream from the Edge node, both Intel and ARMv8-based, the "VTU video streaming and transcoding application" developed by Italtel was selected.
3. To watch the video on the UE, the VLC media player application for the Android™ platform was selected.
4. To generate the upstream video stream from the UE, the "Simple HTTP server" Android™ app was selected, an application that will allow running a local lightweight hypertext transfer protocol (HTTP) server with static content. It is possible to select any folder, and it will become available on the local network via the HTTP protocol.

The abovementioned SW applications were also employed in the generation of D6.6 "Italian in-lab testbed dataset 1".

2.4. NANCY Software Components

The following paragraphs outline the SW applications developed in the NANCY project that are specifically employed in the generation of the second set of data (Dataset 2).

2.4.1. Anomaly Detection in the Edge

The objective of the anomaly detection module is to identify unusual patterns or deviations in the data provided that may indicate faults, security breaches, or system performance issues. By deploying this capability at the edge, the system aims to reduce latency and enable localized decision-making (Figure 5).

The anomaly detection framework employs a Random Forest (RF) classifier due to its robustness and high accuracy. The RF model is composed of multiple decision trees, each of which contributes to a collective decision, determining whether the observed pattern is in line with expected behaviour or deviates significantly. One of the key strengths of the Random Forest model is its ability to provide a level of interpretability through feature importance. In fact, it provides importance scores for each feature that helps identify the underlying causes of detected anomalies.

The implementation process of this software is carried out in several stages. Initially, the model was trained using an offline dataset, which reflects the type of data to be analysed in real-time. This dataset includes various features such as:

- CPU Usage (percentage of CPU utilization)
- Disk Space Usage (percentage of disk space used)
- Network Traffic (amount of network traffic)
- Memory Usage (percentage of memory used)
- Label (indicating whether the sample contains an anomaly or not).

Subsequently, the trained model was containerized using Docker, enabling its use for real-time anomaly detection with the data provided by the testbed.
A comprehensive explanation of the algorithm, including its evaluation metrics and detailed training methodology, will be presented in D5.3, scheduled for submission in month M29.

Dockerization of the Random Forest algorithm

The application has been containerized using Docker, allowing it to run in an isolated, portable, and scalable environment across any platform supporting it.

First, the application is configured through an external configuration file, which defines crucial parameters such as port and URL. In particular, the port on which it listens for incoming data and the URL of the dashboard where it sends the results of the Random Forest classification model. By using a configuration file, the application’s behaviour can be adjusted without modifying the container itself, simplifying deployments and updates.

To ensure the secure handling of sensitive information, the application uses environment variables to manage API authentication secrets. These secrets, such as the API username and password, are provided at runtime via environment variables, keeping them secure and out of the codebase. This method prevents the risk of exposing credentials in version control or public repositories. By passing these secrets through Docker's environment variable mechanism, the containerized application can securely authenticate with external services or APIs as needed.

The application is orchestrated using Docker Compose, which simplifies the process of starting the service with the appropriate configuration. Docker Compose allows the container to be launched with the necessary environment variables, such as the listening port, the dashboard URL, and API credentials, ensuring that the application has all the information it needs to function correctly.

Real-Time Anomaly Detection

The anomaly detection software then operates online as follows:

1. **Receiving Data**
   The system receives data in JSON format from the testbed through a REST API. This data contains information about the monitored features, such as CPU usage, RAM, disk space, and network traffic.
2. **Classification**
   Once the data is received, it undergoes pre-processing and is asynchronously fed into the Random Forest model. The RF model uses these features to determine whether the data points are normal or represent an anomaly in a binary classification.
3. **Sending Results**
   After processing, the model returns a response that includes:
   - A flag indicating the presence of an anomaly (0 or 1).
   - An optional description of the type of anomaly, if one is detected.

This response is then sent via a REST API to the Dashboard, where the result is displayed for the end user.

Libraries and Tools

The following libraries were employed during the implementation of the anomaly detection system:

- **pandas**: designed for data manipulation and analysis. It provides data structures like DataFrames and Series, which facilitate operations such as data cleaning, aggregation, and filtering.
- **numPy**: utilized for numerical operations, including array manipulations, mathematical computations, and transformations of the dataset.
- **scikit-learn**: provides various preprocessing functions such as feature normalization. It also offers the Random Forest Classifier and related tools for model training, evaluation, and optimization of hyperparameters.

These other libraries have been imported for deployment with Docker, software used for packaging the model into a container, ensuring portability and scalability across different edge nodes and environments:

- **requests**: provides a streamlined interface for sending and receiving HTTP requests, supporting methods such as GET, POST, PUT, and DELETE, along with handling headers, parameters, and payloads.
- **flask**: a lightweight and modular web framework that simplifies the development of web applications.

- **werkzeug**: provides low-level utilities for building web applications, handling HTTP requests, responses, routing, error management, and session handling, often used as the foundation for flask frameworks.
- **waitress**: is a WSGI server designed for production environments, offering robustness and scalability for Python web applications. It serves as a performant alternative to Flask's built-in development server.
- **json**: for handling JSON (JavaScript Object Notation) data. It enables the serialization of Python objects into JSON strings and the deserialization of JSON strings into Python objects.
- **logging**: for managing application logs. It provides tools to record debug messages, warnings, and errors, with support for multiple logging levels (e.g., INFO, WARNING, ERROR) and customizable output configurations.
- **threading**: for running concurrent threads within an application. It is particularly useful for implementing multitasking, improving responsiveness, and handling parallel operations in I/O-bound scenarios.
- **os**: provides tools to interact with the operating system, including file and directory management, access to environment variables, and execution of system-level commands.
- **base64**: for encoding and decoding data in Base64 format, commonly used to transmit binary data as readable text strings.

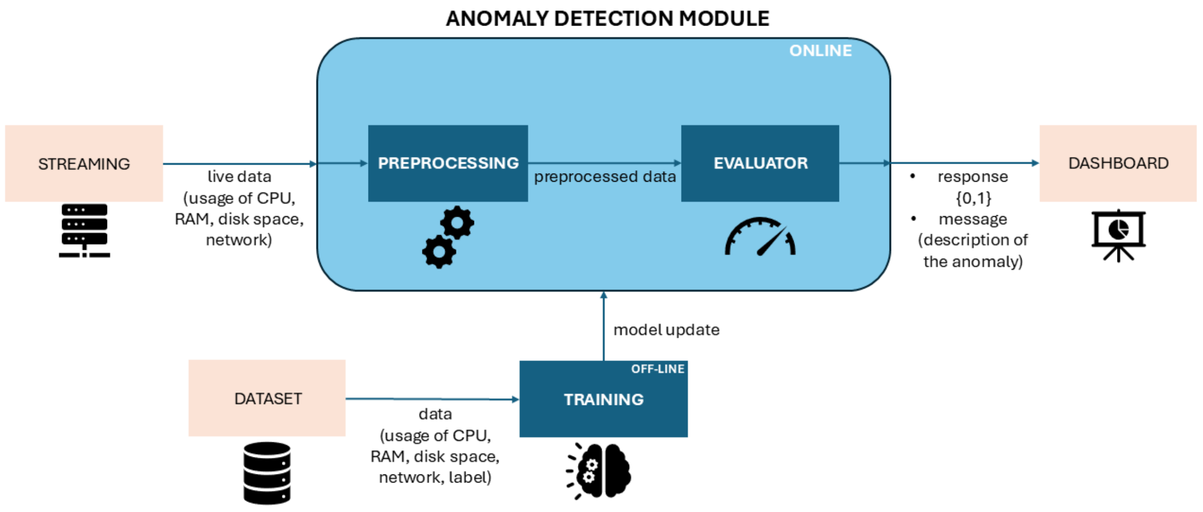

ANOMALY DETECTION MODULE
ONLINE
STREAMING
live data (usage of CPU, RAM, disk space, network)
PREPROCESSING
preprocessed data
EVALUATOR
• response {0,1}
• message (description of the anomaly)
DASHBOARD
model update
DATASET
data (usage of CPU, RAM, disk space, network, label)
OFF-LINE
TRAINING

Figure 5: Anomaly detection module

2.4.2. VoSyS Monitor and vManager

VOSySmonitor and vManager are the two software components that allow the creation of isolated execution environments, also known as partitions or compartments. Specifically, the two components serve different purposes. VOSySmonitor is a low-level software layer that can interact with the hardware to enforce the isolation between the compartments and deploy OSes into them. vManager, instead, is a higher software component which executes as a system process; it implements the logic according to which the compartments are created, destroyed and executed.

