D5.4 – Network Resilience Evaluation Framework
<table>
<thead>
<tr>
<th><strong>Document Number</strong></th>
<th>D5.4</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Status</strong></td>
<td>Completed</td>
</tr>
<tr>
<td><strong>Work Package</strong></td>
<td>WP5</td>
</tr>
<tr>
<td><strong>Deliverable Type</strong></td>
<td>Report</td>
</tr>
<tr>
<td><strong>Date of Delivery</strong></td>
<td>13.12.2017</td>
</tr>
<tr>
<td><strong>Responsible Unit</strong></td>
<td>Fraunhofer</td>
</tr>
</tbody>
</table>
| **Editors**         | Ranjan Shrestha (TUB)  
                      | Marius Corici (Fraunhofer) |
| **Contributors**    | Jaafar Bendriss (Orange)  
                      | Teodora Sandra Buda (IBM)  
                      | Marius Corici (Fraunhofer)  
                      | Michael Crotty (WIT)  
                      | Fabrizio Granelli (UNITN)  
                      | Imen Gridabenyahia (Orange)  
                      | Iryna Haponchyk (UNITN)  
                      | Durga Prasad Kakollu (WIT)  
                      | Taner Metin (TUB)  
                      | Alessandro Moschitti (UNITN)  
                      | Andrea Passerini (UNITN)  
                      | Antonio Agustin Pastor Perales (TID)  
                      | Ranjan Shrestha (TUB)  
                      | Ryan Stubbs (WIT)  
                      | Zsolt Szabó (Fraunhofer)  
                      | Martin Tolan (WIT)  
                      | Kateryna Tymoshenko (UNITN)  
                      | Tom Walsh (WIT) |
| **Reviewers**       | Marius Corici (Fraunhofer)  
                      | Martin Tolan (WIT) |
| **Dissemination level** | PU          |
### Change History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Status</th>
<th>Editor (Unit)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>07.09.2017</td>
<td>Draft</td>
<td>Marius Corici (Fraunhofer)</td>
<td>Provided initial template and deliverable structure based on the intended contributions</td>
</tr>
<tr>
<td>0.2</td>
<td>03.11.2017</td>
<td>Draft</td>
<td>Ranjan Shrestha (TUB)</td>
<td>Added contributions from UNITN, IBM, TUB</td>
</tr>
<tr>
<td>0.3</td>
<td>10.11.2017</td>
<td>Draft</td>
<td>Ranjan Shrestha (TUB)</td>
<td>Added contributions from Orange, WIT</td>
</tr>
<tr>
<td>0.4</td>
<td>11.11.2017</td>
<td>Draft</td>
<td>Ranjan Shrestha (TUB)</td>
<td>Added additional contributions from Fraunhofer, References fixing and formatting of document.</td>
</tr>
<tr>
<td>0.5</td>
<td>14.11.2017</td>
<td>Draft</td>
<td>Ranjan Shrestha (TUB)</td>
<td>List of contributors from UNITN added.</td>
</tr>
<tr>
<td>0.6</td>
<td>14.11.2017</td>
<td>Draft</td>
<td>Marius Corici (Fraunhofer)</td>
<td>Completed the document. Full review.</td>
</tr>
<tr>
<td>1.0</td>
<td>18.11.2017</td>
<td>Draft</td>
<td>Marius Corici (Fraunhofer)</td>
<td>Adaptation of the content based on the internal review.</td>
</tr>
<tr>
<td>1.1</td>
<td>04.12.2017</td>
<td>Draft</td>
<td>Ranjan Shrestha (TUB)</td>
<td>Addressed the comments from WIT (contributions from Fraunhofer, TUB, Orange)</td>
</tr>
<tr>
<td>1.2</td>
<td>13.12.2017</td>
<td>Final</td>
<td>Marius Corici (Fraunhofer)</td>
<td>Completed the responses to the comments. Clean-up of the final version</td>
</tr>
</tbody>
</table>
Executive Summary

This deliverable is the last deliverable of WP5 and includes the features and work not previously described in D5.2 and D5.3 underlining the integration within existing reference implementation of software networks management components such as OpenSDNCore for OpenFlow SDN-based solutions and OpenStack as the reference architecture for the management of virtualized infrastructures.

