D5.4 SPHINX User Behaviour Simulator v1

WP5 – Analysis & Decision Making

Version: 1.00

SPHINX

A Universal Cyber Security Toolkit for Health-Care Industry

Disclaimer

Any dissemination of results reflects only the author's view and the European Commission is not responsible for any use that may be made of the information it contains.

Copyright message

© SPHINX Consortium, 2020

This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both. Reproduction is authorised provided the source is acknowledged.

Document information

Grant Agreement Number	826183	Acronym	SPHINX		
Full Title	A Universal Cyber Security Toolkit for Health-Care Industry				
Topic	SU-TDS-02-2018 Toolkit for assessing and reducing cyber risks in hospitals and care centres to protect privacy/data/infrastructures				
Funding scheme	RIA - Research and Innovation action				
Start Date	1stJanuary 2019	**Duration**	36 months		
Project URL	http://sphinx-project.eu/				
EU Project Officer	Reza RAZAVI (CNECT/H/03)				
Project Coordinator	Dimitris Askounis, National Technical University of Athens - NTUA				
Deliverable	**D5.4 SPHINX User Behaviour Simulator v1**				
Work Package	WP5 - Analysis and Decision Making				
Date of Delivery	**Contractual**	M22	**Actual**	22	
Nature	R - Report	**Dissemination Level**	P - Public		
Lead Beneficiary	NTUA				
Responsible Author	Sotiris Pelekis	**Email**	spelekis@epu.ntua.gr		
		Phone	+306981035190		
Reviewer(s):	Ilias Trochidis (ViLabs), Waqar Asif (TEC)				
Keywords	Emulation environment, Testbed, Network Traffic Replication, User Behaviour Simulation				

Document History

Version	Issue Date	Stage	Changes	Contributor
0.10	01-09-2020	Draft	ToC	Sotiris Pelekis (NTUA)
0.15	15-09-2020	Draft	Added content to section 5 provided by EDGE	Sotiris Pelekis (NTUA)
0.20	22-09-2020	Draft	Added content to section 2	Sotiris Pelekis (NTUA)
0.25	29-09-2020	Draft	Added content to section 3	Sotiris Pelekis (NTUA)
0.26	05-10-2020	Draft	First Draft of all NTUA parts	Sotiris Pelekis (NTUA)
0.30	08-10-2020	Draft	First Review of NTUA content	George Doukas (NTUA)
0.40	14-10-2020	Draft	Added SIMAVI's contribution to section 3	Sotiris Pelekis (NTUA)
0.50	18-10-2020	Draft	Internal Review 1	Ilias Trochidis (ViLabs)
0.60	19-10-2020	Draft	Internal Review 2	Waqar Asif (TEC)
0.70	28-10-2020	Pre-final	Quality Control	George Doukas (NTUA)
1.00	29-10-2020	Final	Final Version	Christos Ntanos (NTUA)

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

Executive Summary

The SPHINX Behaviour Simulation and Experimentation Environment refers to the definition of an innovative heterogeneous platform able to support a variety of cloud, IoT, SDN/NFV technologies in order to offer researchers a wide range of setups in which they can develop, debug, and evaluate their systems. The task includes the development of an emulation environment that will be used for experimentation and thorough assessment of SPHINX security architectures and their corresponding components and modules, along with the additional goal of performing a behavioural study on the applications and user habits and patterns in a repeatable manner. As a result, it is of uttermost importance to enable the design of arbitrary network topologies with scalable resources in a controllable, predictable, and repeatable environment.

The purpose of this deliverable is to present the design details and implementation status of the submodules of the Attack and Behaviour Simulator (ABS) component that correspond to “T5.4 Behaviour Simulation-Experimentation Environment Development” of SPHINX - following the component’s introduction in the SPHINX architecture deliverable (D2.6 - SPHINX Architecture v2) and the demonstration of July 24th, 2020 (during the SPHINX project review meeting) - along with the corresponding research that has been conducted within the context of this task. This research mainly focused on topology emulations, statistical realistic network traffic reproduction and user behaviour simulations.

The next iteration of this deliverable (SPHINX User Behaviour Simulator v2, M34) shall constitute a demonstrator of the Behaviour Simulator of the ABS component incorporating refinements and updates of the proposed methodologies. Eventually, it will also provide case examples for demonstrating the process and the experiments conducted through the component.

Contents

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

Table of Figures

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

Table of Abbreviations

The following table includes all abbreviations used in the document.

ABBREVIATION	EXPLANATION
IoT	Internet of Things
IP	Internet Protocol
VM	Virtual Machine
API	Application Programming Interface
SSH	Secure Shell
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UI	User Interface
GUI	Graphical User Interface
ABS	Attack and Behaviour Simulator (SPHINX Component)
IDS	Intrusion Detection System
VAE	Variational Autoencoder
GAN	Generative Adversarial Network
ML	Maximum Likelihood
DPI	Deep Packet Inspection
NFV	Network Function Virtualization
SFC	Service Function Chain
SDN	Software Defined Networking
CLI	Command Line Interface
VNF	Virtualised Network Functions

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

1 Introduction

1.1 Purpose and Scope

This document reports on the work and research conducted throughout the Task 5.4 of the SPHINX H2020 project along with the respective progress and implementation details of the corresponding submodules of the SPHINX ABS component that involve the user behaviour pattern simulation and replication utilities.

Hospital Ecosystems consist of complex network topologies implying various devices, functionalities, services, data, users, and network protocols. Pertaining to the healthcare sector and therefore involving heavily the factor of human life, it is of uttermost importance that these ecosystems are protected against cyberthreats and constantly function as desired. The SPHINX Ecosystem along with the respective tools and components, mainly described in the context of WP3 and WP4, is responsible for this proper functioning. Nevertheless, for this to happen, it is essential that all of these functions, combined with the tools and architectures that are used for their protection, are seamlessly inspected and tested for their efficiency inside safe, isolated and realistic conditions. Emulation of networks and simulation of relevant traffic constitute an ideal way for testing and validation of the SPHINX components along with the hospitals ICT infrastructures albeit without posing the real ones in danger. Such simulations are expected to function in controllable, scalable, and isolated environments-testbeds that are representative of the ones that appear in hospital topologies and of the main behaviours that characterise them.

Throughout the task 5.4 of the SPHINX project, various concepts and methodologies have been studied, implemented and tested in the context of their appropriateness for such emulations and testing purposes. These include:

- Creation of NFV testbed environments which will allow simulations and the experimentation of the user with arbitrary network topologies through virtualization technologies;
- Network traffic capturing, processing, and statistical modelling and generation methodologies;
- Realistic network traffic replication in order to resemble to the real traffic flowing in hospital networks by capturing host and device behavioural patterns inside the networks.

The technical result – software – of the abovementioned research activities is implemented in the SPHINX ABS component. The main purpose and functionality of the ABS component is to offer the SPHINX researchers a framework that renders them capable of defining arbitrary network topologies that resemble the existing architecture of the hospitals. It has been designed to offer the users a testbed environment in which she/he will be able to reproduce realistic network traffic conditions and furthermore explore the emulated system's responses against cyberattacks. This final part concerning cyberattacks directly corresponds to the linked "Task 5.3 Security Incident/Attack Simulation Environment Development" of the SPHINX project and is thoroughly described in the relevant deliverable D5.3 Security Incident/Attack Simulator v1.

1.2 Structure of the deliverable

This document is structured as follows: Section 1 presents the purpose and scope of the SPHINX Behaviour Simulation/ Experimentation environment and the relevant submodules of the respective ABS component, as well as its relation to other tasks. In Section 2, the methods and virtualisation tools that have already been tested and adopted or planned to be integrated in the ABS component in the future, with respect to the creation of the NFV testbed, are presented. In Section 3, a technical description of the methodological steps for capturing network traffic from real pilot networks is presented. Following, Section 4 analyses a methodology for modelling and reproduction of this pilot traffic based on its statistical properties, along with the relevant

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

development steps of the ABS component. Section 5 addresses the concept of realistic user behaviour simulations along with the testbed provided by EDGE which allows to simulate interactions between IoT devices and healthcare services. Section 6, concludes this document, presenting the final outcomes of the research and the component's development and future steps.

1.3 Relation to other WPs and Tasks

This document, along with task 5.4, is linked to WP3 and WP4 as the majority of the technical components presented there could reside and get tested inside the emulated environment provided by the ABS component. More specifically, it is tightly related to the tasks that partake in network inspections such as Anomaly Detection and Data Traffic Monitoring of D4.1 and D4.2 respectively, as network traffic simulations constitute a central part of it. Furthermore, D5.4 is also linked to the SPHINX Sandbox of T4.2 as this component is able to provide the required isolation properties for the safe implementation of the simulations. Finally, the work described in D5.4 – SPHINX User Behaviour Simulator sets the foundations, for the work and research of T5.3 and the corresponding deliverable D5.3 – Security Incident / Attack Simulator whose technical outcomes also constitute submodules of the ABS component.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

2 Testbed Environment for Network Function Virtualisation

There exist several testbed design strategies. An essential requirement for deploying synthetic traffic traces is to have an experimental setup and a traffic generation software. There are three commonly used testbed design strategies [1]:

- Simulation: In this method, attackers, targets and network devices are all simulated [6]. NS-2 and Opnet are two examples of network simulators that are used.
- Emulation: It is a step forward in realism over simulation. Real or virtual machines are used as hosts, attackers and targets, and the network topology is recreated in software SDN.
- Direct physical representation: In this technique, the desired network topology is built by physically arranging a network of routers, switches, and computers.

In order to reach a balancing point between realism along with feasible and safe deployment, the option of emulations has been selected for the experimentation platform of the ABS component while simulation has preferred in what regards network traffic replication. Emulations are an important tool in network research. As the selected topology often influences the outcome of the simulation, realistic topologies are needed to produce realistic simulation results [2]. Nevertheless, network topology emulations can happen in a variety of ways. Hospital networks include medical devices, various staff workstations and machines, IoT networks and relevant servers and a big number of medical lab subnets that need to be emulated. Therefore, flexible virtualization frameworks are required. The most well-known amongst them are Virtual Machines however Docker[1] constitutes an application containerization framework that can also be useful for virtualization purposes given some appropriate extensions. Both techniques are being used in the context of T5.4 depending on the needs of each implementation stage. Taking into account the complexity of different implementations based on various virtualization technologies, what is also considered in this section is the development of an effective SDN/ NFV framework that enables the creation and local execution of complex SFCs irrespective of the virtualisation technology being used by each separate module of the network.

2.1 Virtual Machines

In computing, a virtual machine (VM) is an emulation of a computer system. Virtual machines are based on computer architectures and provide functionality of a physical computer. Their implementations may involve specialized hardware, software, or a combination. Platform virtualization software, specifically emulators and hypervisors, are software packages that emulate the whole physical computer machine, often providing multiple virtual machines on one physical platform. Amongst the most well-known are Oracle VM VirtualBox[2], VMware[3] and Linux KVM[4].

Taking advantage of the wide use of VMs the first experiments with network topologies involved Oracle VirtualBox manager as a virtualisation software to fit the emulation needs of the respective subtask of T5.4. The manager offers various settings for network emulations such as Bridged, NAT, Host-Only and Internal Networking[5] which permit the deployment of various VM-based networking topologies. NAT networks have been mainly used given that they provide certain isolation from the host network and prevent interactions of the produced traffic with physical network devices. This has been achieved by defining a Nat network (10.0.0.0/24) containing interfaces of all interacting VMs as shown in Figure 1.

[1] https://www.docker.com/
[2] https://www.virtualbox.org/
[3] https://www.vmware.com/
[4] https://www.linux-kvm.org/page/Main_Page
[5] https://www.virtualbox.org/manual/ch06.html

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

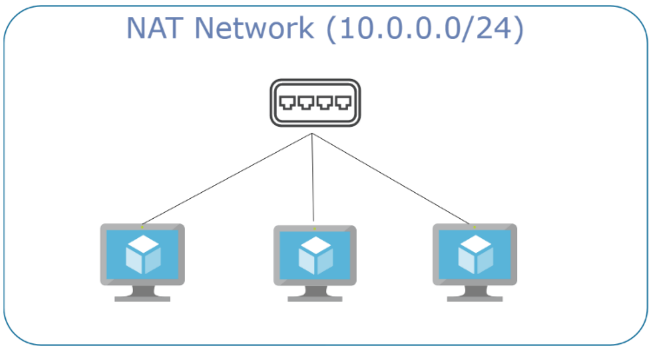
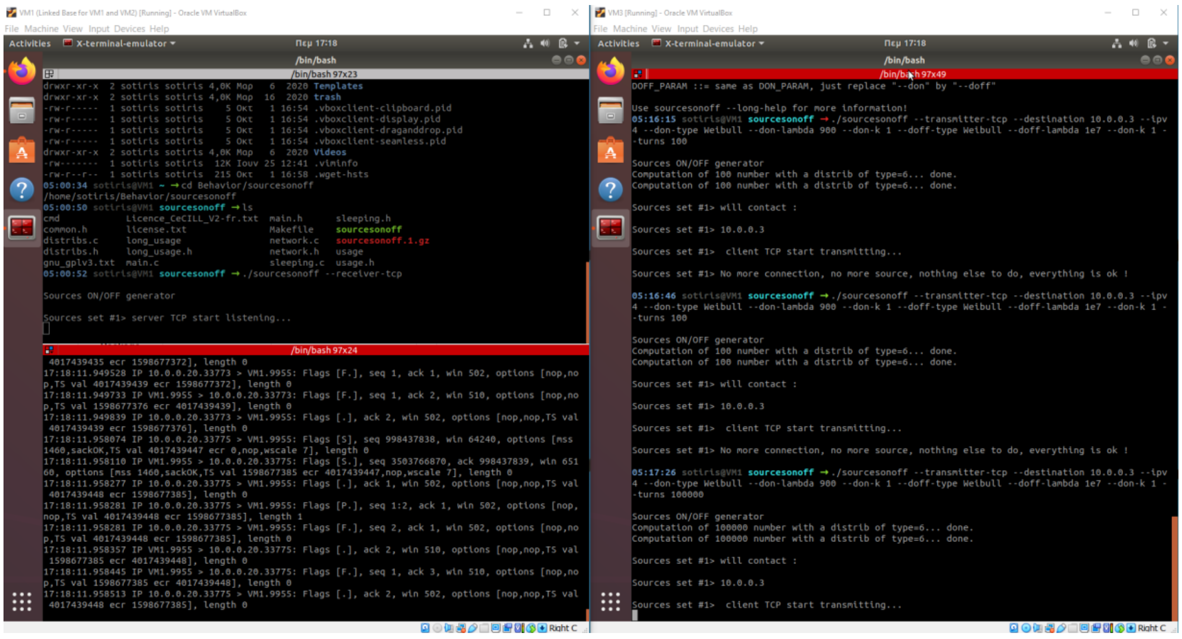


Figure 1: Sample VirtualBox VM-based NAT network

VMs, however, can take up a lot of system resources. Each VM runs not just a full copy of an operating system, but a virtual copy of all the hardware that the operating system needs to run. Additionally, hardware resources are statically allocated to each VM and therefore cannot dynamically adapt to the execution requirements. These quickly add up to a lot of RAM and CPU cycles. VMs are still economical compared to running separate actual computers, but for some applications that involve complex topologies that also require a less rigid resource allocation policy they can be overwhelming and less useful.