To fully understand how the interaction between these two components works, some more details must be given. VOSySmonitor, which sits at the very bottom of the software stack, is a low-level ARMv8 monitor layer that executes in the highest exception level available in the platform (called EL3). As such, it has the delicate job of configuring various peripherals (including their clocks and power domains) in such a way that the following software components (bootloaders and OSes) can use them. Being the most privileged component running in the platforms, it can fully leverage the ARMv8 functionalities (like TrustZone) to partition the system into isolated compartments, each of which is associated with a set of resources. Moreover, this monitor layer can fully control the cores available in the systems. For instance, it can power-up or power-down a core or reset it to some address in order to start executing an OS. As a matter of fact, any OS that wants to modify the power state of a core must forward a request to the monitor which will act accordingly. VOSySmonitor is not meant to perform any I/O operation for security reasons. The OS forwards the requests to the monitor via a well-defined API, called SMC Calling Convention (SMCCC), that relies on a specific ISA instruction to exchange data between the OS and the monitor. Given these limitations, there was the need to create a new software component that could interact with it through the SMCCC API. This is why vManager was created.

vManager is a system process running on top of Linux, in userspace. Given the monolithic design of Linux, vManager alone could not exhaustively interact with the monitor. For this reason, a new kernel module was developed to mediate the interaction between the two software components and validate/sanitize the data being passed. As anticipated, vManager implements the logic of the creation and destruction of compartments: in fact, it has full visibility of the resources available in the system. In addition, it can load the binaries of the OSes to be deployed into the partitions before instructing the monitor to reset a given core (or a set of cores) to a specific address to effectively execute the OS. vManager supports libvirt in such a way as to allow external orchestrators to manage the life-cycle of the compartments.

In the context of the Italian demonstrator, VOSySmonitor was initially ported to the platform, the Texas Instrument SK-AM69 Evaluation Board. The first steps were about succeeding in running one Linux instance on top of the platform, using a modified image that included VOSySmonitor in place of the default monitor. The porting experience continued with the attempts to instantiate two compartments, each of them running the Linux OS. Once this setup was stable, it was decided to keep it as a configuration for collecting the data of the second dataset. In essence, the two compartments have the following characteristics:

- Compartment 1: 31232 MiB of available RAM memory, 4 cores (corresponding to the first cluster of Cortex-A57 cores), one embedded 1Gbps ethernet port, and one UART port. In this compartment, vManager is run. The kernel used is Linux 6.6.36 along with Ubuntu 22.04.
- Compartment 2: 1536 MiB of available RAM memory, 4 cores (corresponding to the second cluster of Cortex-A57 cores), one USB type-C port with a 1Gbps ethernet adapter, and one UART port. The kernel used is Linux 6.10 along with a yocto-based minimal distribution.

For the sake of the collection of data for the second dataset, the partition creation is driven by the user and not by an external orchestrator.

2.4.3. SCHED-DEADLINE

SCHED_DEADLINE is a scheduler for time-sensitive application [2] integrated into mainline Linux, which is based on the Earliest Deadline First scheduling algorithm. It allows providing resource partitioning and resource enforcement to a vast range of computational activities, from Linux threads to virtualized applications based on KVM/QEMU virtual machines and containers [3], also thanks to a patch contributed by SSS.

SCHED_DEADLINE implements the Constant Bandwidth Reserver [4], [5] resource reservation algorithm, which provides each application with a budget Q every period P. The budget and period can be configured for the application, and they directly map to a corresponding assigned CPU bandwidth and CPU worst-case service latency [6]. Hence, the budgeting mechanism is useful for partitioning the underlying CPU's physical cores. Furthermore, SCHED_DEADLINE ensures that no more than the assigned CPU capacity is provided to the application. Mechanisms to properly set the configuration parameters Q and P have been investigated in the context of Task 3.4 and 4.2 [7].

This also offers a useful means to protect mutually untrusted applications, which often result in being colocated on the same physical platform in edge computing use cases such as those of NANCY. Indeed, a misbehaving application, which is subject to, for example, a cyber-attack or a software fault, cannot jeopardize the timing behavior of another application by executing on the CPU cores for much more than expected, because SCHED_DEADLINE ensures that it does not receive more than the assigned CPU bandwidth in any case. Therefore, SCHED_DEADLINE sets an isolation boundary to shield other applications from CPU-related timing behaviors that would otherwise easily cause denial-of-service conditions.

2.4.4. Malicious Traffic Generation Application

This section describes a malicious software that is capable of generating intense memory traffic to jeopardize the system behavior. Modern edge systems execute several applications with often different objectives and specifications at the same time. Some of these are time-critical for the functionality of the whole system, and their malfunctioning might cause issues with the quality of service provided by the system. Because the applications share the same hardware components (e.g., CPUs, memory banks), even if they do not explicitly interact with each other, interference during execution may happen. Consider a highly-time-critical application that runs on a system with a complex memory hierarchy that includes a cache memory, together with a non-time-critical application that performs a vast number of memory accesses.

While CPU-time interference is well understood, an often-neglected issue consists in memory traffic that floods the system bus with transactions and can hence disrupt the timing performance of other applications possibly running in other cores. This means that a Denial of Service (DoS) attack can be mounted.

To emulate the behavior of malicious applications that generate intensive memory traffic, we selected the IsolBench tool. IsolBench consists of a set of benchmarks whose aim is to stress the memory to compute different statistics. This application simulates a DoS attack to verify the isolation of components deployed on the same system. In the context of NANCY, we selected the Latency benchmark of IsolBench to turn it into an attack-generating application. The attack consists of generating a massive number of memory operations by repeatedly accessing the elements of a mock linked list allocated in memory. The total memory size allocated for the data structure is fixed to 131072KB.

There are two versions of the malicious application, run by two different scripts:

- `run-dos-attack.sh`: the application sequentially accesses all elements of the linked list 100 times; at the end of the test, the program prints the total duration of the test, the average memory access latency and the bandwidth;
- `run-dos-attack-endless.sh`: the application endlessly accesses the elements of the linked lists, and it must be interrupted with CTRL-C.

2.4.5. PAPI Extension for ARM Performance Counter Interaction

We developed the memory traffic monitor tool that helps detect if an application is experiencing Denial of Service (DoS) attacks by analysing its memory traffic. The tool monitors the memory accesses to level-1 and level-2 caches to fetch instructions and data and computes the number of *miss* events corresponding to those accesses.

The traffic monitor leverages the PAPI library from the University of Tennessee, Knoxville (UTK) [https://github.com/icl-utk-edu] to read the performance counters from the Cortex A72 Performance Monitoring Unit (PMU) on the TI SK-AM69 board. In detail, the PAPI low-level API manages hardware events in user-defined groups called Event Sets. It is possible to monitor both native and preset events. Native events are all those events that are countable by the CPU and whose names are platform-dependent, while presets or predefined events are a common set of events deemed relevant and useful for application performance tuning. These are mappings from symbolic names (PAPI preset names) to machine-specific definitions for a particular hardware resource.

The monitor attaches to the application to analyse and, when this terminates, it produces a .csv file with the values of the performance counters related to the application execution. Specifically, the traffic analyser reads the following six events from the PMU:

- L1-DCACHE-LOADS
- L1-DCACHE-LOAD-MISSES
- L1-ICACHE-LOADS
- L1-ICACHE-LOAD-MISSES
- L2D_CACHE_ACCESS
- L2D_CACHE_REFILL

These events count the memory accesses and misses to data and instruction level-1 cache, and level-2 cache accesses and refill events. The traffic monitor produces a .csv file for the monitored application, containing the values of the performance counters.

Monitoring experiments

We ran the memory traffic monitor to track the memory accesses of the malicious application. This experiment gave us an insight into how the number of tracked events changes as the load of the system increases, which leads to the detection of overload-based DoS attacks. We varied the memory size allocated by the malicious application and read the six performance counters listed earlier. With these, we counted the memory accesses performed by the application to fetch both instructions and data, and the corresponding number of misses in level-1 and level-2 caches.

Figure 6 reports six plots of the values of six performance counters (reported in the label above each plot) obtained running the memory traffic monitor on the malicious application. The values are shown for different memory sizes allocated by the application.