This deliverable additionally includes a set of evaluation targets, metrics and tools aiming to underline the key technological advancements produced by WP5 in using and extending existing machine learning techniques for the support of management of dynamic network environments, specifically focusing in the direction of securing the network deployments and subscriber connectivity as well as assuring the appropriate resilience and SLA levels for it.

Following, the qualitative evaluations are presented and assessed on top of the prototypes which were previously developed (and described in D5.2 and D5.3), proving the opportunity of having the specific advancements as part of the next evolution of the network management products would be beneficial.

To be able to prove in a more realistic way the results, the testbeds were implemented including the actuation of the system, aiming to prove that machine learning, especially for resilience and security is providing its benefits also for real-time sensing and issue mitigation, specifically responding to the requirements of the virtualized environment.
Table of Contents

1. Introduction ................................................................................................................................. 9
   1.1. Motivation, Objective and Scope ............................................................................................ 9

2. Distributed Security Enablement Testbed .................................................................................. 11
   2.1. Distributed Security Enablement Testbed .............................................................................. 11
      2.1.1. Evaluation Scope ............................................................................................................... 11
      2.1.2. Evaluation Metrics .......................................................................................................... 12
      2.1.3. Evaluation Tools ............................................................................................................. 17
      2.1.4. Testbed description ......................................................................................................... 19
      2.1.5. Evaluation Results and their assessment ........................................................................... 19
      2.1.6. Conclusions and next steps ............................................................................................ 20
   2.2. Honeynet Testbed .................................................................................................................... 22
      2.2.1. Evaluation Scope ............................................................................................................... 22
      2.2.2. Evaluation Metrics .......................................................................................................... 22
      2.2.3. Evaluation Tools ............................................................................................................. 22
      2.2.4. Testbed description ......................................................................................................... 24
      2.2.5. Evaluation Results and their assessment ........................................................................... 24
      2.2.6. Conclusions and next steps ............................................................................................ 26
   2.3. NFV Security Anomaly Detection Testbed .............................................................................. 28
      2.3.1. Evaluation Scope ............................................................................................................... 28
      2.3.2. Evaluation Metrics .......................................................................................................... 28
      2.3.3. Evaluation Tools ............................................................................................................. 29
      2.3.4. Testbed description ......................................................................................................... 31
      2.3.5. Evaluation Results and their assessment ........................................................................... 32
      2.3.6. Conclusions and next steps ............................................................................................ 38
   2.4. LSP-CLUSTER: Network Traffic Clustering towards Anomaly Detection ............................ 39
      2.4.1. Evaluation Scope ............................................................................................................... 39
      2.4.2. Evaluation Metrics .......................................................................................................... 40
      2.4.3. Evaluation Tools ............................................................................................................. 41
      2.4.4. Testbed description ......................................................................................................... 41
      2.4.5. Evaluation Results and their assessment ........................................................................... 41
2.4.6. Conclusions and next steps ................................................................. 42

3. **Network Resilience Testbeds** ..................................................................... 43

3.1. Performance degradation (Dense Urban Area Testbed) ................................. 43

3.1.1. Evaluation Scope .............................................................................. 43

3.1.2. Evaluation Metrics ........................................................................... 43

3.1.3. Evaluation Tools .............................................................................. 44

3.1.4. Testbed description ........................................................................... 45

3.1.5. Evaluation Results and their assessment ............................................... 46

3.1.6. Conclusions and next steps ............................................................... 51

3.2. SLA ........................................................................................................ 52

3.2.1. Evaluation Scope .............................................................................. 52

3.2.2. Evaluation Metrics ........................................................................... 52

3.2.3. Evaluation Tools .............................................................................. 52

3.2.4. Testbed description ........................................................................... 53

3.2.5. Evaluation Results and their assessment ............................................... 54

3.2.6. Conclusions and next steps ............................................................... 57

4. **Conclusions and Further Work** ................................................................. 59

4.1. D5.4 Conclusions ................................................................................ 59

4.2. WP5 conclusions .................................................................................. 60

**Glossary, Acronyms and Definitions** .......................................................... 64

**References** ............................................................................................... 66
List of Tables