VM1 (Linked Base for VM1 and VM2) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities
X-terminal-emulator
Παρ 17:18
/bin/bash
/bin/bash 97x23
/bin/bash 97x24
VM3 [Running] - Oracle VM VirtualBox
/bin/bash 97x49
Right C

```
drwxr-xr-x  2 sotiris sotiris 4,0K Μαρ   6  2020 Templates
drwxr-xr-x  2 sotiris sotiris 4,0K Μαρ  16  2020 trash
-rw-r-----  1 sotiris sotiris    5 Οκτ   1 16:54 .vboxclient-clipboard.pid
-rw-r-----  1 sotiris sotiris    5 Οκτ   1 16:54 .vboxclient-display.pid
-rw-r-----  1 sotiris sotiris    5 Οκτ   1 16:54 .vboxclient-draganddrop.pid
-rw-r-----  1 sotiris sotiris    5 Οκτ   1 16:54 .vboxclient-seamless.pid
drwxr-xr-x  2 sotiris sotiris 4,0K Μαρ   6  2020 Videos
-rw-------  1 sotiris sotiris  12K Ιουν 25 12:41 .viminfo
-rw-r--r--  1 sotiris sotiris  215 Οκτ   1 16:58 .wget-hsts
05:00:34 sotiris@VM1 ~ →cd Behavior/sourcesonoff
/home/sotiris/Behavior/sourcesonoff
05:00:50 sotiris@VM1 sourcesonoff →ls
cmd          Licence_CeCILL_V2-fr.txt  main.h      sleeping.h
common.h     license.txt               Makefile    sourcesonoff
distribs.c   long_usage                network.c   sourcesonoff.1.gz
distribs.h   long_usage.h              network.h   usage
gnu_gplv3.txt  main.c                  sleeping.c  usage.h
05:00:52 sotiris@VM1 sourcesonoff →./sourcesonoff --receiver-tcp

Sources ON/OFF generator

Sources set #1> server TCP start listening...
```

```
 4017439435 ecr 1598677372], length 0
17:18:11.949528 IP 10.0.0.20.33773 > VM1.9955: Flags [F.], seq 1, ack 1, win 502, options [nop,no
p,TS val 4017439439 ecr 1598677372], length 0
17:18:11.949733 IP VM1.9955 > 10.0.0.20.33773: Flags [F.], seq 1, ack 2, win 510, options [nop,no
p,TS val 1598677376 ecr 4017439439], length 0
17:18:11.949839 IP 10.0.0.20.33773 > VM1.9955: Flags [.], ack 2, win 502, options [nop,nop,TS val
 4017439439 ecr 1598677376], length 0
17:18:11.958074 IP 10.0.0.20.33775 > VM1.9955: Flags [S], seq 998437838, win 64240, options [mss
1460,sackOK,TS val 4017439447 ecr 0,nop,wscale 7], length 0
17:18:11.958110 IP VM1.9955 > 10.0.0.20.33775: Flags [S.], seq 3503766870, ack 998437839, win 651
60, options [mss 1460,sackOK,TS val 1598677385 ecr 4017439447,nop,wscale 7], length 0
17:18:11.958277 IP 10.0.0.20.33775 > VM1.9955: Flags [.], ack 1, win 502, options [nop,nop,TS val
 4017439448 ecr 1598677385], length 0
17:18:11.958281 IP 10.0.0.20.33775 > VM1.9955: Flags [P.], seq 1:2, ack 1, win 502, options [nop,
nop,TS val 4017439448 ecr 1598677385], length 1
17:18:11.958281 IP 10.0.0.20.33775 > VM1.9955: Flags [F.], seq 2, ack 1, win 502, options [nop,no
p,TS val 4017439448 ecr 1598677385], length 0
17:18:11.958357 IP VM1.9955 > 10.0.0.20.33775: Flags [.], ack 2, win 510, options [nop,nop,TS val
 1598677385 ecr 4017439448], length 0
17:18:11.958445 IP VM1.9955 > 10.0.0.20.33775: Flags [F.], seq 1, ack 3, win 510, options [nop,no
p,TS val 1598677385 ecr 4017439448], length 0
17:18:11.958513 IP 10.0.0.20.33775 > VM1.9955: Flags [.], ack 2, win 502, options [nop,nop,TS val
 4017439448 ecr 1598677385], length 0
```

```
DOFF_PARAM ::= same as DON_PARAM, just replace "--don" by "--doff"

Use sourcesonoff --long-help for more information!
05:16:15 sotiris@VM1 sourcesonoff →./sourcesonoff --transmitter-tcp --destination 10.0.0.3 --ipv
4 --don-type Weibull --don-lambda 900 --don-k 1 --doff-type Weibull --doff-lambda 1e7 --don-k 1 -
-turns 100

Sources ON/OFF generator
Computation of 100 number with a distrib of type=6... done.
Computation of 100 number with a distrib of type=6... done.

Sources set #1> will contact :

Sources set #1> 10.0.0.3

Sources set #1>  client TCP start transmitting...

Sources set #1> No more connection, no more source, nothing else to do, everything is ok !

05:16:46 sotiris@VM1 sourcesonoff →./sourcesonoff --transmitter-tcp --destination 10.0.0.3 --ipv
4 --don-type Weibull --don-lambda 900 --don-k 1 --doff-type Weibull --doff-lambda 1e7 --don-k 1 -
-turns 100

Sources ON/OFF generator
Computation of 100 number with a distrib of type=6... done.
Computation of 100 number with a distrib of type=6... done.

Sources set #1> will contact :

Sources set #1> 10.0.0.3

Sources set #1>  client TCP start transmitting...

Sources set #1> No more connection, no more source, nothing else to do, everything is ok !

05:17:26 sotiris@VM1 sourcesonoff →./sourcesonoff --transmitter-tcp --destination 10.0.0.3 --ipv
4 --don-type Weibull --don-lambda 900 --don-k 1 --doff-type Weibull --doff-lambda 1e7 --don-k 1 -
-turns 100000

Sources ON/OFF generator
Computation of 100000 number with a distrib of type=6... done.
Computation of 100000 number with a distrib of type=6... done.

Sources set #1> will contact :

Sources set #1> 10.0.0.3

Sources set #1>  client TCP start transmitting...
```

Figure 2: Two VirtualBox Ubuntu VMs exchanging tcp flows

2.2 Docker Technology

2.2.1 Overview

Docker[6] is an open platform for developing, shipping, and running applications. Docker enables the separation of applications from infrastructure so that software can be delivered quickly. Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security allow the developer to run many containers simultaneously on a given host. Containers are lightweight given that they do not require the extra load of a hypervisor, instead they run directly within the host machine's kernel. Containers also dispose of software defined and dynamic memory, CPU and storage resources. This means that more containers can run on a given hardware combination than if VMs were being used.

[6] https://www.docker.com/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

Furthermore, Docker has a large community that shares images that can ease the deployment of new models and the reproducibility of other users' experiments.

2.2.2 Networking

In the context of networking, Docker containers can be connected together, or to non-Docker workloads as well. The Docker platform provides a variety of network drivers to enable the effective interaction amongst containers, that are:

- **User-defined bridge networks** are best when it is important that multiple containers communicate on the same Docker host. If the user does not specify a driver, the network default to the **Default Docker bridge network**.
- **Host networks** are best when the network stack should not be isolated from the Docker host, but other aspects of the container need to be isolated.
- **Overlay networks** are best when containers need to run on different Docker hosts to communicate, or when multiple applications work together using swarm services.
- **Macvlan networks** are suitable when a migration from a VM setup occurs or when containers need to resemble like physical hosts on the network, each with a unique MAC address.
- **Third-party network plugins** also facilitate integration of Docker with specialized network stacks.

In the context of T5.4 and until now, the experimentation has mainly taken place on user-defined bridge networks as they allow easy connectivity amongst containers. Furthermore, automatic DNS resolution is provided which means that containers on the same bridge network can refer to each other by their names except for IP addresses.

2.3 Architectural and emulation concepts of the ABS component

Regarding the topology creation and SDN, the current maturity of the ABS component only involves Docker containers and therefore it requires user's knowledge of docker-compose[7] and DockerFiles[8]. A docker-compose file allows the user to create networking topologies, manage behaviour and interactions amongst separate containers whose characteristics are described in a DockerFile. However, it requires advanced technical skills by the application user as she/he has to explicitly program the topology of her/his choice. The created flows of traffic can be managed either through the container terminals provided by the Docker platform or through the Flask-based[9] front end Web GUI which is presented in the section 4 of this deliverable.

A sample Docker topology, which has been implemented and has been used for experimentation purposes is demonstrated in Figure 3 along with the respective docker-compose script which can be found in Annex A. The interacting containers (server, client, attacker), which emulate real network nodes, are able to easily initiate connections, exchange packets and data flows and therefore reproduce real network behaviour, if programmed appropriately. Focusing more in the content of Figure 3, it consists of a network setting where two containers act as TCP or UDP client and server, exchanging normal network traffic while an attacking container, in this case based on the Kali distribution of Linux is performing network attacks (e.g. DoS attack) which are thoroughly described in the context of D5.3.

[7] https://docs.docker.com/compose/
[8] https://docs.docker.com/engine/reference/builder/
[9] https://flask.palletsprojects.com/en/1.1.x/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

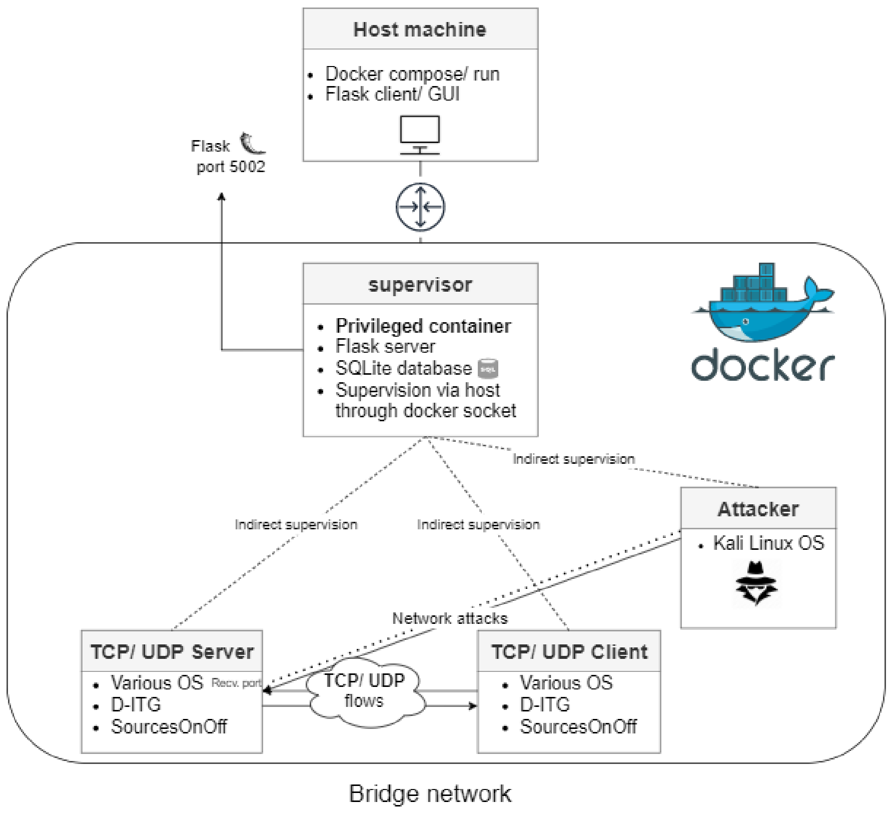


Figure 3: Example of Docker-based network topology. The respective docker-compose file can be found in Annex A.

During this process, containers are being "supervised" by the main container called "super" (which is also responsible for exposing the GUI to the host through port 5002). In this way, the specific container instructs the rest of containers to act according to the user's preference. In order to achieve that, this container is functioning in privileged mode, exposing its docker socket to the host machine. Of course, such practices of supervision could raise security issues, however in Section 2.5 an efficient manner for bypassing them in a totally secure way is described, by taking advantage of the synergy between the ABS and the SPHINX Sandbox component.

2.4 Overview of NFV/ SDN technologies

Docker provides the opportunity to build much more complex network topologies in order to scale up the size of emulations. An indicative example is depicted in Figure 4[10]. However, building such complex emulations is not an easy task, which is eminent from the respective docker-compose file provided in Annex A. Additionally, as mentioned above, a feature that would bring high added value to the functioning of the ABS component is the provision of a flexible NFV/SDN framework. This provision allows different containerisation and virtualisation technologies to be combined (such as Docker containers, VirtualBox VMs, KVM machines) along with common network components such as switches, routers and communication lines. Therefore, a variety of open-source tools have been examined throughout the implementation of T5.4 in an effort to narrow down the alternatives to the most appropriate ones. The final selection is adopted in a way that, further development can be integrated in a modular way.