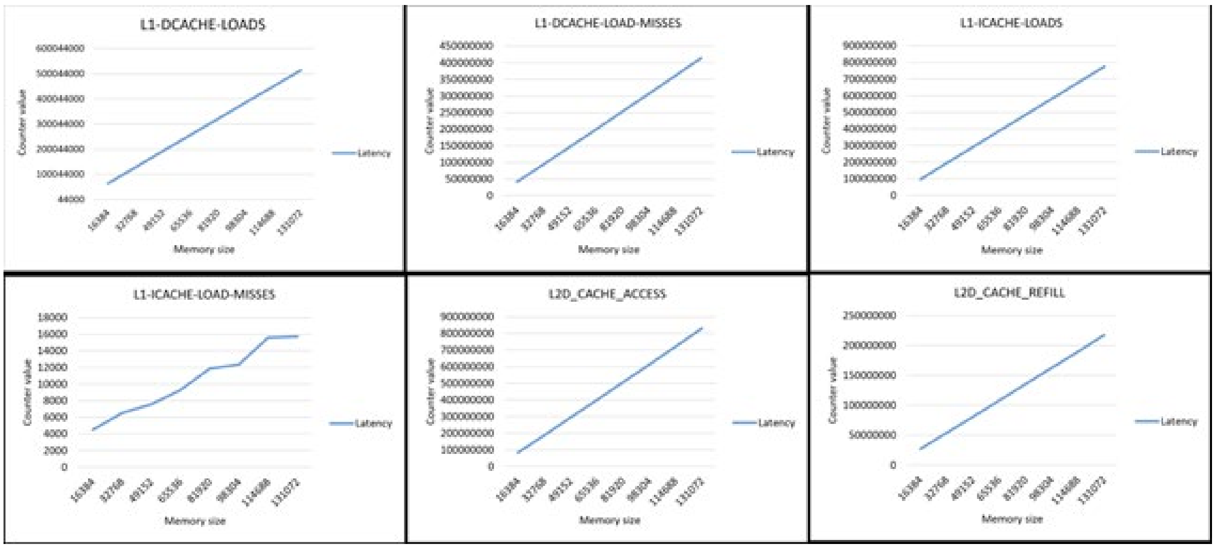


Figure 6: Performance counters

The results show that the number of events is directly proportional to the size of the allocated memory. Hence, the traffic analyser tool can be used to detect DoS attacks: when tracking an application, if the numbers read from the counters are higher than those expected in an attack-free environment, then the application is likely undergoing a DoS attack.

Two steps are required to execute the monitor:

- compile and install the PAPI library by running the script "`compile.sh`" (this needs to be done only the first time the monitor is executed);
- run the traffic analyser with the script "`start-monitor.sh`" and pass as an argument the PID of the process to monitor; example of usage: "`./start-monitor.sh 12345`" to monitor process with PID 12345.

The latter script executes the actual monitoring application, which accepts the following parameters:

```
Usage: latency [options]

        -h ...................... Print available command-line options.

        -p pid .................. Trace process with PID pid.

        -ne nevts ............... Set the number of events to trace (min 1,
max 6). Use it before -e.

        -e evt1,evt2,...,evtn .... List the events to trace, separated by
commas (without blanks)
```

3. Dataset Generation Process and Structure Description

3.1. Scenario 1 - ARMv8-based Edge Host: Normal Condition

The methodology employed to generate the collected metrics is outlined below:

1. Integration of the anomaly detection component from CRAT on the intel-based Edge host;
2. Integration of ITL video streaming application on the ARMv8-based Edge host in one of the two available partitions (i.e., isolated compartment, provided by VOS for NANCY);
3. Initial set of “dataset 2” collection, running ITL video streaming application under normal conditions - without attack;
4. The metrics gathered in the actual operational environment are being fed into the anomaly detection application for training purposes.

3.2. Scenario 2 - ARMv8-based Edge Host: Attack Condition, Separate (Isolated) Partitions

The methodology employed to generate the collected metrics is outlined below:

1. Integration of the anomaly detection component from CRAT on the intel-based Edge host.;
2. Integration of ITL video streaming application on the ARMv8-based Edge host in one of the two available partitions (i.e., isolated compartment, provided by VOS for NANCY);
3. Integration of SSS “Malicious traffic generation” application in the second available partition (i.e., ITL VTU and SSS Malicious application hosted in different isolated compartments);
4. Second set of dataset 2 collection, running ITL video streaming application in one partition and SSS Malicious application in the other partition, i.e., under attack conditions in a separate partition;
5. The metrics gathered in the actual operational environment are being fed into the anomaly detection application; **expected result:** no anomaly to be detected – i.e., the malicious application does not affect the behavior of the application running in a different (isolated) compartment.

3.3. Scenario 3 - ARMv8-based Edge Host: Attack Condition, Same Partition (Computational Resources Shared)

The methodology employed to generate the collected metrics is outlined below:

1. Integration of the anomaly detection component from CRAT on the intel-based Edge host;
2. Integration of ITL video streaming application on the ARMv8-based Edge host in one of the two available partitions (i.e., isolated compartment, provided by VOS for NANCY);
3. The “Malicious traffic generation” application (by SSS) integrated into the same partition where ITL video streaming application runs. (i.e., ITL VTU and SSS Malicious traffic generation application, hosted in the same compartment, share the related resources);
4. Third set of dataset 2 collection, running ITL video streaming application and Malicious traffic generation application in the same partition, i.e., under attack conditions sharing resources;
5. The metrics gathered in the actual operational environment are being fed into the anomaly detection application; **expected result:** anomaly to be detected– i.e., the malicious application

affects the behavior of the application since they run into the same compartment, sharing resources.

3.4. Captured Data

Each file captured is associated with a 10-minute video streaming of the "Big Buck Bunny" video. The following 5-tuple: Source IP, Destination IP, Source Port, Destination Port, and Protocol identifies a network flow. This video was transmitted with different resolutions (480p, 720p, 1080p, 2160p, and 4320p) and different bandwidth parameters (10 MHz and 20 MHz) on two different bands (N3 and N78). The HTTP protocol is monitored for both DL and UL. Also, both for N3 and N78 bands, data related to the resource usage were also captured using vmstat, for a total of more than 240 data files.

All the collected data are relevant to assessing the scenarios associated with the Italian in-lab testbed and considering the impact of NANCY architecture and components in different conditions. Figure 7 to Figure12 illustrate the testbed and associated equipment during the data collection test phases.