Table 1 Short overview of the deliverable 5.4 ........................................................................................................ 10
Table 2 Example of the DSE Testbed Policy Document ....................................................................................... 16
Table 3 List of attack types supported by the DSE Testbed ................................................................................... 18
Table 4 Evaluation Metrics based on statistical analysis approach ........................................................................ 20
Table 5 AUC score for each algorithm and type of malicious traffic ................................................................. 25
Table 6 LSP-cluster performance on NetClu and NSL-KDD datasets .................................................................. 41
Table 7 Comparison of LSP-cluster to other systems on NLS-KDD dataset ...................................................... 42
Table 8 ADE Prediction Evaluation on the test data ............................................................................................. 48
Table 9 Observation of different metrics for MME1 during Attachment with ADE .............................................. 50
Table 10 Observation of different metrics for MME1 during Attachment without ADE ................................. 51
List of Figures:

Figure 1 List of testbeds and components described in this deliverable .............................................. 9
Figure 2 DSE Testbed perspectives for metric evaluation ........................................................................... 12
Figure 3 SFlow packet format including element description ................................................................. 13
Figure 4 SFlow Data Extract from DSE Testbed ......................................................................................... 13
Figure 5 Population of threshold values during algorithm training ......................................................... 15
Figure 6 Logger entries for the DSE Testbed when attack detected ......................................................... 17
Figure 7 Honeynet case in Mouseworld .................................................................................................. 24
Figure 8 AUC stabilization with size of training ......................................................................................... 26
Figure 9 NFV Security Anomaly Detection Testbed .................................................................................. 31
Figure 10 CPU usage metric of server processing normal and malicious traffic .................................... 33
Figure 11 Traffic throughput in normal state and during an attack ......................................................... 34
Figure 12 Time taken by Server to send HTTP response during normal traffic, attack and blocked attack ......................................................................................................................... 35
Figure 13 UDP throughput received by Server in normal and quarantine zone .................................... 36
Figure 14 - Time taken by system to react after an attack started ......................................................... 37
Figure 15 Performance degradation testbed for Network Resilience ....................................................... 45
Figure 16 ADE applied on MME1 system CPU utilization collected from the Network Resilience testbed ........................................................................................................................................ 47
Figure 17 ADE applied on MME1 system CPU load collected from the Network Resilience testbed ........................................................................................................................................ 48
Figure 18 Comparison of rate of reaction using RTT method ................................................................. 49
Figure 19 Confusion Matrix over 3 SLOs ................................................................................................. 53
Figure 20 Testbed description for SLA .................................................................................................... 54
Figure 21 Comparison between LSTM and FFNN by accuracy over the validation set ....................... 55
Figure 22 ANN initialization method versus accuracy. GN: Glorot Normal, GU: Glorot Uniform, HN: He Normal, HU: He Uniform, LU: Lecun Uniform, N: Normal, O: Orthogonal, U: Uniform, Z: zero ......................................................................................... 55
Figure 23 Accuracy with respect to Activation functions in the output layer. This figure shows that the choice of the activation layer is amongst the most important ones ........................................... 56
Figure 24 Top 5 ANNs based on the mean over validation score accuracy. All the best ANNs appear to converge on 94% average accuracy ................................................................................................. 56
1. Introduction

1.1. Motivation, Objective and Scope

In this deliverable, the same set of testbeds that were presented in D5.2 and D5.3 are assessed, underlining specific aspects which are significant for the usage of the machine learning as means of making highly dynamic and adaptable decisions and their enforcement in the directions of subscriber communication security, network security, network reliability and performance. The list of the testbeds and components described in this deliverable is illustrated in Figure 1.

![Figure 1 List of testbeds and components described in this deliverable](image)

In the experiments, the full cycle of the management operations is measured and assessed including the monitoring, the decision and the actuation.

Please note that this deliverable builds on the content of the previous ones from WP5 as well as on WP3 and partially WP4 developments. Because of this and in order to be able to maintain the reader attention, a strict approach towards not duplicating of the content was taken. Although
most of the sections could be read in an independent fashion, we consider that prior knowledge as available in the other WP5 deliverables would be needed for a comprehensive understanding.