10 https://medium.com/tenable-techblog/simulating-enterprise-networks-in-development-using-the-docker-networking-stack-94bf547743c9

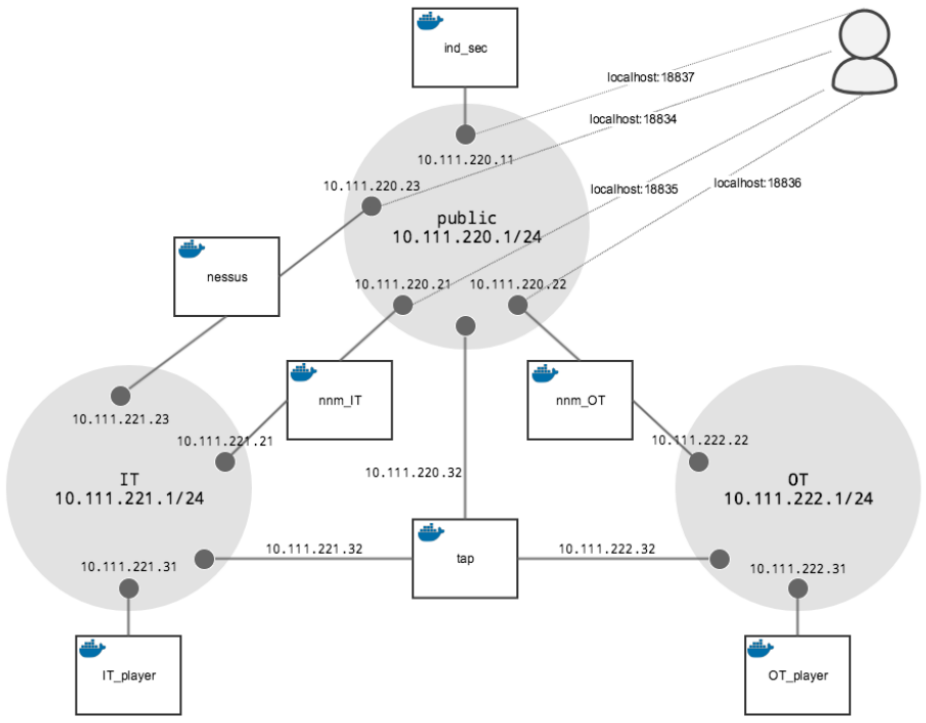


Figure 4: Complex Docker-based network topology. The respective docker-compose file can be found in Annex A.

2.4.1 vSDNemul

vSDNEmul [3] is a container-based emulator that allows the construction of complex networks with realism and scalability. The base of the structure of vSDNEmul is the use of Docker containers. vSDNEmul provides ease in defining and modelling what resources each image will use, such as applications or tools. Those configurations are managed through DockerFiles. Thus, each node in the emulated network executes in an independent and isolated fashion, increasing the realism of the emulation and affording behaviour similar to that in production infrastructures. In addition, Docker enables containers with resource constraints. Each container is executed with memory, CPU, storage or network limitations. With this feature, nodes with resource requirements can be setup during the emulation, which allows the emulation of several types of behaviours in an experiment. Finally, vSDNemul guarantees performance fidelity given that it provides isolation of resources (e.g., Memory, CPU or I/O) required by not only for hosts but for switches and controllers as well.

2.4.1.1 Architecture and API

vSDNEmul is formed by a three-layer architecture that was designed to configure network device and computer machine characteristics. Figure 5 illustrates the architecture and respective layers: User, API and Infrastructure. The User layer is the block that interacts with the user to build virtual networks. The user can set up her/his virtual network through CLI commands or Python scripts. The API layer comprises the libraries needed to perform resource allocation on the host computer, such as virtual links and machines. In addition, this layer contains the vSDNEmul Python API that abstracts low level elements on the emulator (e.g., a new node based on a Docker image or new links).

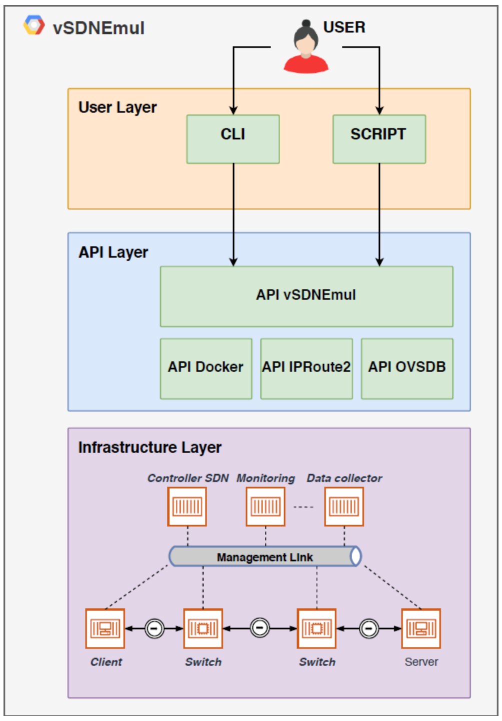
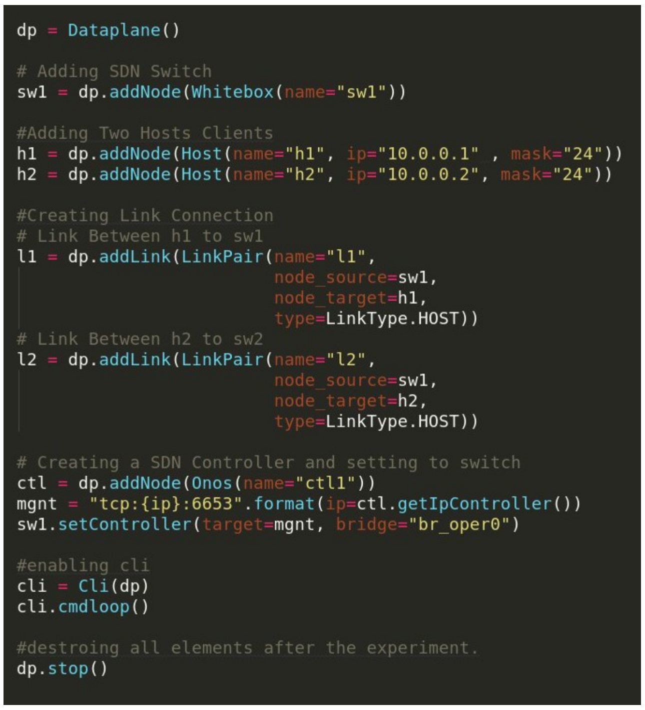


Figure 5: vSDNEmul architecture11

Following, Figure 6 illustrates a script that sets up a small topology with one controller, two hosts, one switch and two links that connect the hosts to the switch. Essentially, the script includes a data plane environment to which the nodes are added (e.g., h1, h2, ctl and sw1). Subsequently, the link on the data plane is built to connect the hosts to the switch (e.g., h1 ¨ sw1, h2 ¨ sw2). Next, the script associates the SDN switch to the controller that dictates forwarding operations in the switch and permits the connection between h1 to h2. The last lines start the CLI and the experiment. Upon executing these operations, all elements and resources are deallocated.



```
dp = Dataplane()

# Adding SDN Switch
sw1 = dp.addNode(Whitebox(name="sw1"))

#Adding Two Hosts Clients
h1 = dp.addNode(Host(name="h1", ip="10.0.0.1" , mask="24"))
h2 = dp.addNode(Host(name="h2", ip="10.0.0.2", mask="24"))

#Creating Link Connection
# Link Between h1 to sw1
l1 = dp.addLink(LinkPair(name="l1",
                         node_source=sw1,
                         node_target=h1,
                         type=LinkType.HOST))
# Link Between h2 to sw2
l2 = dp.addLink(LinkPair(name="l2",
                         node_source=sw1,
                         node_target=h2,
                         type=LinkType.HOST))

# Creating a SDN Controller and setting to switch
ctl = dp.addNode(Onos(name="ctl1"))
mgnt = "tcp:{ip}:6653".format(ip=ctl.getIpController())
sw1.setController(target=mgnt, bridge="br_oper0")

#enabling cli
cli = Cli(dp)
cli.cmdloop()

#destroing all elements after the experiment.
dp.stop()
```

Figure 6: Basic example script of topology on vSDNEmul.

[11] Image source: [3]

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

2.4.1.2 Drawbacks

Proceeding to some restrictions of the VSDNemul framework, the emulations can only involve Docker containers and do not support other virtualisation technologies which renders them less flexible in terms of hosting implementations based on various virtualisation technologies. Additionally, it does not provide a GUI for network topology prototyping, thus the user should be able to interact through a python API or the CLI.

2.4.2 ContainerNet

Lantz et al. [4] introduced Mininet, a network emulation software that enables the quick launch of a prototype virtual network with SDN switches. This lightweight virtualization uses Linux-container virtualization (namespace) features that comprise processes and network namespaces for analysing and developing SDN solutions. Containernet[12] is a fork of the Mininet network emulator that allows the use of Docker containers as hosts in emulated network topologies. This enables useful functionalities to build networking/cloud emulators and testbeds. Besides this, Containernet is actively used by the research community, focussing on experiments in the field of cloud computing, fog computing, NFV, and multi-access edge computing (MEC). Containernet 2.0 [5] extends the existing project by adding execution support for VM-based VNFs to the platform. This feature renders ContainerNet a much flexible option which permits hybrid NFV deployments.

2.4.2.1 Architecture and APIs

As a first step, the developer defines the virtual network by using either Containernet's GUI editor or a script that calls Containernet's Python API (1). After this, Containernet deploys and interconnects the involved VNFs in its local, Mininet-based emulation environment (2). Once all VNFs are running, the developer uses Containernet's interactive CLI to interact with and configure the running VNFs that can either be Docker containers or full-featured VMs (3). To establish the network between the VNFs, the underlying Mininet is used. This architecture, along with the way that a network service developer uses it, is illustrated in Figure 7. At this point, it is important to mention that in order for ContainerNet to be able to automatically manage and configure the networking settings of the containers and virtual machines in the virtual network, images with preinstalled networking tools, such as ping and ifconfig, have to be used.

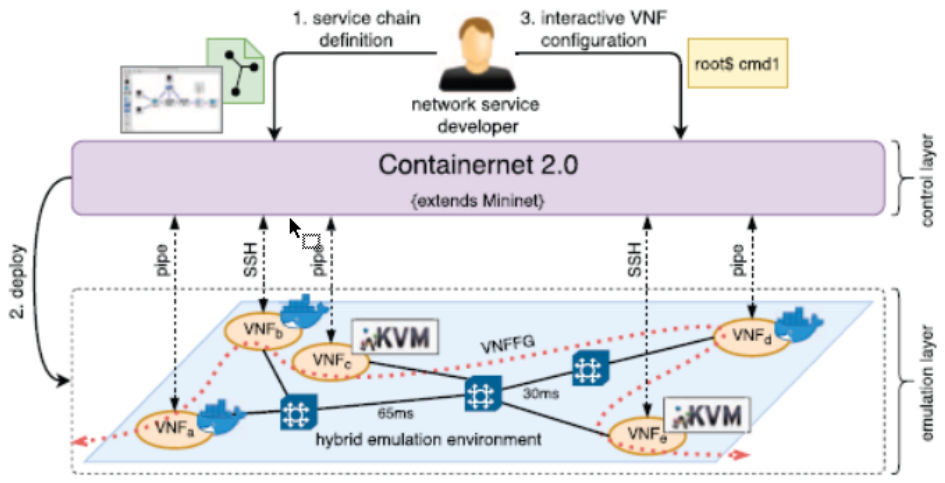


Figure 7: Demonstration overview showing a network service developer prototyping a hybrid virtual topology consisting of three container-based and two VM-based VNFs, all running in an locally emulated network.[13]

[12] https://containernet.github.io/

[13] Image source: [5]

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

2.4.2.2 Drawbacks

Proceeding, to some disadvantages of the ContainerNet emulator the following need to be highlighted:

- It only provides ethernet links which means that wireless connections should be emulated alternatively.
- Does not provide resource isolation for Mininet components (e.g controllers, switches). They all run over Mininet on the host machine. This reduces the scalability of the tool, and it is a fact that will be seriously considered throughout the second phase of the implementation.
- It can only be hosted on Ubuntu 18.04 hosts, however there is also a docker image provided on Docker Hub[14] to solve this issue. However, container resource limitations, e.g., CPU share limits, are not supported in the nested container deployment. Bare-metal installations on Ubuntu LTS 18.04 should be used shall the user needs those features.

2.4.3 Comparison results

The above presented technologies (vSDNemul, Containernet) have been thoroughly studied throughout T5.4 for supporting the task of NFV. Given that the two main disadvantages of vSDNemul seem to be critical shortcomings against the requirements of the SPHINX ABS component, the next steps of the emulator are planned to focus on the adoption, further-development and integration, in a modular way, of the ContainerNet emulator. However, the efficacy and usability of ContainerNet will be thoroughly examined and reassessed in terms of scalability in the latter phases of the implementation and the requirements posed by future experiments.

2.5 Interaction with the SPHINX Sandbox – Isolation

As described in D2.6 - SPHINX Architecture v2, The SPHINX Sandbox component is a multipurpose component. One of its two main functionalities is to **provide a safe environment where components could be deployed without compromising any of the other services.** In this mode, the Sandbox is a security mechanism for separating running programs, usually in an effort to mitigate compromised medical devices or software vulnerabilities from spreading. The proposed shared environment can also be used to execute untested or untrusted programs or code, from unverified or untrusted third parties, suppliers, users or websites, without risking any harm to the host machine or operating system. SB provides a tightly controlled set of resources for guest programs to run in the IT infrastructure.

Therefore, the SPHINX Sandbox is capable of hosting the SPHINX ABS component emulations in order to ensure the secure experimentation and testing of the corresponding SPHINX components.