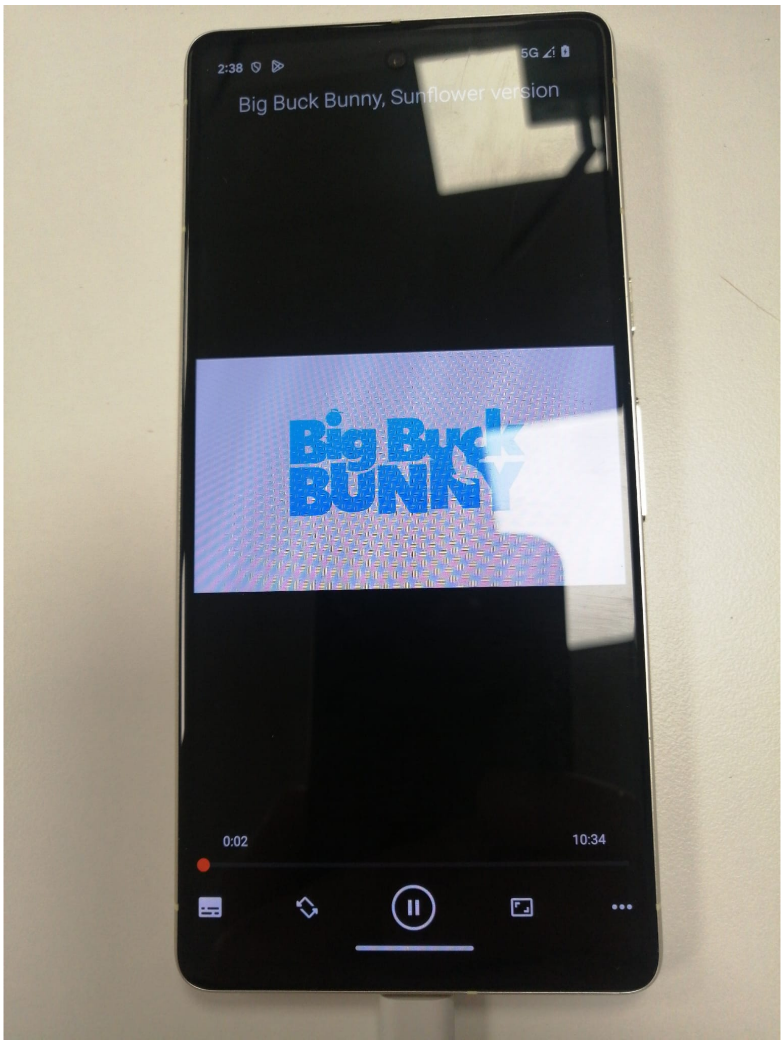
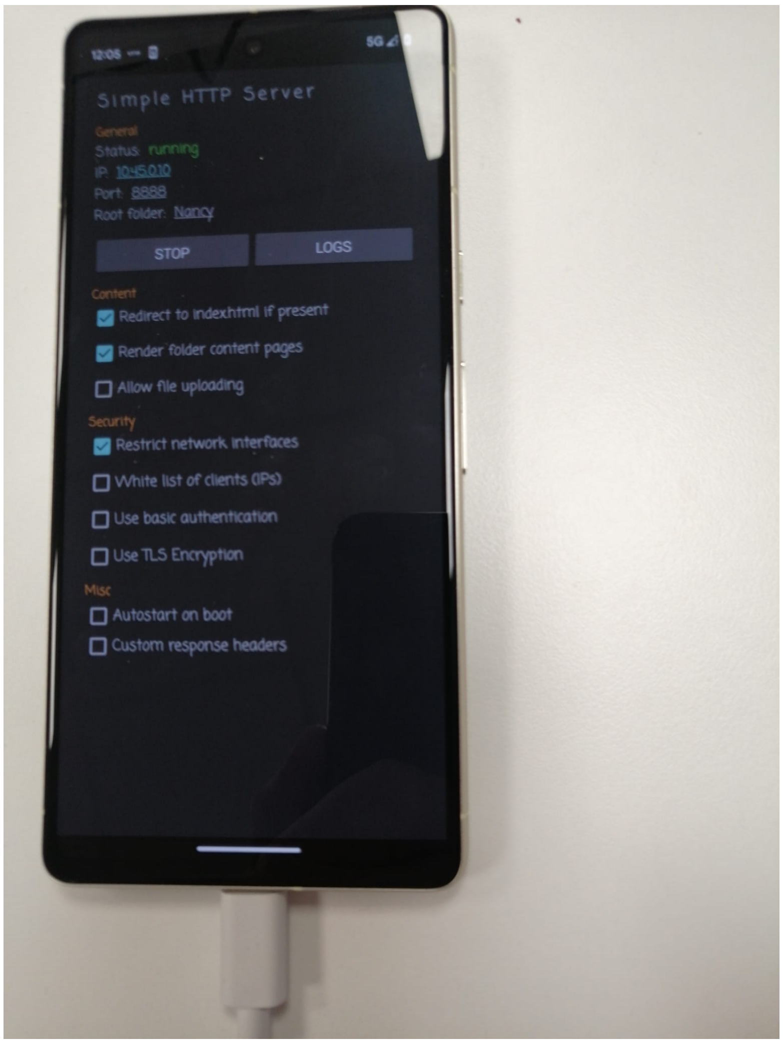


Figure 7: Pixel7 Android v13



12:05

Simple HTTP Server

General

Status: running

IP: 10.45.0.10

Port: 8888

Root folder: Nancy

Content

- [✓] Redirect to index.html if present
- [✓] Render folder content pages
- [ ] Allow file uploading

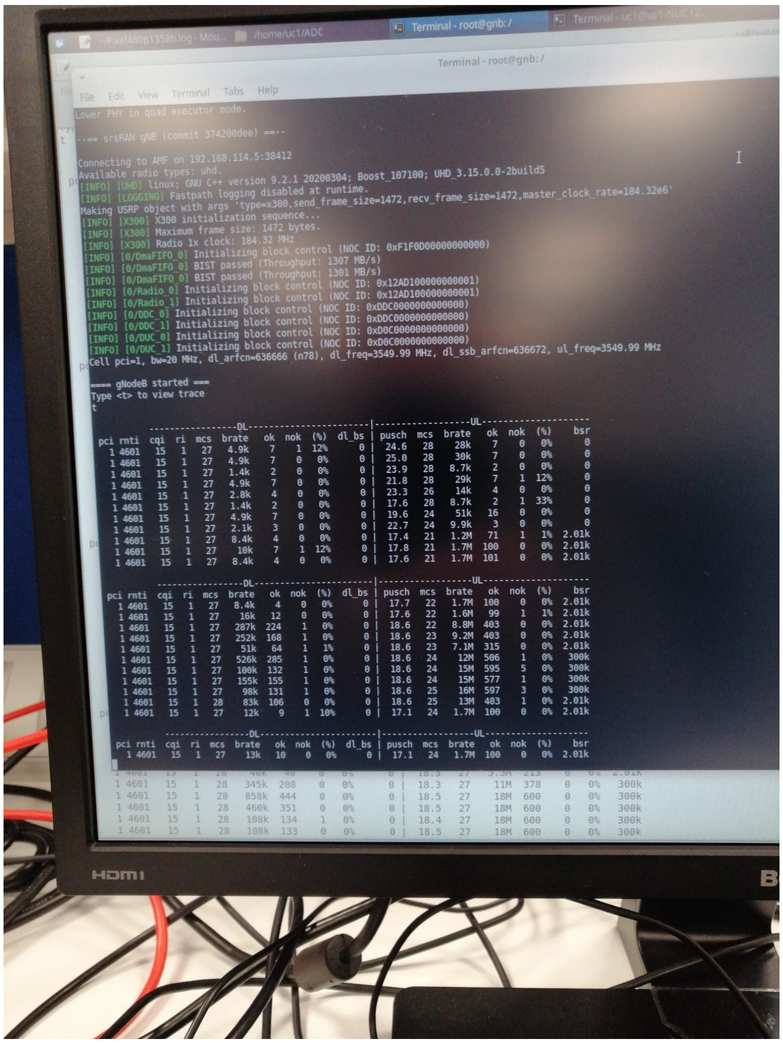
Security

- [✓] Restrict network interfaces
- [ ] White list of clients (IPs)
- [ ] Use basic authentication
- [ ] Use TLS Encryption