The deliverable includes the following sections:

<table>
<thead>
<tr>
<th>Sections</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Distributed Security</td>
<td>The section includes the evaluation of the DSE as well as functional extensions of the testbed. This represents an update of Section 2.1 of D5.2 and D5.3.</td>
</tr>
<tr>
<td>Enablement</td>
<td></td>
</tr>
<tr>
<td>2.2 Honeynet</td>
<td>The section includes the detection of data flows within the Honeynet testbed described in Section 2.2 from D5.2 and D5.3.</td>
</tr>
<tr>
<td>2.3 NFV Security Anomaly</td>
<td>This section includes the evaluation of the security anomaly detection in distributed firewalls and represents an update of Section 2.3 and Section 2.4 from D5.2 as well as of Section 2.3 from D5.3.</td>
</tr>
<tr>
<td>Detection</td>
<td></td>
</tr>
<tr>
<td>3.1 Resilience testbed</td>
<td>The section includes the evaluation of the testbed described in Section 3 testbed from D5.2 and Section 3.1 from D5.3.</td>
</tr>
<tr>
<td>3.2 Media SLA</td>
<td>This section includes the evaluation of the testbed described in Section 3.2 of D5.3.</td>
</tr>
</tbody>
</table>

Table 1 Short overview of the deliverable 5.4
2. Distributed Security Enablement Testbeds

2.1. Distributed Security Enablement Testbed

2.1.1. Evaluation Scope

In D5.3, we presented that the primary objective of the distributed security enablement testbed is the creation, deployment and maintenance of security based service function chains within the operator’s infrastructure (NFVI). We also described how this architecture can be used as a means to separate the processing of subscriber requests to different security zones where the trust of those requests can be considered differently.

In order to support the generation of normal traffic patterns and the overlay of malicious attack traffic on top a testbed was required. This testbed had to be first constructed to allow for the provisioning of the actors in the system (both the normal & malicious traffic generating nodes) and for the deployment of the service function chains that is used for the routing of the traffic within the operators infrastructure that connects the client applications (running in the public domain and can be regarded as being outside of the tenants network) to the set of operator and/or parallel services (can be regarded as running within the tenant network). In D5.3 this configuration was simulated using Docker where each node (actor) constituted a container and routing between clients and servers was more direct. The migration of this design to the final testbed architecture has allowed for the effort to be applied to the testbed setup as the client/server relationships had already been proven within the simulated Docker environment.

To facilitate the generation of viable metrics the entire testbed has to be considered from multiple perspectives. These following areas have been identified within the distributed security enablement testbed for individual analysis:

1. Traffic generation and overlay of attack data including multiple attack types and vectors.
2. Extraction of metric data to provide traffic analysis.
3. Online analysis of the metric data and ability to learn normal traffic patterns.
4. Lead time for identification of attack of system being monitored from whence attack fist being deployed.
5. Effectiveness of attack identification with suitable metadata.
6. Generation of corrective actions to mitigate identified threat.
7. Actuation of corrections required in the operator’s infrastructure.
8. Elimination of the threat from the network (operator’s infrastructure).
The Figure 2 overlays these perspective points onto the design of the distributed security enablement testbed.

Each of these perspectives will be considered in the following sections where they are discussed within the context of the evaluation to be carried out.

### 2.1.2. Evaluation Metrics

The primary purpose of the distributed security enablement testbed is to provide complete real-time control over the infrastructure fabric such that different operational conditions can be achieved without significant delay or stress. As the nature of the evaluation changes, so too does the dynamics of the testbed thereby facilitating the scenario under test. There are several set of metrics to be considered at this point:

- Traffic Flow Metrics
- Attack Detection Accuracy Temporal Metrics
- Corrective Action Execution Metrics

#### 2.1.2.1 Traffic Flow Metrics

At its most basic form this is the raw metric data that is "gleamed" from the infrastructure traffic and is presented to the CogNet Common Infrastructure for analysis and attack detection. This metric data takes the form of SFlow\(^1\) data format and give a sampled overview of the network

---

\(^1\) [http://sflow.org/sflow_version_5.txt](http://sflow.org/sflow_version_5.txt)
traffic. Currently only SFlow is provided but additional formats can be added (NetFlow\(^2\)) and the metric generator can be extended. When evaluating SFlow data there are two data structures produced: Control data and Flow data. For the purposes of CogNet and the DSE testbed we are concentrating only on the Flow data format.