[14] https://hub.docker.com/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

3 Traffic Capturing

In order to simulate network traffic in virtual networks and analyse user behaviour it is of uttermost importance in the initial stage to capture real traffic produced inside the real network infrastructures of the pilot hospitals. This section provides an initial outline of a sample of the pilot network infrastructure architecture along with some technical methodological steps in order to conduct a network traffic capturing process given the described infrastructure. It should be mentioned here that the procedures described below shall be applied in practice only in the case that the pilots ensure anonymisation and privacy of data as mentioned in the relevant deliverables concerning the legal aspects of SPHINX.

3.1 Sample Pilot Infrastructure

The purpose of this section is to provide a sample pilot infrastructure, as a use case for the capturing and analysis methodologies that are provided in the following sections. The architecture presented in Figure 8 is inspired by Dype5's real hospital network infrastructure as the NTUA team visited their premises in the City of Larissa, during February 2020. The team inspected some of the basic components of the network there while they also interacted with the technical staff of the hospital in order to obtain a first view of the architecture of hospital networking infrastructure. The Syzefxis node refers to the National Network of Public Administration which offers to Greek public sector entities efficient interconnection to the Internet, firewall services and lastly access to high quality web services.

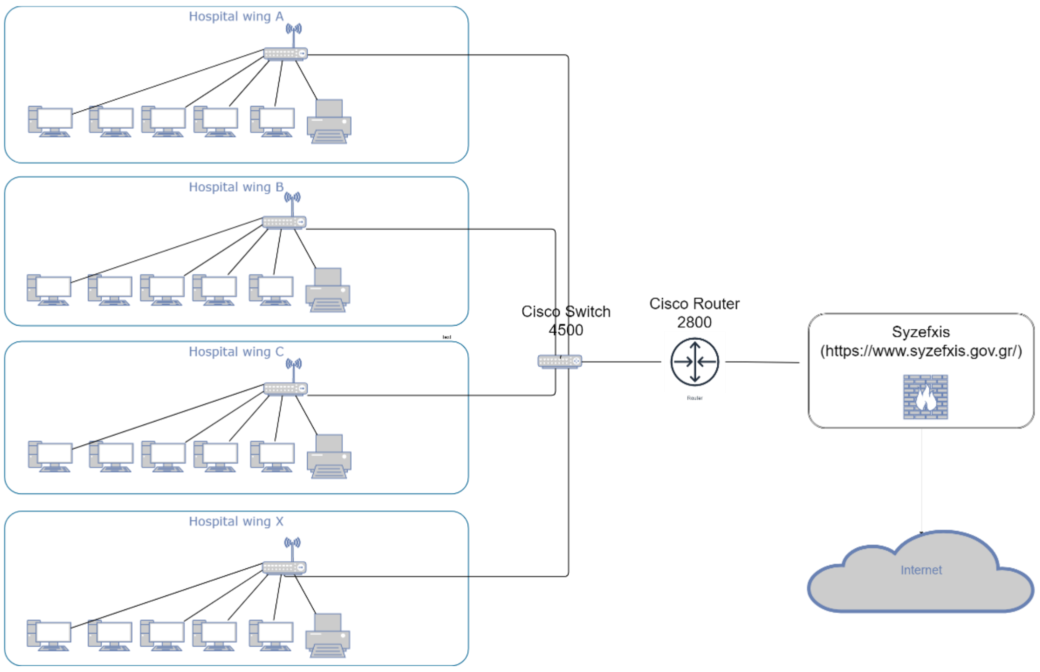


Figure 8: Indicative example view of the Dype5 network infrastructure

3.2 Traffic Sniffing from Pilot Networks

The network traffic capturing tool that will be used is Wireshark[15]. Wireshark is an open-source packet sniffer and analyser. It runs on most UNIX-like and Windows platforms.

The exact details of network traffic capturing depend on the specific network infrastructure. Nevertheless, this section attempts to present a general guideline based on the example network infrastructure from section 3.1.

[15] https://www.wireshark.org/#download

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

In order to capture traffic from a subnet there are 2 scenarios to consider:

a) The subnet contains a proxy server or firewall through which all the traffic is flowing. In this case Wireshark can be installed on this server. After starting Wireshark, the network interface that receives the network traffic has to be selected. Wireshark will automatically capture from the interface. To save the PCAP file, the capturing has to be stopped and the menu option File->Save has to be selected.

b) The subnet contains a switch with a port analyzer. Using the example in Figure 8, to capture traffic directly from the switch, SPAN has to be configured as described in the respective Cisco guide. A computer with Wireshark installed is connected to SPAN in order to capture the mirrored traffic, following the same principle as in scenario a).

One important issue to consider is that, for high volume network traffic, the PCAP file will become large. To manage this situation, the PCAP file should be rotated every GB or hour, depending on preferences. This can be done from Wireshark->Capture->Options->Output->Check Create a new file automatically.

The equivalent cli command for this capturing is:

```
tshark --interface enp0s3 --ring-buffer filesize:1048576 -n -w out.pcapng
```

3.3 Flow Metadata Extraction from Captured Traffic

In order to analyse the network traffic, it is not always necessary to have the full packets captured by Wireshark. It is sufficient to have a summarization of the network traffic flows containing relevant features (ie flow start date, duration, protocol, source and destination IP and port, number of bytes etc.). This can be realized using protocols like NetFlow, IPFIX, sampled NetFlow, sFlow etc.

Continuing the example above, in order to configure NetFlow on Cisco switch, the relevant Cisco guide[16] should be used:

Netflow does not capture whole traffic as PCAP but only specified "what and where" query.

In order to manually extract NetFlow information from the pcap file there a few possible approaches.

a) Create a virtual replica of the real network. The virtual network should contain an OpenvSwitch[17] instance. Open vSwitch hat has support for exporting NetFlow, sFlow, IPFIX etc. The traffic is then replayed and NetFlow information is extracted from the virtual switch.

b) Use the netflow exporter CicFlowMeter[18]

c) Use the netflow exporter NProbe[19], which supports NetFlow v9 and also includes layer 7 protocols.

[16] https://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst4500/12-2/52sg/configuration/guide/config/nfswitch.html
[17] https://www.openvswitch.org/
[18] https://github.com/ahlashkari/CICFlowMeter
[19] https://www.ntop.org/products/netflow/nprobe/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

4 Statistical Traffic Modelling and Generation

Traffic modelling is essential, despite the existence of diverse range of tools for pure traffic replay such as Tcpreplay[20], where the user needs to replay the captured traces, i.e. by retransmitting the sniffed packets exactly in the same order, and separated with the same properties (delays, burstiness, payload) without being able to add any stochasticity to the captured trace. Statistical traffic modelling methods continue to gather significant attention. Despite the fact that current techniques of "Deep Packet Inspection" (DPI) [6] provide better and more accurate information regarding the identification of the applications, they albeit require a complete and costly exploration of the payload of the packets. This induces an important load and is not practical when traffic is encrypted [7]. This section analyses the initial statistical modelling and replication framework of T5.4 referring to low-level (TCP-IP) traffic characteristics. Furthermore, DPI does not provide any traffic generation methodologies, as it can only serve as a traffic classifier.

One of the main goals of T5.4 is the replication of realistic network traffic similar to the one that occurs inside real pilot networks, with the purpose of testing the relevant infrastructures and the SPHINX components inside them. To achieve that, except for the emulation of a similar network topology, which is discussed in Section **Error! Reference source not found.**, it is of paramount importance to study and adopt frameworks and m ethodologies linked first with network traffic modelling techniques and user/network behaviour capturing and second with network realistic traffic generation based on these models. There are various ways of statistically modelling network traffic. Traffic analysis and relevant datasets (e.g. IDS datasets) can be categorized into network-based, host-based and application-based. [4]. Approaches and methodologies are further subdivided into packet-based and flow-based. The approach of this task is mainly focused on network-based and host-based modelling meaning that subnet-oriented and user-oriented traffic is mainly examined. Finally, both flow-level and packet-level approaches have been examined throughout T5.4 and have been integrated in the functionalities SPHINX ABS component. Nevertheless, the research has been mainly devoted to flow-level modelling and generation frameworks.

4.1 Flow-level Modelling

Flow-level network traffic modelling has played until now a significant role in the context of SPHINX Behaviour Simulation/ Experimentation environment and this deliverable. Such type of modelling permits easier tracing and provides a feasible and realistic way to setup simulation properties, which allows better replication of realistic hospital network traffic profiles.

4.1.1 NetFlow Protocol

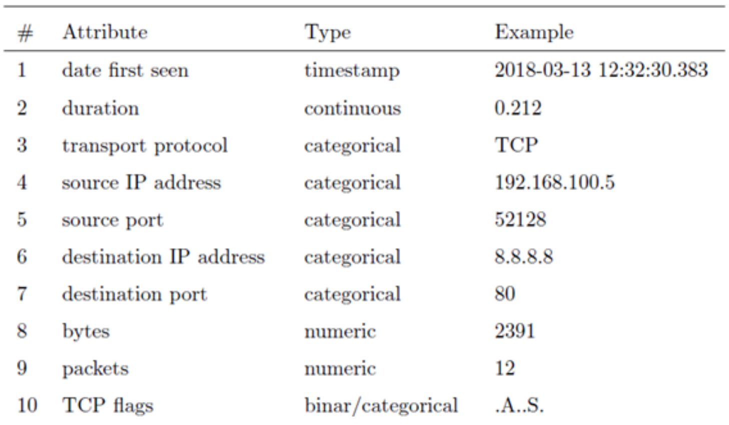
Flow-level models are based on the NetFlow [9] protocol. NetFlow is a network protocol developed by Cisco[21] for collecting IP traffic information and monitoring network flow. The Netflow protocol is supported by most modern routers and switches and results to exporting a traffic trace that basically constitutes an effective and compact aggregation of the traditional PCAP trace files. There exist multiple NetFlow standards and versions. However traditionally, an IP Flow is based on a set of 5 and up to 7 IP packet attributes which are generally similar to the ones depicted in the schema of Figure 9. Network traffic is thus aggregated in a manner that only specific traffic attributes are stored concerning a certain connection between two host IPs (e.g. TCP connection). This connection is also known as a flow. The main reason why researchers focus on the NetFlow data is that it captures only high level statistics which have been shown sufficient [10] for detecting threats and thus allows to process data from high speed backbone networks which cannot be achieved with other

[20] Tcpreplay website: http://tcpreplay.synfin.net/
[21] https://www.cisco.com/c/en/us/index.html

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

techniques (e.g. deep packet inspection) [11]. It is also of paramount importance that NetFlow traces are much smaller in size and storage requirements compared to full traces. Additionally, Flow information is extremely useful for understanding network behaviour as:

- Source address allows the understanding of who is originating the traffic
- Destination address tells who is receiving the traffic
- Ports characterize the application utilizing the traffic
- Class of service examines the priority of the traffic
- The device interface tells how traffic is being utilized by the network device
- Tallied packets and bytes show the amount of traffic



#	Attribute	Type	Example
1	date first seen	timestamp	2018-03-13 12:32:30.383
2	duration	continuous	0.212
3	transport protocol	categorical	TCP
4	source IP address	categorical	192.168.100.5
5	source port	categorical	52128
6	destination IP address	categorical	8.8.8.8
7	destination port	categorical	80
8	bytes	numeric	2391
9	packets	numeric	12
10	TCP flags	binar/categorical	.A..S.

Figure 9: Example of common NetFlow attributes

Considering two example hosts A, B NetFlow can be distinguished as follows:

- Unidirectional NetFlow**:** The flow of traffic from a host A to host B consists of packet exchange in two directions (A→B and B→A <reply>). These are considered two different unidirectional flows.
- Bidirectional NetFlow: The flow of traffic from a host A to host B is considered just one flow, from A→B. This includes the sum of packets from A→B and the reply B→A.

4.1.2 The On-Off Model

In this section, the types, the properties and the parameter estimation methodologies of the On-Off traffic modelling approach are described.

4.1.2.1 Simple On-Off Model

The On-Off model constitutes a modelling concept based on NetFlow. The simple on/off model is motivated by sources that alternate between active and idle periods [12]. An obvious example is speech where a person is either talking or listening. The time in the active state is exponentially distributed with mean $1/\alpha$, and the time in the idle state is exponentially distributed with mean $1/\beta$. In that case, the state of the source can be represented by the two-state continuous-time Markov chain shown in Figure 10.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

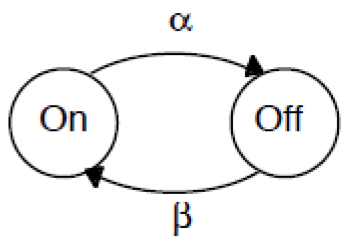


Figure 10: Two-state Markov chain

In the active state, the source is generating packets at a constant rate 1/T, i.e., inter-arrival times between packets are a constant T. In other words, the active state corresponds to the generation of a flow from the sender node towards a receiving node of the modelled network. In the idle state, no packets are generated. This on/off model can be considered a simple example of a modulated process, which is a combination of two processes. The basic process is a steady stream of packets at constant rate 1/T. The basic process is multiplied by a modulating process which is the Markov chain shown in Figure 10 (the active state is equivalent to 1 while the idle state is equivalent to 0). The modulation has the effect of cancelling the packets during the idle state. The resulting on/off traffic is shown in Figure 11.

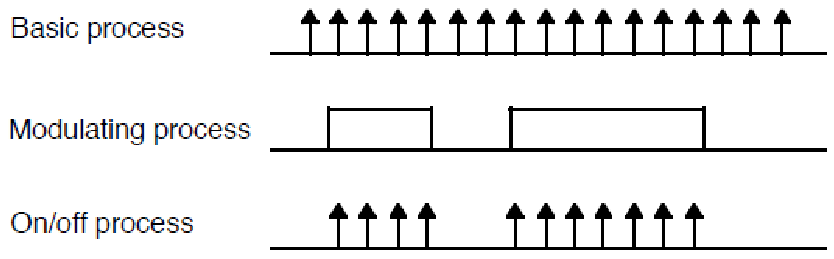


Figure 11: On/off traffic as example of a modulated process.