Misc

- [ ] Autostart on boot
- [ ] Custom response headers

Figure 8: Simple http Server on Pixel7 Android v13



```
Lower PHY in quad executor mode.

--== srsRAN gNB (commit 374200dee) ==--

Connecting to AMF on 192.168.114.5:38412
Available radio types: uhd.
[INFO] [UHD] linux; GNU C++ version 9.2.1 20200304; Boost_107100; UHD_3.15.0.0-2build5
[INFO] [LOGGING] Fastpath logging disabled at runtime.
Making USRP object with args 'type=x300,send_frame_size=1472,recv_frame_size=1472,master_clock_rate=184.32e6'
[INFO] [X300] X300 initialization sequence...
[INFO] [X300] Maximum frame size: 1472 bytes.
[INFO] [X300] Radio 1x clock: 184.32 MHz
[INFO] [0/DmaFIFO_0] Initializing block control (NOC ID: 0xF1F0D00000000000)
[INFO] [0/DmaFIFO_0] BIST passed (Throughput: 1307 MB/s)
[INFO] [0/DmaFIFO_0] BIST passed (Throughput: 1301 MB/s)
[INFO] [0/Radio_0] Initializing block control (NOC ID: 0x12AD100000000001)
[INFO] [0/Radio_1] Initializing block control (NOC ID: 0x12AD100000000001)
[INFO] [0/DDC_0] Initializing block control (NOC ID: 0xDDC0000000000000)
[INFO] [0/DDC_1] Initializing block control (NOC ID: 0xDDC0000000000000)
[INFO] [0/DUC_0] Initializing block control (NOC ID: 0xD0C0000000000000)
[INFO] [0/DUC_1] Initializing block control (NOC ID: 0xD0C0000000000000)
Cell pci=1, bw=20 MHz, dl_arfcn=636666 (n78), dl_freq=3549.99 MHz, dl_ssb_arfcn=636672, ul_freq=3549.99 MHz

==== gNodeB started ===
Type <t> to view trace
t

          ----------------DL----------------------|------------------UL--------------------
 pci rnti  cqi  ri  mcs  brate   ok  nok  (%)  dl_bs | pusch  mcs  brate   ok  nok  (%)    bsr
   1 4601   15   1   27   4.9k    7    1  12%      0 |  24.6   28    28k    7    0   0%      0
   1 4601   15   1   27   4.9k    7    0   0%      0 |  25.0   28    30k    7    0   0%      0
   1 4601   15   1   27   1.4k    2    0   0%      0 |  23.9   28   8.7k    2    0   0%      0
   1 4601   15   1   27   4.9k    7    0   0%      0 |  21.8   28    29k    7    1  12%      0
   1 4601   15   1   27   2.8k    4    0   0%      0 |  23.3   26    14k    4    0   0%      0
   1 4601   15   1   27   1.4k    2    0   0%      0 |  17.6   28   8.7k    2    1  33%      0
   1 4601   15   1   27   4.9k    7    0   0%      0 |  19.6   24    51k   16    0   0%      0
   1 4601   15   1   27   2.1k    3    0   0%      0 |  22.7   24   9.9k    3    0   0%      0
   1 4601   15   1   27   8.4k    4    0   0%      0 |  17.4   21   1.2M   71    1   1%  2.01k
   1 4601   15   1   27    10k    7    1  12%      0 |  17.8   21   1.7M  100    0   0%  2.01k
   1 4601   15   1   27   8.4k    4    0   0%      0 |  17.6   21   1.7M  101    0   0%  2.01k

          ----------------DL----------------------|------------------UL--------------------
 pci rnti  cqi  ri  mcs  brate   ok  nok  (%)  dl_bs | pusch  mcs  brate   ok  nok  (%)    bsr
   1 4601   15   1   27   8.4k    4    0   0%      0 |  17.7   22   1.7M  100    0   0%  2.01k
   1 4601   15   1   27    16k   12    0   0%      0 |  17.6   22   1.6M   99    1   1%  2.01k
   1 4601   15   1   27   287k  224    1   0%      0 |  18.6   22   8.8M  403    0   0%  2.01k
   1 4601   15   1   27   252k  168    1   0%      0 |  18.6   23   9.2M  403    0   0%  2.01k
   1 4601   15   1   27    51k   64    1   1%      0 |  18.6   23   7.1M  315    0   0%  2.01k
   1 4601   15   1   27   526k  285    1   0%      0 |  18.6   24    12M  506    1   0%   300k
   1 4601   15   1   27   100k  132    1   0%      0 |  18.6   24    15M  595    5   0%   300k
   1 4601   15   1   27   155k  155    1   0%      0 |  18.6   24    15M  577    1   0%   300k
   1 4601   15   1   27    98k  131    1   0%      0 |  18.6   25    16M  597    3   0%   300k
   1 4601   15   1   28    83k  106    0   0%      0 |  18.6   25    13M  483    1   0%  2.01k
   1 4601   15   1   27    12k    9    1  10%      0 |  17.1   24   1.7M  100    0   0%  2.01k

          ----------------DL----------------------|------------------UL--------------------
 pci rnti  cqi  ri  mcs  brate   ok  nok  (%)  dl_bs | pusch  mcs  brate   ok  nok  (%)    bsr
   1 4601   15   1   27    13k   10    0   0%      0 |  17.1   24   1.7M  100    0   0%  2.01k
```

```
   1 4601   15   1   28   345k  208    0   0%      0 |  18.3   27    11M  378    0   0%   300k
   1 4601   15   1   28   858k  444    0   0%      0 |  18.5   27    18M  600    0   0%   300k
   1 4601   15   1   28   460k  351    0   0%      0 |  18.5   27    18M  600    0   0%   300k
   1 4601   15   1   28   108k  134    1   0%      0 |  18.4   27    18M  600    0   0%   300k
   1 4601   15   1   28   108k  133    0   0%      0 |  18.5   27    18M  600    0   0%   300k
```