<table>
<thead>
<tr>
<th>Sflow Packet Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timestamp</td>
<td>The timestamp in milliseconds</td>
</tr>
<tr>
<td>Type of Sample</td>
<td>This is a type of data being recorded</td>
</tr>
<tr>
<td>Agent IP</td>
<td>This is the IP address of the device that sent the data</td>
</tr>
<tr>
<td>Input Port</td>
<td>Inbound Port</td>
</tr>
<tr>
<td>Output Port</td>
<td>Outbound Port</td>
</tr>
<tr>
<td>SRC MAC</td>
<td>Source MAC address</td>
</tr>
<tr>
<td>DST MAC</td>
<td>Destination MAC address</td>
</tr>
<tr>
<td>Ethernet Type</td>
<td>The type of Ethernet</td>
</tr>
<tr>
<td>In VLAN</td>
<td>Inbound VLAN</td>
</tr>
<tr>
<td>Out VLAN</td>
<td>Outbound VLAN</td>
</tr>
<tr>
<td>SCR IP</td>
<td>Source IP</td>
</tr>
<tr>
<td>DST IP</td>
<td>Destination IP</td>
</tr>
<tr>
<td>IP Protocol</td>
<td>The IP Protocol</td>
</tr>
<tr>
<td>IP TOS</td>
<td>The IP type of Service</td>
</tr>
<tr>
<td>IP TTL</td>
<td>The IP Time to Live</td>
</tr>
<tr>
<td>UDP SRC OR TCP SRC PORT ICMP TYPE</td>
<td>N/A</td>
</tr>
<tr>
<td>UDP DST OR TCP DST PORT OR ICMP CODE</td>
<td>N/A</td>
</tr>
<tr>
<td>TCP Flag</td>
<td>The TCP flag value</td>
</tr>
<tr>
<td>Packet size</td>
<td>The size of the packet</td>
</tr>
<tr>
<td>IP Size</td>
<td>The size of the IP header</td>
</tr>
<tr>
<td>Sampling Rate</td>
<td>The rate of the Sflow samples</td>
</tr>
</tbody>
</table>

For the DSE Testbed the sampling rate applied to the SFlow generator is configurable and is set to sixty-four meaning that the analysis is not overwhelmed with every packet traversing the network and this gives enough visibility into the nature of the traffic.

Figure 3 SFlow packet format including element description

Figure 4 SFlow Data Extract from DSE Testbed

2.1.2.2 Attack Detection Accuracy Temporal Metrics

As traffic metric data is handed off to the CogNet Smart Engine the SFlow data is transformed into a useable format and the appropriate features are extracted for each type of attack to be detected. As stated in D5.3 each attack type will have a dedicated machine learning Docker container provisioned within the CogNet Common Infrastructure and its sole responsibility is for the detection of its particular attack type. For the detection to be accurate the algorithm must first be trained so that it can learn what traffic patterns are “normal” within this network. Typically, the algorithm trains for up to one hour which accounts for over 3,000,000 SFlow packets. The training period will allow for the CogNet smart engine (and the algorithm) to generate normal operating condition thresholds that are representative of the network activity. Once training has been completed the algorithm switches from training mode into attack detection mode and thresholds go live. It is possible to modify the training period and initial testing has shown that the threshold percentages are within %10 of final values after only twenty minutes of training (of course this is dependent upon the activity of the network). Figure 5 displays the threshold values as the algorithm is undergoing training.