4.1.2.2 Advanced On-Off Model

W. Willinger proposed in [13] to generate data following packet-train models, i.e. with strictly alternating, independent and identically distributed ON and OFF periods and proved that these ON/OFF sources generate data similar to experimental measurements on real networks. Thus, the modelling process can be generalised from the exponential distribution of Section 4.1.2.1 to any desired distribution from which it is feasible to sample values from. As mentioned previously, the flow is associated with a source and destination couple and also with data, transmitted as a set of packets called a train of packets. The departure time of any source is computed with the departure time of the preceding source plus a random duration. This randomness follows the user defined “Doff distribution” (also named “distribution of inter-train durations”). Duration times should be similar for all sources. They should follow a user-defined random distribution called “Don distribution” (also named “distribution of train duration” or “distribution of flow duration”). It is of paramount importance to mention that the On/Off model does not model or replicate any payload information of the network traffic. On the contrary, it constitutes a throughput-oriented modelling method (duration and amount of data transferred in each flow are being parametrised and optimised). LAN-like throughput exhibits two main characteristics:

- **High variability** is characterized by an infinite mathematical variance and means that sudden discontinuous changes can always occur. Some mathematical distributions like Pareto and Weibull are heavy-tailed [14] (i.e. the tail of the distribution is not exponentially bounded) and thus can be used to generate sets of values with high variance and therefore high variability [15].

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

- **Self-similarity** which is defined by a long-range dependence characteristic, which means there are bursts of traffic any time over a wide range of time scales. In other words, a small sub-range of values is “similar” to the whole range of values.

Furthermore, it is common in literature that this duration is metered either in seconds or payload size (in Bytes) keeping mind the correlation that exists between them due to TCP congestion controls. Additionally, very often in literature TCP and UDP traffic are exclusively being modelled, given that they tend to occupy more than 95% percent of common network activity. [16]

4.1.2.3 Parameter Estimation

Following to the traffic capturing according to the NetFlow protocol, or alternatively the conversion of the ordinary PCAP trace to the NetFlow format, flows can be modeled per host or per subnet according to the attributes mentioned in the previous sections, that are flow duration and inter-train duration. Extracting this information from the NetFlow data leads to two independent empirical distributions (Don and Doff respectively) of values with respect to those two attributes. These distributions can be modeled according to the two following frameworks:

1. Based on well-known mainly heavy-tailed analytical distributions (or mixtures of them) and then:
    a. Arbitrarily select the parameters of the Don and Doff distributions. The selection of the Don and Doff distribution type can differ amongst networks depending on a variety of factors such as the domain (e.g. hospital, university campus, public domain or private business network etc.). Indicative default values can be obtained from [15], [17].
    b. Rely on parameter estimation methods based on optimization of likelihood-oriented criteria such as Maximum Likelihood (ML) [18], Bayesian Information Criterion (BIC), Akaike Information Criterion (AIC) [19]. These methods often lead to tractable objective functions, however stochastic optimization methods can always be followed such as the EM algorithm [20] or Gradient Descent algorithm [21].
2. Using Artificial Neural Networks (ANN) [22] and more specifically Deep Generative Models [23] such as VAEs [24], GANs [25] or Normalizing Flows [26]. These generative models are able to capture and learn the underlying probability distribution of the training data, irrespective of its complexity and multimodality, so that it could easily sample new data from that learned distribution. Deep generative models offer the opportunity to model easily more NetFlow attributes such as sender and destination IPs, ports and protocol as shown in [27].

In the context of T5.4 the research, until now, has been focused mainly on the first method, given that except for modeling also flow-based generation is required in order to replicate the traffic inside the emulator.

4.1.3 Flow-level Network Traffic Generation

SourcesOnOff [28] is the only validated flow level generator that permits the replication of network traffic and offers the capability for the Don and Doff Distributions to be defined by the user on the CLI. Harpoon [29] is another existing Open Source flow-level traffic generator, however this tool requires the user to define flow by flow the data weight he wants to transmit on the network. Moreover, the automatic generation of the network profile is limited to constant and uniform distributions.

The SourcesOnOff tool can be deployed in different hosts of the network emulating TCP and UDP clients and servers which exchange data flows. Summing up its functionalities, different sets of Don and Doff random values are generated. They are then used for data communications. Command-line parameters specify, for each set of (Don; Doff) values, the different socket properties: are they transmitters and so who is the destination host?

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

Or is it a bottomless pit? Do we use UDP or TCP protocol? What is the desired port number? Does IPv4 or IPv6 work better? Do a set of sources need to be delayed before starting (to let another set of sources take some advance)? Do the set of sources need to be stopped after a delay, even if there are still active sources? How many maximum sources should be created? Does the random generator be initialized with a specific seed (i.e. to reproduce two exact source generations)?

Figure 12 demonstrates an example of On-Off sources as activate by the SourcesOnOff tool while the following commands provide an example of the interactions with the tool from first in a VirtualBox virtualised subnet and second inside a docker bridge network.

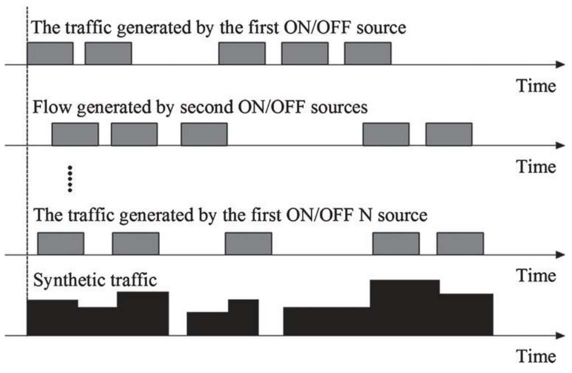

The traffic generated by the first ON/OFF source
Time
Flow generated by second ON/OFF sources
Time
The traffic generated by the first ON/OFF N source
Time
Synthetic traffic
Time

Figure 12: The On / Off generation model. Every source transmits according to a specific set of Don and Doff distributions

- **Example linux virtual network commands in a virtual subnet 10.0.0.0/24 to initiate a series of 100 tcp connections.** Transmitter (IP: 10.0.0.2) and receiver (IP: 10.0.0.3) VMs exchange 100 TCP flows whose payload-size and inter train durations are governed by two independent Weibull distributions of different (shape, scale) parameters with values (1, 900) and (1, 100) respectively: (Also see Figure 13: Two virtual machines exchanging tcp flows with flow and iter-flow durations governed by the Weibull distribution of different shape (k) and scale (λ) parameters)

  On host machine with IP 10.0.0.3:

```
./sourcesonoff --receiver-tcp
```

  On host machine with IP 10.0.0.2:

```
./sourcesonoff --transmitter-tcp --destination 10.0.0.3 --ipv4 --don-type Weibull
--don-lambda 900 --don-k 1 --doff-type Weibull --doff-lambda 1e7 --don-k 1 --turns
100
```

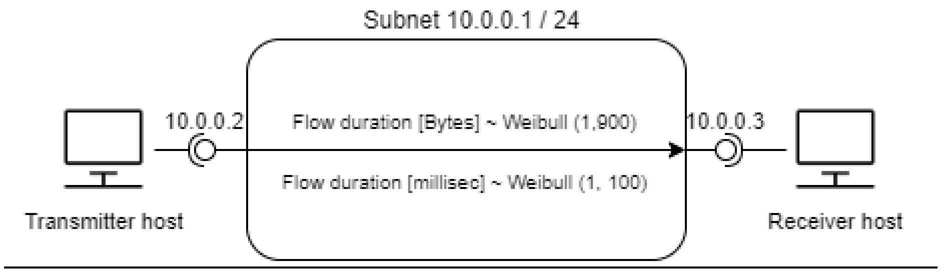

Subnet 10.0.0.1 / 24
10.0.0.2
Flow duration [Bytes] ~ Weibull (1,900)
Flow duration [millisec] ~ Weibull (1, 100)
10.0.0.3
Transmitter host
Receiver host

Figure 13: Two virtual machines exchanging tcp flows with flow and iter-flow durations governed by the Weibull distribution of different shape (k) and scale (λ) parameters

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

- **Example docker virtual bridge network command.** Transmitter (hostname: sender) and receiver (hostname: receiver) containers exchange 100 UDP flows whose payload-size and inter train durations are governed by two independentt Weibull distributions of different (shape, scale) parameters with values (1, 900) and (1,100) respectively.

  Container <receiver>:

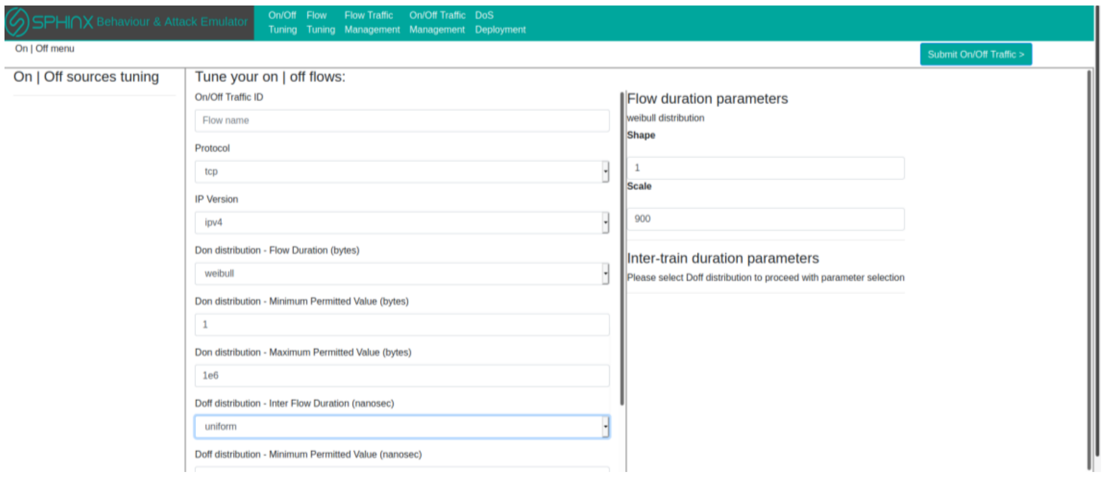
```
docker exec receiver ./sourcesonoff --receiver-udp
```

  Container <sender>:

```
docker exec sender ./sourcesonoff --transmitter-udp --destination receiver --ipv4
--don-type Weibull --don-lambda 900 --don-k 1 --doff-type Weibull --doff-lambda
1e7 --don-k 1 --turns 100
```

4.1.3.1 Flow-level Network Traffic Generation in the ABS component

Currently, the SPHINX ABS component **On-Off sources** menu implements a Web Graphical User Interface, which is based on the Flask[22] framework, for the interaction of the Docker hosts that run in the user created Docker network with the backend which is based on the SourcesOnOff application. The application is installed inside the Docker images and run by the network hosts permitting them to function as clients and servers during the flow exchange interactions.



On | Off sources tuning

Tune your on | off flows:

On/Off Traffic ID: Flow name

Protocol: tcp

IP Version: ipv4

Don distribution - Flow Duration (bytes): weibull

Don distribution - Minimum Permitted Value (bytes): 1

Don distribution - Maximum Permitted Value (bytes): 1e6

Doff distribution - Inter Flow Duration (nanosec): uniform

Doff distribution - Minimum Permitted Value (nanosec)

Flow duration parameters

weibull distribution

Shape: 1

Scale: 900

Inter-train duration parameters

Please select Doff distribution to proceed with parameter selection

Figure 14: The ABS GUI for the creation of On/Off Sources.

As shown in Figure 14, the user is prompted to select the values of the most important parameters required by the SourcesOnOff backend to initiate the exchange of the set of flows. Such parameters are the following for each set of flows (source) programmed by the user:

- Name
- Protocol: TCP, UDP
- IP Version: IPV4, IPV6
- Don distribution (Bytes): Weibull, Exponential, Normal, Uniform, Pareto
- Don distribution minimum permitted value (Bytes)
- Don distribution maximum permitted value (Bytes)

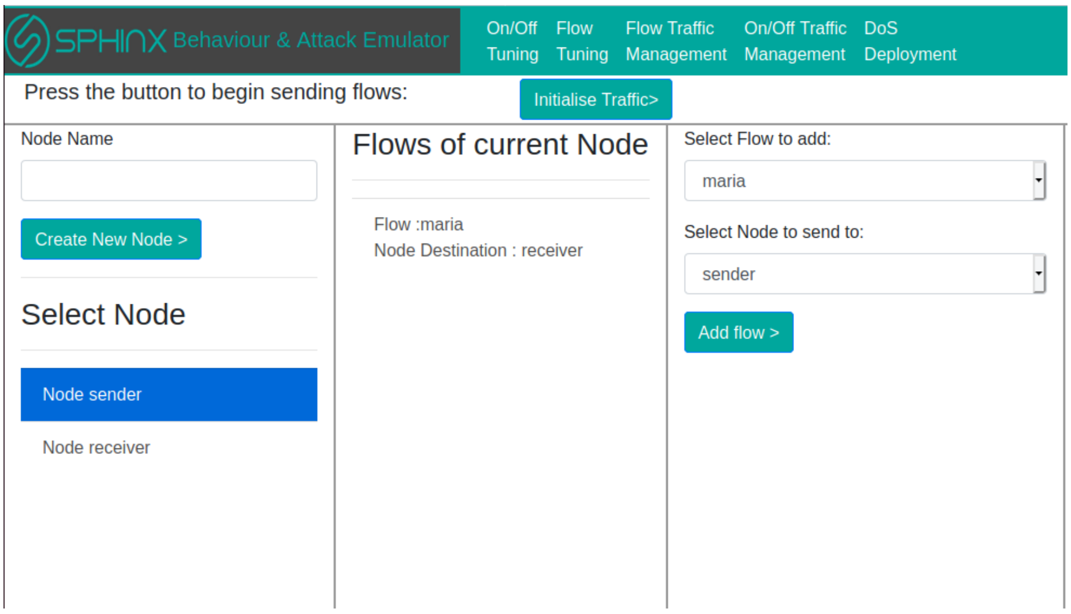
[22] https://flask.palletsprojects.com/en/1.1.x/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

- Doff distribution (nanoseconds): Weibull, Exponential, Normal, Uniform, Pareto
- Doff distribution minimum permitted value (nanoseconds)
- Doff distribution maximum permitted value (nanoseconds)
- Delay (nanoseconds)
- Number of flows (1e6 is considered as ∞)

This choice of parameters permits the user to strictly define a source/set of flows. After submitting a source, the user can submit as many as she/he desires in a similar manner and add them to the list of saved sources that are candidate for exchange amongst hosts.