Figure 9: srsRAN gNB

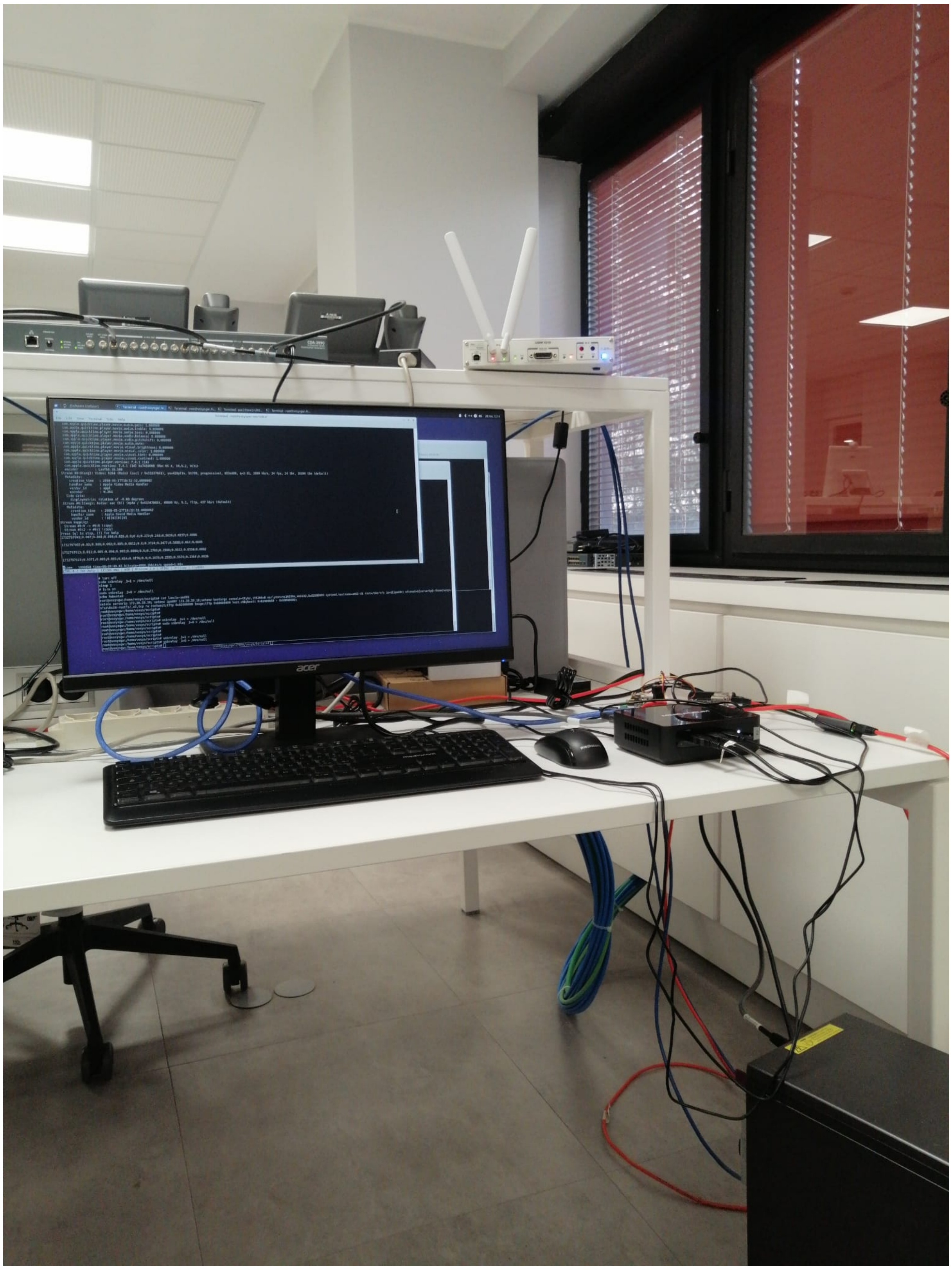


Figure 10: Italian in-lab testbed equipment during test

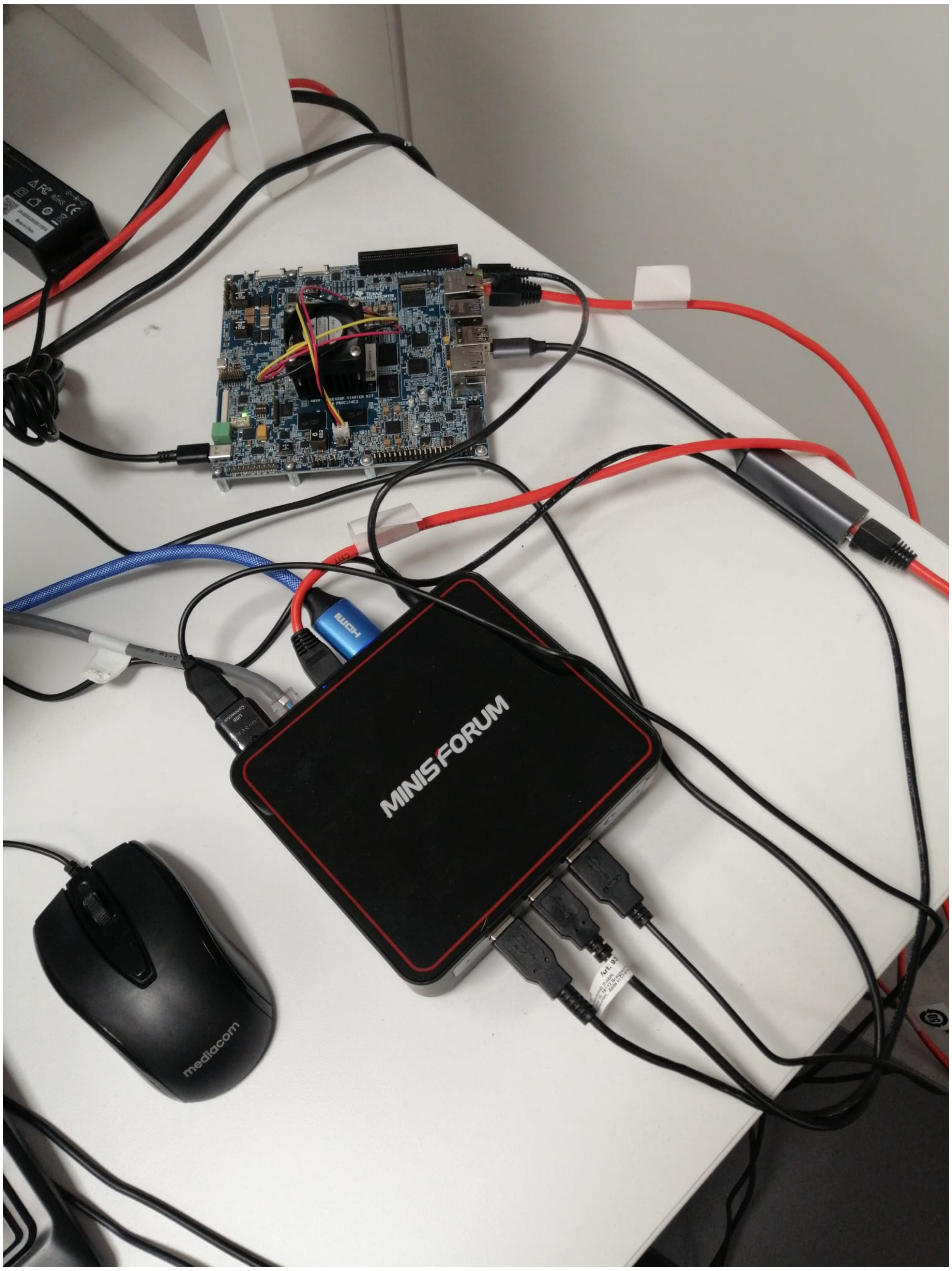
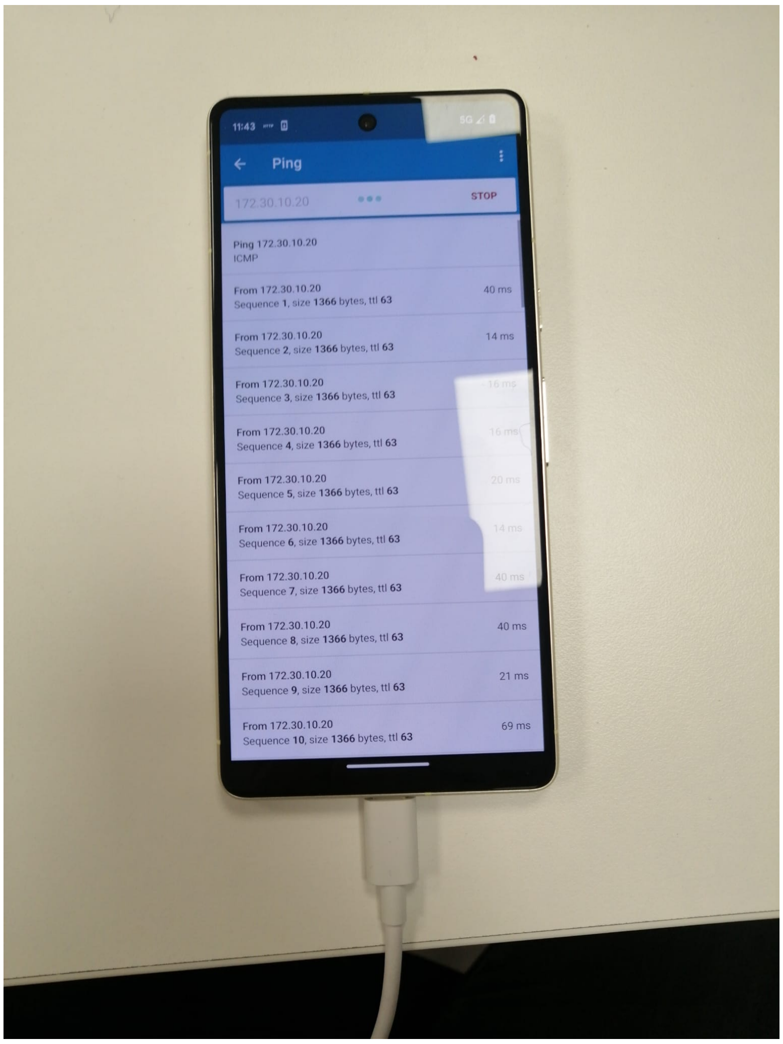


Figure 11: ARMv8 Edge host



```
Ping 172.30.10.20
ICMP
From 172.30.10.20                                    40 ms
Sequence 1, size 1366 bytes, ttl 63
From 172.30.10.20                                    14 ms
Sequence 2, size 1366 bytes, ttl 63
From 172.30.10.20                                    16 ms
Sequence 3, size 1366 bytes, ttl 63
From 172.30.10.20                                    16 ms
Sequence 4, size 1366 bytes, ttl 63
From 172.30.10.20                                    20 ms
Sequence 5, size 1366 bytes, ttl 63
From 172.30.10.20                                    14 ms
Sequence 6, size 1366 bytes, ttl 63
From 172.30.10.20                                    40 ms
Sequence 7, size 1366 bytes, ttl 63
From 172.30.10.20                                    40 ms
Sequence 8, size 1366 bytes, ttl 63
From 172.30.10.20                                    21 ms
Sequence 9, size 1366 bytes, ttl 63
From 172.30.10.20                                    69 ms
Sequence 10, size 1366 bytes, ttl 63
```

Figure 12: UE during RTT metrics collection

3.5. Dataset Folder Structure

The data collected are organized in folders. Starting from the upper level, the dataset consists of the following folders:

- The folder “Pixel7_android_v13” contains the datasets collected in the “normal conditions - without attack” scenario;
- The folder “Pixel7_android_v13_ADP” contains the datasets collected in the “Attack conditions – different partitions” where ITL VTU App and SSS Malicious App run on ARMv8 Edge in different (isolated) compartments;
- The folder “Pixel7_android_v13_ASP” contains the datasets collected in the “Attack conditions – same partition” where ITL VTU App and SSS Malicious App run on ARMv8 Edge in the same compartment and share computational resources;
- The folder “Movies_Sample” contains the “Big Buck Bunny” video, used for the test, at different resolutions.

The folder “Pixel7_android_v13” is organized into two main subfolders, “Band_N3_10Mhz” and “Band_N78_10Mhz”, respectively associated with the data captured using bands N3 and N78.

Similarly, each “Band_Nx_10Mhz” folder is organized in as many subfolders as the resolution types (i.e., 480p, 720p, 1080p, 2160p, 4320p). Each of these folders contain the “HTTP_protocol” subfolder, which is organized in two subfolders: “upload” and “download”. The “upload” folder contains all the files related to the data captured during the “upstream” tests, while the “download” folder contains all the files related to the data captured during the “downstream” tests.

Finally, each file present in these folders and named:

<capture point>_<resolution>_<video stream direction>.<type>

The naming convention is described in Table 2 to Table 6, while Figure 13 provides an example of the folder structure.