The algorithm is designed to run with batches of SFlow data every fifteen seconds, that is, the SFlow processed data is accumulated into batches each of fifteen seconds capacity. The batch is collated and prepared for analysis based on the source IP Addresses and Ports as well as the destination IP Address and ports. Once the batch has been sorted the batch results are compared against the training thresholds. A slope factor of 1.5 is applied to the thresholds results and if the batch results are within the applied slope then the batch is considered to be acceptable and no attack has been detected. The dynamic slope of 1.5 times the threshold is to allow for variations in the network patterns that can accommodate variable traffic patterns but will not filter out any undesirable patterns that could be an attack underway. Using this method, it is possible to detect an attack within the network between fifteen to thirty seconds.
2.1.2.3 Corrective Action Execution Metrics

When the CogNet smart engine detects an attack the output from the algorithm is captured in a policy document along with all of the metadata required to correctly identify the offending node in the network including the nature of the attack type detected. The Table 2 displays an example of the policy document that is generated when an attack is detected. This policy document is placed on a Kafka queue for consumption by the Policy Manager. The policy manager scans the policy document and based on the metadata within the document it will select the appropriate corrective action to execute. The corrective actions vary between applications and attack types detected but the simplest form of actuation is to remove the offending node form the network. The policy document also contains the endpoint to the Distributed Security Enablement Application (DSE-App) which is an application that takes the role of the OSS but is automatically triggered by the output of the policy manager instead of requiring human intervention.

```json
//TrainingData.json
"trainingData": {
  "sflow_type": {
    "FLOW": 30692
  },
  "agent_address": {
    "10.108.0.4": 30692
  },
  "Input_port": {
    "45": 2,
    "47": 30666,
    "49": 462
  },
  "output_port": {
    "45": 2,
    "47": 462,
    "49": 30667
  },
  "src_mac": {
    "eaae76f999ad": 30666,
    "Gabbe6f966c": 462,
    "56f34494197a": 2
  },
  "dst_mac": {
    "eaae76f999ad": 462,
    "Gabbe6f966c": 30667,
    "56f34494197a": 2
  },
  "ethernet_type": {
    "0x0000": 30692,
    "0x0000": 0
  },
  "In_vlan": {
    "0": 30692
  },
  "out_vlan": {
    "0": 30692
  },
  "src_ip": {
    "192.108.1.1": 30666,
    "192.108.1.1": 1
  }
},
"supa-policy": {
  "supa-policy-validity-period": {
    "start": "2017-10-01T09:15:57.527404Z"
  },
  "supa-policy-target": {
    "domainName": "systemOpNFV",
    "subNetwork": "192.168.1.1",
    "instanceName": "wit",
    "topicName": "dse-firewall"
  },
  "supa-policy-statement": {
    "event":
```

Figure 5 Population of threshold values during algorithm training
Table 2 Example of the DSE Testbed Policy Document

The DSE–App can be deployed within the CogNet common infrastructure but the logical location for this application is adjacent to the tenant’s network as the set of corrective actions may require API calls that can only be executed within the privacy of the tenant’s infrastructure. The DSE–App is agnostic to the traffic and nodes operating within the network under management but when commands for actuation are received from the policy manager the DSE–App will then exploit its influence over the VNFi to perform the set of corrective actions as requested by the policy manager. As described earlier there are many corrections actions that can be executed but one of the simplest is the exclusion of the offending nodes from the network, this can be achieved by drafting a new Access Control List rule that specifically targets the malicious/malfunctioning node and pushing that rule to OpenDaylight – the Software Defined Networking Controller deployed to the DSE testbed. From initial testing the time taken from attack detection to rule deployment and enforcement of the new rule within the gateway to the tenant’s network has been in the matter of seconds. The architecture of the DSE testbed has the advantage of having and almost real-time reactionary response which is one of the major goals of CogNet. In tests to date the actuation response to attack detection has always been less than thirty seconds. The Figure 6 displays the...
logger entries that are generated when an attack is detected by the CSE and the policy manager execute the corrections actions from the policy.

![LogAnalyzer](image)

**Figure 6 Logger entries for the DSE Testbed when attack detected**

### 2.1.3. Evaluation Tools

The set of evaluation tools used within the DSE testbed broadly falls within two main areas: generation and analysis. Although these were originally specified and defined with D5.3 and D5.3 they are now an integral component within the final version of the DSE testbed and are utilising the full potential of the VNFi.

Each of the traffic generating nodes utilises a combination of HTTP verbs (for managing the transfer of data and files across the network) and ICMP pings. The nodes have scripts included as part of their deployment and will continuously execute those actions so traffic is constantly being created. The attack nodes are utilising the hping3 utility that is part of the Kali4 test server distribution and this application is configured within the different attack nodes in order to generate the different attack types. Once activated, each node will run continuously until they are disabled thereby providing the ability to vary the attack types and durations. The list of attack types supported by the DSE Testbed is captured in Table 3.