Following, the user can proceed to the corresponding **On/Off Traffic Management** menu (see Figure 15**Error! Reference source not found.**) where he is given the opportunity to define the network nodes of her/his preference, always according to the topology defined in the docker-compose file and the container characteristics defined in DockerFiles (see Section 2). The user can refer to the network nodes according to their hostnames or IP addresses. After that she/he can create pairs of sender and receiver containers/network nodes in order to assign the previously defined On/Off sources for exchange. After matching the desired pairs to the respective sources through the Web GUI the user can push the **Initialise Traffic** button to start the transmission of all scheduled sources together. Sources that have been defined with non-zero delay will of course start transmitting later on.



Press the button to begin sending flows:

Node Name

Select Node

Node sender

Node receiver

Flows of current Node

Flow :maria
Node Destination : receiver

Select Flow to add: maria

Select Node to send to: sender

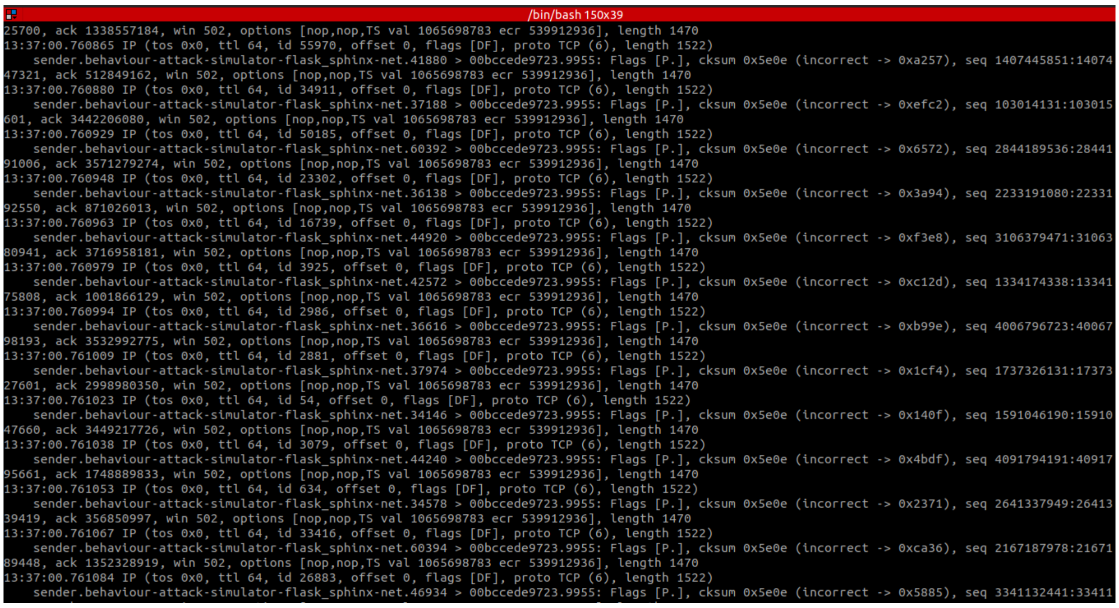
Figure 15: The ABS component GUI for the sending of the previously created On/Off sources.

The results of the process can easily be inspected in the terminal (see Figure 16) through the Docker environment using the preinstalled in each container sniffing application Tcpdump[23]. A typical command to inspect the incoming traffic of a specific host is structured as follows.

```
docker exec receiver tcpdump -Qin -l -v
```

[23] https://www.tcpdump.org/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.



```
25700, ack 1338557184, win 502, options [nop,nop,TS val 1065698783 ecr 539912936], length 1470
13:37:00.760865 IP (tos 0x0, ttl 64, id 55970, offset 0, flags [DF], proto TCP (6), length 1522)
    sender.behaviour-attack-simulator-flask_sphinx-net.41880 > 00bccede9723.9955: Flags [P.], cksum 0x5e0e (incorrect -> 0xa257), seq 1407445851:14074
47321, ack 512849162, win 502, options [nop,nop,TS val 1065698783 ecr 539912936], length 1470
13:37:00.760880 IP (tos 0x0, ttl 64, id 34911, offset 0, flags [DF], proto TCP (6), length 1522)
    sender.behaviour-attack-simulator-flask_sphinx-net.37188 > 00bccede9723.9955: Flags [P.], cksum 0x5e0e (incorrect -> 0xefc2), seq 103014131:103015
601, ack 3442206080, win 502, options [nop,nop,TS val 1065698783 ecr 539912936], length 1470
13:37:00.760929 IP (tos 0x0, ttl 64, id 50185, offset 0, flags [DF], proto TCP (6), length 1522)
    sender.behaviour-attack-simulator-flask_sphinx-net.60392 > 00bccede9723.9955: Flags [P.], cksum 0x5e0e (incorrect -> 0x6572), seq 2844189536:28441
91006, ack 3571279274, win 502, options [nop,nop,TS val 1065698783 ecr 539912936], length 1470
13:37:00.760948 IP (tos 0x0, ttl 64, id 23302, offset 0, flags [DF], proto TCP (6), length 1522)
    sender.behaviour-attack-simulator-flask_sphinx-net.36138 > 00bccede9723.9955: Flags [P.], cksum 0x5e0e (incorrect -> 0x3a94), seq 2233191080:22331
92550, ack 871026013, win 502, options [nop,nop,TS val 1065698783 ecr 539912936], length 1470
13:37:00.760963 IP (tos 0x0, ttl 64, id 16739, offset 0, flags [DF], proto TCP (6), length 1522)
    sender.behaviour-attack-simulator-flask_sphinx-net.44920 > 00bccede9723.9955: Flags [P.], cksum 0x5e0e (incorrect -> 0xf3e8), seq 3106379471:31063
80941, ack 3716958181, win 502, options [nop,nop,TS val 1065698783 ecr 539912936], length 1470
13:37:00.760979 IP (tos 0x0, ttl 64, id 3925, offset 0, flags [DF], proto TCP (6), length 1522)
    sender.behaviour-attack-simulator-flask_sphinx-net.42572 > 00bccede9723.9955: Flags [P.], cksum 0x5e0e (incorrect -> 0xc12d), seq 1334174338:13341
75808, ack 1001866129, win 502, options [nop,nop,TS val 1065698783 ecr 539912936], length 1470
13:37:00.760994 IP (tos 0x0, ttl 64, id 2986, offset 0, flags [DF], proto TCP (6), length 1522)
    sender.behaviour-attack-simulator-flask_sphinx-net.36616 > 00bccede9723.9955: Flags [P.], cksum 0x5e0e (incorrect -> 0xb99e), seq 4006796723:40067
98193, ack 3532992775, win 502, options [nop,nop,TS val 1065698783 ecr 539912936], length 1470
13:37:00.761009 IP (tos 0x0, ttl 64, id 2881, offset 0, flags [DF], proto TCP (6), length 1522)
    sender.behaviour-attack-simulator-flask_sphinx-net.37974 > 00bccede9723.9955: Flags [P.], cksum 0x5e0e (incorrect -> 0x1cf4), seq 1737326131:17373
27601, ack 2998980350, win 502, options [nop,nop,TS val 1065698783 ecr 539912936], length 1470
13:37:00.761023 IP (tos 0x0, ttl 64, id 54, offset 0, flags [DF], proto TCP (6), length 1522)
    sender.behaviour-attack-simulator-flask_sphinx-net.34146 > 00bccede9723.9955: Flags [P.], cksum 0x5e0e (incorrect -> 0x140f), seq 1591046190:15910
47660, ack 3449217726, win 502, options [nop,nop,TS val 1065698783 ecr 539912936], length 1470
13:37:00.761038 IP (tos 0x0, ttl 64, id 3079, offset 0, flags [DF], proto TCP (6), length 1522)
    sender.behaviour-attack-simulator-flask_sphinx-net.44240 > 00bccede9723.9955: Flags [P.], cksum 0x5e0e (incorrect -> 0x4bdf), seq 4091794191:40917
95661, ack 1748889833, win 502, options [nop,nop,TS val 1065698783 ecr 539912936], length 1470
13:37:00.761053 IP (tos 0x0, ttl 64, id 634, offset 0, flags [DF], proto TCP (6), length 1522)
    sender.behaviour-attack-simulator-flask_sphinx-net.34578 > 00bccede9723.9955: Flags [P.], cksum 0x5e0e (incorrect -> 0x2371), seq 2641337949:26413
39419, ack 356850997, win 502, options [nop,nop,TS val 1065698783 ecr 539912936], length 1470
13:37:00.761067 IP (tos 0x0, ttl 64, id 33416, offset 0, flags [DF], proto TCP (6), length 1522)
    sender.behaviour-attack-simulator-flask_sphinx-net.60394 > 00bccede9723.9955: Flags [P.], cksum 0x5e0e (incorrect -> 0xca36), seq 2167187978:21671
89448, ack 1352328919, win 502, options [nop,nop,TS val 1065698783 ecr 539912936], length 1470
13:37:00.761084 IP (tos 0x0, ttl 64, id 26883, offset 0, flags [DF], proto TCP (6), length 1522)
    sender.behaviour-attack-simulator-flask_sphinx-net.46934 > 00bccede9723.9955: Flags [P.], cksum 0x5e0e (incorrect -> 0x5885), seq 3341132441:33411
```

Figure 16: Receiving flow-based traffic at the server-side of the flow-based interaction

4.1.4 Packet-Level Modelling and Generation

4.1.4.1 Overview

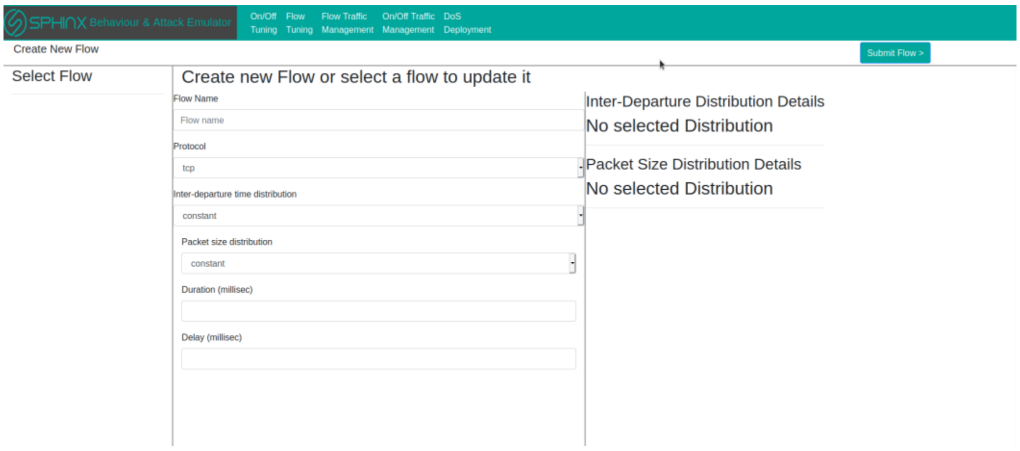
Packet-level modelling methods refer to the second group of statistical techniques for modelling network traffic that focus on low-level statistical traffic properties, that are packet-sizes and inter-packet times. Such models are more informative [7], [30] albeit more resource consuming than flow-level methods, given that every single packet size and time interval until the next one has to be modelled and yet generated using relevant packet-level generators. The methodology that has been followed is similar to that of Section 4.1.3 and therefore is not going to be described in detail in this section. The main differences of modelling can be determined in the following factors:

1. Packets are being modelled (payload size, inter-packet times) instead of flows using well-known analytical distribution
2. The NetFlow protocol is not useful anymore as it is incapable of capturing packet level behaviours of a trace.
3. They require packet-level traffic generators for traffic replication. However, such generators were found to be excessively computationally intensive and unreliable due to high latencies added by the overdetailed nature of the model, in terms of reproduction of realistic network traffic.

4.1.4.2 Packet-level Network Traffic Generation in the ABS component

The generation framework that was implemented during this research is based on the D-ITG [31] packet-level traffic generator. Similar to section 4.1.3, the SPHINX ABS component **Flow Tuning** menu (see Figure 17) implements a Graphical User interface for the interaction of the Docker hosts that run in the user created Docker network with the backend which is based on the D-ITG application. The application is installed inside the Docker images ran by the network hosts permitting them to function as clients and servers during the packet exchange interactions. The User has then to visit the **Flow Traffic Management** menu (see Figure 18) in order to schedule the previously created flows.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.



Create New Flow

Select Flow

Create new Flow or select a flow to update it

Flow Name: Flow name

Protocol: tcp

Inter-departure time distribution: constant

Packet size distribution: constant

Duration (millisec)

Delay (millisec)

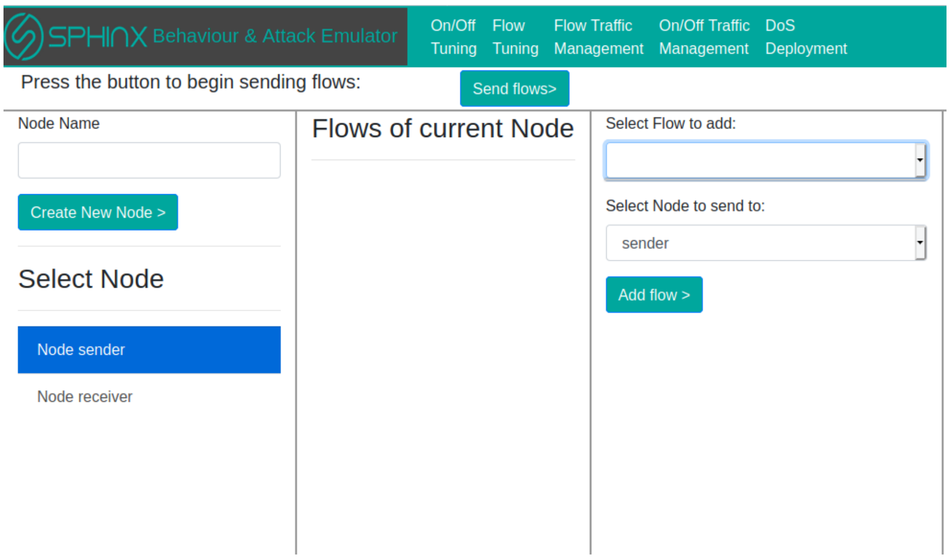
Inter-Departure Distribution Details

No selected Distribution

Packet Size Distribution Details

No selected Distribution

Figure 17: The ABS GUI for the creation of packet-based sources.