Table 2: “capture point” naming convention

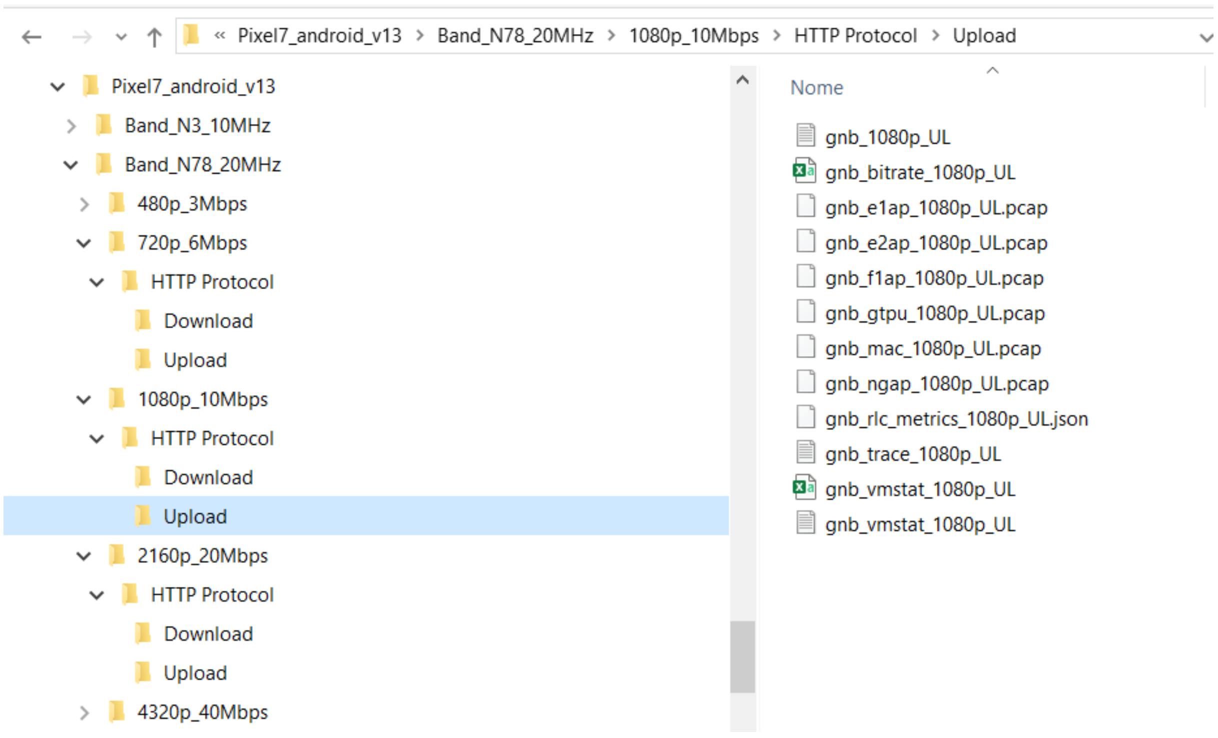
capture point	Description	Notes
gnb_trace	Contains the config and the trace of the gNB	Refer to srsRAN documentation
gnb_mac	MAC protocol 5G traffic data collected at gNB through srsRAN	
gnb_ngap	NG application protocol (NGAP) 5G traffic data collected at gNB through srsRAN	
gnb_rlc_metrics	Data collected at the gNB, related to the Radio Link Control (RLC) layer	
gnb_e2ap	E2AP 5G traffic data collected at gNB through srsRAN	
gnb_f1ap	F1AP 5G traffic data collected at gNB through srsRAN	
gnb_e1ap	E1AP 5G traffic data collected at gNB through srsRAN	
gnb_gtpu	GTP-U 5G traffic data collected at gNB through srsRAN	
gnb_bitrate	Bitrate at the gNB	
gnb_vmstat	Data collected at the gNB using vmstat and related to the resource usage	
EdgeServer	Data collected at the edge server	
EdgeClient	Data received by the edge client (generated by UE)	The Data coming out from the UE cannot be captured by PCAPdroid on the UE

Table 3: “resolution” naming convention

Resolution	Description
480p	Standard definition (SD) resolution
720p	High Definition (HD) resolution
1080p	Full HD resolution
2160p	4K UHD (Ultra High Definition)
4320p	8K UHD

Table 4: "video stream direction" naming convention

Video Stream Direction	Description
DL	Downloading direction – from Edge host to UE
UL	Uploading direction – from UE to Edge host


« Pixel7_android_v13 › Band_N78_20MHz › 1080p_10Mbps › HTTP Protocol › Upload
Nome

```
Pixel7_android_v13
  Band_N3_10MHz
  Band_N78_20MHz
    480p_3Mbps
    720p_6Mbps
      HTTP Protocol
        Download
        Upload
    1080p_10Mbps
      HTTP Protocol
        Download
        Upload
    2160p_20Mbps
      HTTP Protocol
        Download
        Upload
    4320p_40Mbps
```

```
gnb_1080p_UL
gnb_bitrate_1080p_UL
gnb_e1ap_1080p_UL.pcap
gnb_e2ap_1080p_UL.pcap
gnb_f1ap_1080p_UL.pcap
gnb_gtpu_1080p_UL.pcap
gnb_mac_1080p_UL.pcap
gnb_ngap_1080p_UL.pcap
gnb_rlc_metrics_1080p_UL.json
gnb_trace_1080p_UL
gnb_vmstat_1080p_UL
gnb_vmstat_1080p_UL
```

Figure 13: Dataset folder structure example

Table 5: “scenario” naming convention

Scenario	Description	Notes
-	Normal conditions – no attack present	ITL VTU app runs on ARMv8 Edge in an isolated compartment
ADP	Attack conditions – different partitions	ITL VTU app and SSS Malicious App run on ARMv8 Edge in different (isolated) compartments
ASP	Attack conditions – same partition	ITL VTU app and SSS Malicious App run on ARMv8 Edge in the same compartment and share computational resources

Table 6: “type” naming convention

Type of File	Description
log	Log files are a historical record of everything and anything that happens within a system.
json	JavaScript Object Notation (JSON) is an open standard file format for sharing data that uses human-readable text to store and transmit data.
csv	Comma-separated Value (CSV) file allows data to be saved in a tabular format.
pcap	PCAP - Packet capture files - are a common format for storing packet captures.

3.5.1. Latency Data (RTT) and Counters Relating to Resource Utilization Rates (CPU, Disk, and Network)

Two other folders, namely “Band_N78_20Mhz_DL RTT+ADC” and “Band_N3_10Mhz_DL RTT+ADC”, respectively associated with the data captured using bands N3 and N78, contain both RTT (Round Trip Time) and metrics related to resources utilization rates. These last ones, named “Anomaly detection counters” (ADC), are the set of metrics used by the “Anomaly Detection Application”, developed for NANCY by CRAT, to identify the presence or absence of attacks.

Steps and useful information related to RTT and ADC data

Considering the topology depicted in Figure 2, the following steps are performed:

- **Pixel7 UE** requires Video from **ITL Video Streaming APP**
- Video server streams the video in different formats (**480p**, **720p**, **1080p**, **2160p, 4320p**)
- During each video playing, in a different format, **RTT is sampled using ping every 1 sec** (Ping is transmitted every 1 second from the video server APP, along with 64bytes, 708bytes and 1358bytes payload packets).

RTT = (a+b+c)*2 (RTT is the sum of the crossing time of segments a+b+c multiplied by two, and it represents the round trip time from the upper protocol layer of the “video streaming app” to the upper protocol layer of the “video application” on the UE)

- Every 1 sec, **DLbrate** and **ULbrate** samples are captured on the gNB from “*gNB trace*” (point B in figure), while **RTT_ms** is captured on the ITL Video Streaming APP (point E in figure).
- Every 1 sec, samples of the “Anomaly detection data” are captured. These data are listed in Table 7.

Table 7: “Anomaly detection data”

Name	Type	Description
timestamp	Int	Unix Timestamp of the sample
cpu1	Float	Usage percentage of CPU 1 between 0 and 1
cpu2	Float	Usage percentage of CPU 2 between 0 and 1
cpu3	Float	Usage percentage of CPU 3 between 0 and 1
cpu4	Float	Usage percentage of CPU 4 between 0 and 1
ram	Float	Usage percentage of RAM between 0 and 1
disk_s	Float	Occupied disk space percentage between 0 and 1
disk_u	Float	Usage percentage of disk between 0 and 1
net_u	Float	Usage percentage of network uplink between 0 and 1
net_d	Float	Usage percentage of network downlink between 0 and 1

RTT File naming convention (instructions):
<UE type>_<video Format>_ping<packet size>;
values: “UE type” = Pixel7; “video Format” = 480p, 720p, 1080p, 2160p, 4320p; “packet size” = 64b, 708b, 1358b.