Press the button to begin sending flows:

Node Name

Select Node

Node sender

Node receiver

Flows of current Node

Select Flow to add:

Select Node to send to: sender

Figure 18: The ABS component GUI for the sending of the previously created packet-based flows.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

5 High Level Behaviour Simulation

5.1 User Behaviour Simulation in Hospital Networks

5.1.1 Behavioural Characteristics of a Traffic Trace

In order to extract user behaviour information from captured traffic (see Chapter 3), it is important to also keep track of the type of users behind machine IPs. For example, a user could be a doctor, IT staff, secretariat staff, patient, a medical monitoring device etc. IP addresses usually change, therefore tracking MAC addresses is essential. Possessing this abstract and anonymised information permits to associate the statistics data extracted from the captured file to a type of user, identify it's behaviour during the day (sites visited, number of interactions with mail servers, frequency of interaction with hospital servers etc.). The NetFlow information concerning the captured traffic can also be uploaded to a search and analytics engine such as Elasticsearch[24] and statistical information about all relevant parameters can be extracted from the real data.

5.1.2 Normal User Behaviour Simulation

Modelling network and user behaviours at a low-level TCP/UDP (transport layer) cannot be representative of specific application layer user activities that are really important in order to build a traffic and user behaviour profile for the hospital infrastructure. Hence, the future steps of T5.4 involve the simulation of normal user behaviour through scripts on the emulated network clients. During this process it is important that specific guidelines are defined for generation of realistic network traffic. The first guideline focuses on the realistic simulation of user behaviour whereas the second guideline is to take into account the heterogeneity of operating systems. To fulfil these guidelines, during the implementation of scripts the following features should be considered:

1. Be runnable on different operating systems;
2. Define typical computerized activities of employees;
3. Define different tasks and working methods of employees in possibly different subnets (e.g nurses, doctors, IT staff, managerial services);
4. Avoid periodic repetition of user activities;
5. Define typical working hours and breaks.

The first feature is met by using the platform independent language Python for implementing the user behaviour scripts/agents. In the context of user behaviour, hospital employees have a wide range of activities during their daily work such as composing and sending emails, creating documents and presentations, browsing (private or business searches), printing, sharing files and so on. More specifically, for file transfers and printing tasks, it is important to ensure that the corresponding files vary in terms of type and size. In the same context, when sending emails, the number of attachments should vary amongst iterations of the process. Realistic user behaviour cannot be characterized by repeating (replaying) a list of activities periodically. Instead, the temporal sequence of user activities should be randomised, and the kind of activities should vary. Additionally, activities should not be totally random instead follow probability distributions which subject to typical working hours. Typically, employees are not permanently performing tasks which cause network traffic. It is important to consider meetings, offline work or coffee breaks as mentioned above. Scripts should be developed in a way that boosts traffic footprints during working hours and minimises activities during the evening or breaks, when at least most managerial and IT premises of the hospitals are inactive.

[24] https://www.elastic.co/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

For simulating such activities with respect to potential different characteristics of different employees according to their habits and working position, each emulated client machine will require a unique configuration file. Configuration files shall control the kind and frequency of activities and for each client concerning each activity. Hence, different user profiles will be assigned to different clients of the network. Client configuration files should of course be modular and modifiable in order to effectively reproduce various user behaviours.

All these user behaviour patterns are going to be extracted either from analysing the network traces that will be collected based on the methodology presented in Section 3 or from extensive questionnaires regarding the activities of employees in different departments, labs and subnetworks of the pilots.

5.1.3 Abnormal Behaviour Simulation

Normal network traffic is planned to be generated by Python scripts as described above. The task of generation of malicious network traffic will follow a different process that is mainly discussed in deliverable D5.3. Summarising this process, some (internal or external to the main emulated hospital network) clients are planned to perform various attacks using Linux tools and Python scripts (e.g Nmap[25] for scanning, Metasploit[26] for server exploitations, hping3[27] and Heartleech[28] for DoS attacks, Brute Force Attacks, Worm code execution etc). Such attacks will be deployed by the ABS user inside the simulation/emulation environment, preserving the general concepts that were presented in Section 5.1.1.

5.2 Remote Healthcare Monitoring: User Behaviour Simulation

Remote medical and health devices allow healthcare professionals to closely follow patients outside of the office, either through telehealth (video consultation) or remote patient monitoring, the latter involving connected devices, Apps and/or web-applications. Apps collect patient-generated health data (manually or automatically through devices) and connect, via the patient's home network or cellular network, to healthcare provider's web-services to be used by the primary care provider or care team. Since remote healthcare provision involves interaction with outside networks and systems (e.g., the patient's home environment), it also brings a number of challenges in what regards cybersecurity since it creates new, often insecure, entry points for hackers and rising data security and liability risks. For example, *Users of mobile devices are increasingly subject to malicious activity that is pushing malware apps to their phones, tablets and other devices running Android and iOS* [D2.1 (1.5.2)]. These malicious applications can steal personal data and be used to conduct illicit network activities. Since these devices are outside the healthcare organisation's control, there is also a lack of visibility and control over personal devices, as well as the absence of awareness of these devices' vulnerabilities that attackers could take advantage of.

SPHINX addresses use-cases involving remote patient monitoring, aiming to prevent and detect suspicious and illicit activity. For this purpose, EDGE brings the eCare Platform Testbed, a simulation environment that allows generating requests and interactions between **patients** (i.e., simulated eCare Platform users), **devices** (i.e., simulated healthcare IoT equipment) and **healthcare services**.

EDGE eCare and the associated testbed are described next.

[25] https://nmap.org/
[26] https://www.metasploit.com/
[27] https://tools.kali.org/information-gathering/hping3
[28] https://en.kali.tools/all/?tool=1759

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

5.2.1 EDGE eCare

EDGE's eCare Platform (eCare) is EDGE's smart and personalised care intelligence platform that creates a welcoming assisted environment for health and wellness at home. EDGE's healthcare assisted ambient delivers high-quality and comfortable healthcare to patients requiring observation or monitoring, either in post-medical intervention or in chronic condition situations.

eCare is designed to support the unobtrusive monitoring of health-related conditions and behaviour in home-based interventions, offering a personalised healthcare framework assisted by multimodal sensing. eCare gathers a wide range of measurements acquired **manually or automatically through heterogeneous sensors and devices** that are seamlessly embedded in the living environment or are worn or interacted with by users/patients. Amongst the measurements registered in eCare's remote monitoring platform are **vital signs (temperature, heart rate, blood glucose, blood pressure and respiration rate)** and **physiological measurements** (**weight, physical activity**).

In addition to health-related data, the eCare Platform also measures activity data from wearables or home devices (e.g., motion sensor) as well as measurements of environmental parameters, including temperature, humidity, air quality and smoke.

In the context of the SPHINX Project, the eCare Platform is the centrepiece of the pilot activities in Portugal, concerning cybersecurity and the adoption of mobile Health and remote monitoring platforms by healthcare providers. The eCare Platform used is illustrated in Figure 19.

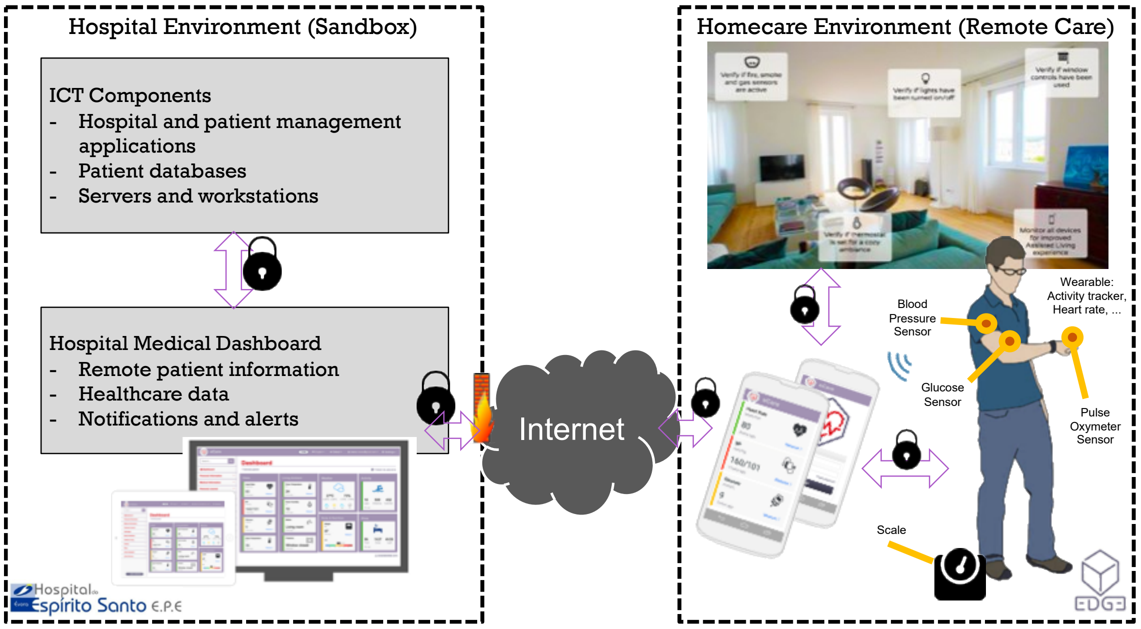

Hospital Environment (Sandbox)
ICT Components
- Hospital and patient management applications
- Patient databases
- Servers and workstations
Hospital Medical Dashboard
- Remote patient information
- Healthcare data
- Notifications and alerts
Hospital Espírito Santo E.P.E
Internet
Homecare Environment (Remote Care)
Verify if fire, smoke and gas sensors are active
Verify if lights have been turned on/off
Verify if window controls have been used
Verify if thermostat is set for a cozy ambiance
Monitor all devices for improved Assisted Living experience
Blood Pressure Sensor
Wearable: Activity tracker, Heart rate, ...
Glucose Sensor
Pulse Oxymeter Sensor
Scale
EDGE

Figure 19: The mHealth and Remote Patient Monitoring Service planned for the SPHINX Pilot in Portugal

The eCare Platform comprises the **eCare portal**, a server-side component installed in the Hospital's facilities ("Hospital Environment" (trusted and controlled network) that hosts the patients' data and that, upon authentication, is accessible via web browsers by healthcare professionals.

The "Homecare Environment" refers to the environment where the patient (or user) lives, presenting heterogeneous sensors and devices that measure health-related parameters. Users/patients interact with the eCare Platform via web browsers or through the **eCare App**, either by viewing their health parameters as they are automatically acquired by the eCare Platform or by inserting manually their health and wellbeing

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

measurements. With the users/patients' authorisation, this data is shared in near real time with the eCare Platform at the point of care and accessed by authorised medical staff.

5.2.2 EDGE eCare Testbed

EDGE's eCare Testbed allows simulating the interaction between eCare users (i.e., patients and devices) and the eCare Platform. The eCare Testbed allows to create simulated users that interact with the eCare portal and backend services by sending simulated data from heterogeneous simulated sources (e.g., heart rate measurement from a healthcare device and temperature from a home sensor). The data exchange process follows the same mechanism as if it were a real user, thus resulting in a realistic setting for the testing and validation of the eCare Platform. EDGE's eCare Testbed is illustrated in Figure 20.

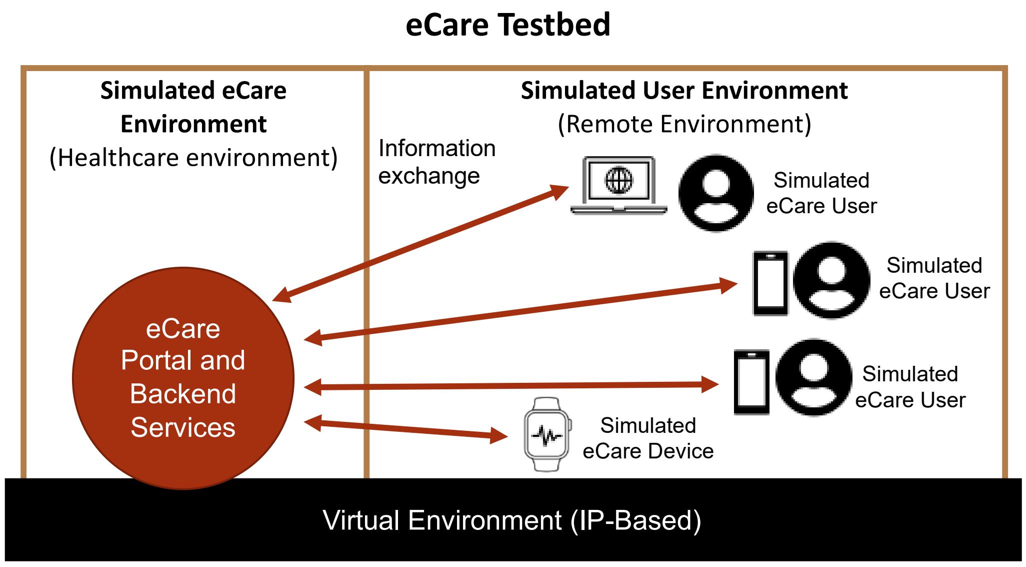


Figure 20 - eCare Testbed

The eCare Testbed comprises the following tools:

- Virtual environment for setup and deployment, using Vagrant (https://www.vagrantup.com/) based on Oracle VirtualBox emulator.
- Testing and Staging environments, allowing testing eCare locally or under controlled (albeit realistic) conditions, similar to the production environment. It includes a set of deployment and configuration scripts that setup the eCare Platform for testing purposes.
- The eCare User Simulator that represents a simulated user of eCare, interacting with the eCare Platform in the same way a real user does. It generates information pertaining to data measurements from heterogeneous devices. The eCare User Simulator is subjected to the same authentication process of a real user.
- The eCare Device Simulator that represents a simulated device of eCare, interacting with the eCare Platform in the same way a real eCare device does. It generates information pertaining to the applicable data measurement. The eCare Device Simulator is subjected to the same authentication process of a real device.

5.2.3 EDGE's eCare Testbed in SPHINX

In SPHINX, EDGE's eCare Testbed will be adapted to be part of the SPHINX Simulation Environment and support user behaviour analysis, in the context of remote monitoring healthcare applications.

The main features of the eCare User and Device Simulators are:

- To create and instantiate one or multiple simulated users/devices;

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

- To simulate the user/device authentication process;
- Where applicable, to simulate the periodic production of data associated with:
    - Vitals (e.g., heart rate, blood pressure);
    - Wellbeing (e.g., weight, sleep);
    - Physical activity (e.g., number of steps);
    - Ambient (e.g., temperature, indoor air quality, motion);
- To set data frequency to a fixed or a random value;
- To set data content to a fixed value plus a random variation.

Leveraging on the defined pilot case in Portugal, the eCare Testbed is suitable to be adapted to meet the SPHINX Simulation Environment needs pertaining remote healthcare monitoring.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

6 Conclusions and Future Plans

6.1 Conclusions

This deliverable presented the level of research conducted throughout Task 5.4 of the SPHINX H2020 project, along with the overall development status of the respective parts of the ABS component on M22 of the project's lifetime. Within this first research and development phase, the following accomplishments have been achieved:

- Docker and VM based environments have been deployed and used for experiments concerning the generation of flow-based network traffic. Docker containers proved to be more flexible in terms of resources and deployment capabilities.
- Two automated prototyping technologies for NFV have been studied and compared according to their flexibility, scalability, user friendliness and ability to host multiple virtualisation technologies. Amongst them, ContainerNet proved to serve better the future experiments of the Behaviour Simulator and the ABS component as a whole.
- Network traffic sniffing methodologies have been examined thoroughly in order to permit the accurate, valid and anonymous traffic trace extraction from the hospital premises as well as the emulation testbeds.
- Various network traffic modelling and generation methodologies have been studied and then have been deployed inside experimental virtual networks.
- Extensive research has been conducted on effective frameworks for user/ host behaviour simulations inside a network. Furthermore, the EDGE testbed has been presented whose functionality is the realistic simulation of the interaction between hospital eCare users and platform.
- A first prototype of the Behaviour Simulator of the ABS component has been implemented, allowing topology emulations through the Docker Engine (DockerFiles and docker-compose), and the generation of flow-based network traffic. Traffic parameters and flow scheduling are defined by the component's user through a friendly Web GUI.

6.2 Future plans

The next steps of this task which will lead to the final deliverable/demonstrator of the Behaviour Simulator include:

- Implementation of a user-friendly NFV prototyping environment mainly based on docker containers and potentially supporting other virtualisation technologies (e.g. VMs). This environment will constitute the core of the ABS component and will permit researchers to create arbitrary network topologies.
- Capability of deploying the SPHINX components inside the emulator for testing purposes through Docker.
- Synergy and deployment within the SPHINX Sandbox in order to ensure the security of experiments.
- The development and integration of user behaviour simulation scripts, which will be run by network hosts, instead of flow-based traffic generation. Hence a higher level and more accurate user behaviour modelling will be achieved.
- Integration of the EDGE testbed in the emulated environments.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

7 References

[1] I. Sharafaldin, A. Gharib, A. H. Lashkari, and A. A. Ghorbani, “Towards a Reliable Intrusion Detection Benchmark Dataset,” *Softw. Netw.*, vol. 2018, no. 1, pp. 177–200, Jan. 2018, doi: 10.13052/jsn2445-9739.2017.009.

[2] O. Heckmann, M. Piringer, J. Schmitt, and R. Steinmetz, “On realistic network topologies for simulation,” in *Proceedings of the ACM SIGCOMM workshop on Models, methods and tools for reproducible network research*, 2003, pp. 28–32.

[3] F. N. N. Farias, A. de O. Junior, L. B. da Costa, B. A. Pinheiro, and A. J. G. Abelém, “vSDNEmul: A Software-Defined Network Emulator Based on Container Virtualization,” *ArXiv190810980 Cs*, Aug. 2019, Accessed: Oct. 02, 2020. [Online]. Available: http://arxiv.org/abs/1908.10980.

[4] B. Lantz, B. Heller, and N. McKeown, “A network in a laptop: rapid prototyping for software-defined networks,” in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010, pp. 1–6.

[5] M. Peuster, J. Kampmeyer, and H. Karl, “Containernet 2.0: A rapid prototyping platform for hybrid service function chains,” in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, 2018, pp. 335–337.

[6] A. W. Moore and K. Papagiannaki, “Toward the accurate identification of network applications,” in *International Workshop on Passive and Active Network Measurement*, 2005, pp. 41–54.

[7] M. Jaber, R. G. Cascella, and C. Barakat, “Can we trust the inter-packet time for traffic classification?,” in *2011 IEEE International Conference on Communications (ICC)*, 2011, pp. 1–5.

[8] M. Ring, S. Wunderlich, D. Grüdl, D. Landes, and A. Hotho, “Flow-based benchmark data sets for intrusion detection,” in *Proceedings of the 16th European conference on cyber warfare and security*, 2017, pp. 361–369.

[9] B. Claise, G. Sadasivan, V. Valluri, and M. Djernaes, “Cisco systems netflow services export version 9,” 2004.

[10] P. Barford and D. Plonka, “Characteristics of network traffic flow anomalies,” in *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, 2001, pp. 69–73.

[11] J. Stiborek, M. Rehák, and T. Pevnỳ, “Towards scalable network host simulation,” in *SECOND INTERNATIONAL WORKSHOP ON AGENTS AND CYBERSECURITY*, 2015, p. 27.

[12] T. M. Chen, “Network traffic modeling,” in *The handbook of computer networks*, vol. 3, Wiley Hoboken, NJ, 2007, p. 156.

[13] W. E. Leland, W. Willinger, M. S. Taqqu, and D. V. Wilson, “On the self-similar nature of Ethernet traffic,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 25, no. 1, pp. 202–213, 1995.

[14] S. Luo and G. A. Marin, “Realistic internet traffic simulation through mixture modeling and a case study,” in *Proceedings of the Winter Simulation Conference, 2005.*, 2005, pp. 9–pp.

[15] P. Olivier and N. Benameur, “Flow level IP traffic characterization,” in *Teletraffic Science and Engineering*, vol. 4, Elsevier, 2001, pp. 25–36.

[16] A. Varet and N. Larrieu, “Realistic network traffic profile generation: theory and practice,” 2014.

[17] S. Gebert, R. Pries, D. Schlosser, and K. Heck, “Internet access traffic measurement and analysis,” in *International Workshop on Traffic Monitoring and Analysis*, 2012, pp. 29–42.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

[18] I. J. Myung, “Tutorial on maximum likelihood estimation,” *J. Math. Psychol.*, vol. 47, no. 1, pp. 90–100, Feb. 2003, doi: 10.1016/S0022-2496(02)00028-7.

[19] “AIC and BIC: Comparisons of Assumptions and Performance - Jouni Kuha, 2004.” https://journals.sagepub.com/doi/abs/10.1177/0049124103262065 (accessed Sep. 25, 2020).

[20] “The EM Algorithm and Extensions - Geoffrey J. McLachlan, Thriyambakam Krishnan - Google Books.” https://books.google.gr/books?hl=en&lr=&id=NBawzaWoWa8C&oi=fnd&pg=PR3&dq=em+algorithm+&ots=tp63MK0DyO&sig=m2uC3QysE-PIKsruwS9LOXOiwo0&redir_esc=y#v=onepage&q=em%20algorithm&f=false (accessed Sep. 25, 2020).

[21] S. Ruder, “An overview of gradient descent optimization algorithms,” *ArXiv160904747 Cs*, Jun. 2017, Accessed: Sep. 25, 2020. [Online]. Available: http://arxiv.org/abs/1609.04747.

[22] B. Yegnanarayana, *Artificial neural networks*. PHI Learning Pvt. Ltd., 2009.

[23] D. P. Kingma, S. Mohamed, D. Jimenez Rezende, and M. Welling, “Semi-supervised Learning with Deep Generative Models,” in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 3581–3589.

[24] D. P. Kingma and M. Welling, “An Introduction to Variational Autoencoders,” *Found. Trends® Mach. Learn.*, vol. 12, no. 4, pp. 307–392, 2019, doi: 10.1561/2200000056.

[25] I. J. Goodfellow *et al.*, “Generative Adversarial Networks,” *ArXiv14062661 Cs Stat*, Jun. 2014, Accessed: Sep. 25, 2020. [Online]. Available: http://arxiv.org/abs/1406.2661.

[26] I. Kobyzev, S. J. D. Prince, and M. A. Brubaker, “Normalizing Flows: An Introduction and Review of Current Methods,” *IEEE Trans. Pattern Anal. Mach. Intell.*, pp. 1–1, 2020, doi: 10.1109/TPAMI.2020.2992934.

[27] M. Ring, D. Schlör, D. Landes, and A. Hotho, “Flow-based network traffic generation using Generative Adversarial Networks,” *Comput. Secur.*, vol. 82, pp. 156–172, May 2019, doi: 10.1016/j.cose.2018.12.012.

[28] A. Varet and N. Larrieu, “How to generate realistic network traffic?,” in *2014 IEEE 38th annual computer software and applications conference*, 2014, pp. 299–304.

[29] J. Sommers, H. Kim, and P. Barford, “Harpoon: A Flow-Level Traffic Generator for Router and Network Tests,” p. 2.

[30] E. Garsva, N. Paulauskas, G. Grazulevicius, and L. Gulbinovic, “Packet Inter-arrival Time Distribution in Academic Computer Network,” *Elektron. Ir Elektrotechnika*, vol. 20, no. 3, Art. no. 3, Mar. 2014, doi: 10.5755/j01.eee.20.3.6683.

[31] “D-ITG distributed Internet traffic generator - IEEE Conference Publication.” https://ieeexplore.ieee.org/abstract/document/1348045/ (accessed Sep. 17, 2020).

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

Annex I:

I.1 The docker-compose file that corresponds to the topology of Figure 3.

```
---
version: '3'


version: "3"
services:

  node1:
    #image:behaviour-attack-simulator-flask_node1
    container_name: sender
    stdin_open: true
    tty: true
    build:
      context: .
      dockerfile: Dockerfile-attacker
    networks:
      - sphinx-net
    working_dir: /app
    command: ["./sender-receiver.sh"]

  node2:
    #image:behaviour-attack-simulator-flask_node2
    container_name: receiver
    stdin_open: true
    tty: true
    build:
      context: .
      dockerfile: Dockerfile-victim
    networks:
      - sphinx-net
    working_dir: /app
    command: ["./sender-receiver.sh"]

  supervisor:
    #image:behaviour-attack-simulator-flask_supervisor
    container_name: super
    stdin_open: true
    tty: true
    build:
      context: .
      dockerfile: Dockerfile-supervisor
    ports:
      - "5002:5002"
    networks:
      - sphinx-net
    volumes:
      - '/var/run/docker.sock:/var/run/docker.sock'
    command: ["./flask.sh"]
```

```
networks:
  sphinx-net:
    driver: bridge
```

I.2 The docker-compose file that corresponds to the topology of Figure 4[29].

```
---
version: '3'
services:

#-----------------
# PUBLIC
#-----------------
ind_sec:
image: industrial-security
ports:
- "18837:8837"
networks:
public:
ipv4_address: 10.111.220.11

#-----------------
# IT
#-----------------
nnm_IT:
image: nnm
depends_on:
- ind_sec
ports:
- "18835:8835"
networks:
public:
ipv4_address: 10.111.220.21
IT:
ipv4_address: 10.111.221.21

nessus:
image: nessus
ports:
- "18834:8834"
networks:
public:
ipv4_address: 10.111.220.23
IT:
ipv4_address: 10.111.221.23

IT_player:
build: ./docker/pcap_player
depends_on:
- nnm_IT
environment:
```

[29]Source: https://medium.com/tenable-techblog/simulating-enterprise-networks-in-development-using-the-docker-networking-stack-94bf547743c9

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

```
- loop=4
- speed=2
volumes:
- ./docker/pcap_player/pcaps/IT:/tmp/pcaps
networks:
IT:
ipv4_address: 10.111.221.31

#-----------------
# OT
#-----------------
nnm_OT:
image: nnm
depends_on:
- ind_sec
ports:
- "18836:8835"
networks:
public:
ipv4_address: 10.111.220.22
OT:
ipv4_address: 10.111.222.22

OT_player:
build: ./docker/pcap_player
depends_on:
- nnm_OT
environment:
- loop=4
- speed=2
volumes:
- ./docker/pcap_player/pcaps/OT:/tmp/pcaps
networks:
OT:
ipv4_address: 10.111.222.31

#-----------------
# TAP
#-----------------
tap:
build: ./docker/pcap_player
volumes:
- ./docker/pcap_player/pcaps:/tmp/pcaps
networks:
public:
ipv4_address: 10.111.220.32
IT:
ipv4_address: 10.111.221.32
OT:
ipv4_address: 10.111.222.32
command: bash

networks:
public:
driver: "bridge"
ipam:
```

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.

```
config:
- subnet: 10.111.220.1/24
IT:
driver: "bridge"
ipam:
config:
- subnet: 10.111.221.1/24
OT:
driver: "bridge"
ipam:
config:
- subnet: 10.111.222.1/24
```

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 826183 - Digital Society, Trust & Cyber Security E-Health, Well-being and Ageing.