ADC File naming convention (instructions):
<UE type>_<video Format>_ping<packet size>_adc_<video stream direction>;
values: “UE type” = Pixel7; “video Format” = 480p, 720p, 1080p, 2160p, 4320p; “packet size” = 64b, 708b, 1358b; “video stream direction” =UL, DL.

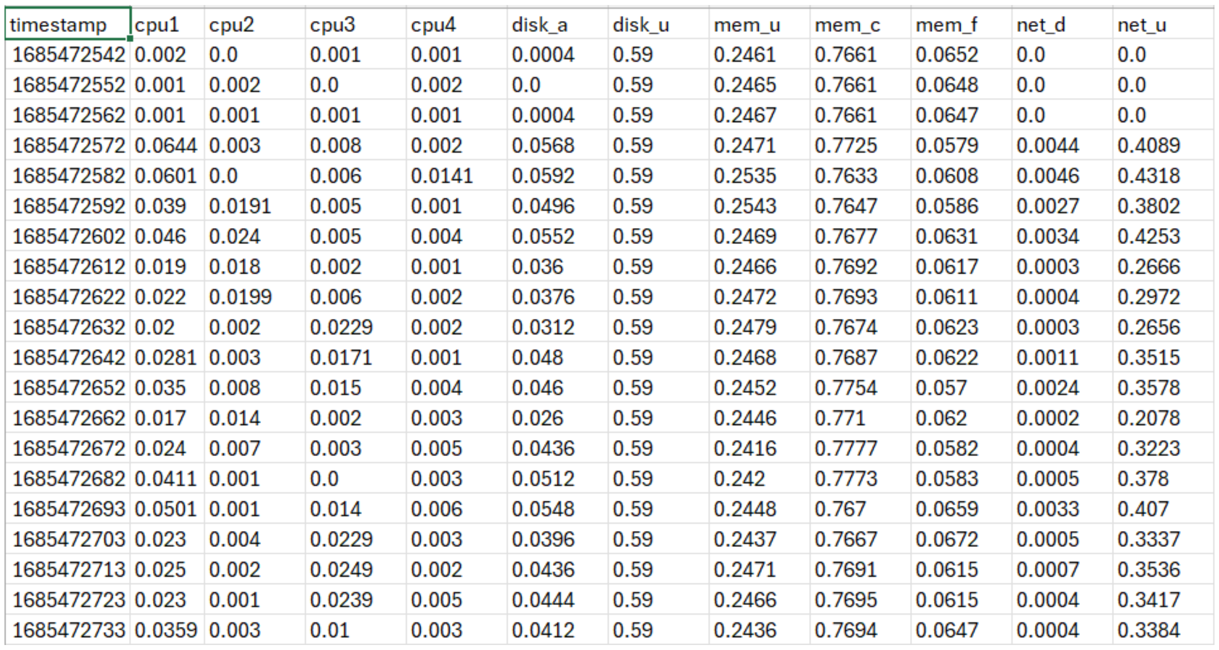
An additional set of RTT samples and ADC data has been collected for a duration equivalent to that of the video stream and under conditions of no data flow/traffic along the path. This makes it possible to sample RTT in order to collect reference values, which depend solely on the characteristics of the devices used and integrated into the testbed. An example of the ADC structure is shown in Figure 14.

RTT no video File naming convention:
Pixel7_novideo_ping<packet size>;
EdgeApp_novideo_ping<packet size>;

values: “packet size” = 64b, 708b, 1358b.

ADC no video File naming convention:
Pixel7_novideo_ping<packet size>_adc_<video stream direction>;
EdgeApp_novideo_ping<packet size>_adc_<video stream direction>;;

values: “packet size” = 64b, 708b, 1358b; “video stream direction” =UL, DL



timestamp	cpu1	cpu2	cpu3	cpu4	disk_a	disk_u	mem_u	mem_c	mem_f	net_d	net_u
1685472542	0.002	0.0	0.001	0.001	0.0004	0.59	0.2461	0.7661	0.0652	0.0	0.0
1685472552	0.001	0.002	0.0	0.002	0.0	0.59	0.2465	0.7661	0.0648	0.0	0.0
1685472562	0.001	0.001	0.001	0.001	0.0004	0.59	0.2467	0.7661	0.0647	0.0	0.0
1685472572	0.0644	0.003	0.008	0.002	0.0568	0.59	0.2471	0.7725	0.0579	0.0044	0.4089
1685472582	0.0601	0.0	0.006	0.0141	0.0592	0.59	0.2535	0.7633	0.0608	0.0046	0.4318
1685472592	0.039	0.0191	0.005	0.001	0.0496	0.59	0.2543	0.7647	0.0586	0.0027	0.3802
1685472602	0.046	0.024	0.005	0.004	0.0552	0.59	0.2469	0.7677	0.0631	0.0034	0.4253
1685472612	0.019	0.018	0.002	0.001	0.036	0.59	0.2466	0.7692	0.0617	0.0003	0.2666
1685472622	0.022	0.0199	0.006	0.002	0.0376	0.59	0.2472	0.7693	0.0611	0.0004	0.2972
1685472632	0.02	0.002	0.0229	0.002	0.0312	0.59	0.2479	0.7674	0.0623	0.0003	0.2656
1685472642	0.0281	0.003	0.0171	0.001	0.048	0.59	0.2468	0.7687	0.0622	0.0011	0.3515
1685472652	0.035	0.008	0.015	0.004	0.046	0.59	0.2452	0.7754	0.057	0.0024	0.3578
1685472662	0.017	0.014	0.002	0.003	0.026	0.59	0.2446	0.771	0.062	0.0002	0.2078
1685472672	0.024	0.007	0.003	0.005	0.0436	0.59	0.2416	0.7777	0.0582	0.0004	0.3223
1685472682	0.0411	0.001	0.0	0.003	0.0512	0.59	0.242	0.7773	0.0583	0.0005	0.378
1685472693	0.0501	0.001	0.014	0.006	0.0548	0.59	0.2448	0.767	0.0659	0.0033	0.407
1685472703	0.023	0.004	0.0229	0.003	0.0396	0.59	0.2437	0.7667	0.0672	0.0005	0.3337
1685472713	0.025	0.002	0.0249	0.002	0.0436	0.59	0.2471	0.7691	0.0615	0.0007	0.3536
1685472723	0.023	0.001	0.0239	0.005	0.0444	0.59	0.2466	0.7695	0.0615	0.0004	0.3417
1685472733	0.0359	0.003	0.01	0.003	0.0412	0.59	0.2436	0.7694	0.0647	0.0004	0.3384

Figure 14: ADC samples structure

4. Conclusion

D6.8 “Italian in-lab testbed dataset 2” documents the second set of collected data which are relevant to assessing the scenarios associated with the Italian in-lab testbed considering the impact of NANCY architecture and components.

The data were collected using the Italian in-lab testbed, which features a MEC-assisted 5G network scenario. This scenario involved a video streaming application for traffic generation and integrated the relevant NANCY applications, exploring various scenarios under both normal conditions and attacks.

Bibliography

[1] P. Lucas, K. Chappuis, M. Paolino, N. Dagieu, and D. Raho, “VOSYSmonitor, a Low Latency Monitor Layer for Mixed-Criticality Systems on ARMv8-A,” 29th Euromicro Conference on Real-Time Systems (ECRTS 2017), Leibniz International Proceedings in Informatics (LIPIcs), vol 76, pp. 6:1-6:18, Jun. 2017.

[2] J. Lelli, C. Scordino, L. Abeni, and D. Faggioli, “Deadline scheduling in the Linux kernel,” Software: Practice and Experience, vol. 46, no. 6, pp. 821–839, 2016.

[3] L. Abeni, A. Balsini, and T. Cucinotta, “Container-based real-time scheduling in the linux kernel,” ACM SIGBED Review, vol. 16, pp. 33– 38, Nov. 2019

[4] L. Abeni and G. Buttazzo, “Integrating multimedia applications in hard real-time systems,” 19th IEEE Real-Time Systems Symposium, Dec. 1998.

[5] L. Abeni, G. Lipari, and J. Lelli, “Constant bandwidth server revisited,” ACM SIGBED Review, vol. 11, no. 4, pp. 19-24, Jan. 2015.

[6] I. Shin and I. Lee, “Compositional real-time scheduling framework,” Real-Time Systems Symposium, Dec. 2004, pp. 57–67.

[7] L. Abeni, T. Cucinotta, and D. Casini, "Period Estimation for Linux-based Edge Computing Virtualization with Strong Temporal Isolation," 3rd Real-time And intelliGent Edge computing workshop (RAGE 2024), May 2024, pp. 1-6.