---


4 [https://www.kali.org/](https://www.kali.org/)
<table>
<thead>
<tr>
<th>Attack Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syn Flood</td>
<td>The attack node is targeting other nodes in the network by sending as many TCP connection request packets to random ports as fast as possible.</td>
</tr>
<tr>
<td>ICMP Flood</td>
<td>The attack node is targeting other nodes in the network by sending as many ICMP packets as fast as possible.</td>
</tr>
<tr>
<td>Vertical Syn Flood</td>
<td>The attack node is targeting another node in the network by sending packets to every port on that node as fast as possible.</td>
</tr>
</tbody>
</table>

Table 3 List of attack types supported by the DSE Testbed

The analysis of the network is carried out within the CogNet Smart Engine which is an integral part of the CogNet common infrastructure. The set of security anomaly detection algorithms will monitor and act upon traffic pattern anomalies detected in the tenant's data plane which when not addressed in a timely manner the efficiency of the service providers' network will suffer degradation and the performance will be affected. The goal of the DSE Testbed demo is to remove the supported types of security threats from the traffic within the network and thereby providing the allocated resources to the tenant for the processing of legitimate traffic within that network.

A set of algorithms have been designed and deployed within the CSE that are based upon simple statistical analysis of the traffic patterns and determining if an anomaly occurs. Although this method proved effective it didn't provide the robustness that can be achieved with machine learning approaches such as a constant redefinition of what is normal within the network. Initial machine learning development work for the attack detection was carried out using decision trees. The initial results looked good and when the worked moved towards the gradient boosted areas for decision trees we can see the results have significant improvements. The initial results have shown a prediction accuracy of greater than %99 and the work of integrating the xgboost\(^5\) model into the CSE for the DSE testbed will continue.

2.1.4. Testbed description

The initial set of evaluation tools were developed using a set of Docker contains for each node but have since been migrated into the final version of the VNFi that is the basis of the DSE testbed, in this case the VNFi is OpNFV\(^6\) and the OpenStack\(^7\) deployment as part of OpNFV. Also part of the OpNFV platform is OpenDaylight\(^8\), the SDN controller that is being utilised to configure and update the set of service function chains being deployed within the VNFi. Once the chains have been deployed and configured within the VNFi the subscriber nodes and the tenant service servers are deployed at either side of the chains. The configuration of the VNFi will guarantee that all traffic originating from the subscribers will have to traverse the service function chains in order to reach the tenant’s services and receive a response. One of the links in all of the deployed chains will always be a network traffic probe that will generate the SFlow metric data for analysis by the CogNet smart engine. The Figure 2 displays the DSE tested and also captures which aspects are specific to the DSE testbed and which are located in the CogNet common infrastructure.

The actuation within the VNFi is a vital component within the DSE testbed. This corrective action closes to loop on the scenario and provides the visual proof that the attack has not only been identified but has been dealt with in an appropriate manner without human intervention. As the flow of the scenario moves from the CSE (attack detection) to the policy manager the nature of the activity changes from detection to corrective action selection. Once decided the policy manager will call APIs exposed by the DSE–App that has intimate knowledge of the VNFi and the services available within it (the service function chains). The corrective actions from the policy manager are translated into MANO type actuations within the DSE-App that can, for example, modify the topology of the VNFi and/or reconfigure the existing configuration in the VNFi. Once the corrective action has been executed the results should be apparent immediately, more than likely the malicious/malfunctioning node is either removed from the network or is re-routed via a different service path (another chain) for further analysis (analysis such as deep packet inspection or application behavioural analysis).

As stated earlier the DSE testbed is dynamic by design and can facilitate the addition of new attack types for both simulations within the VNFi and detection in the CSE. The facility provides users the ability to concentrate on the development of attack detection technologies without the overhead of maintaining or deployment of the testbed. Further details about the DSE testbed are available in the integration work package deliverables D6.3.

2.1.5. Evaluation Results and their assessment

The evaluation of the results consists of two main areas; evaluation of the results for each of the supported attack types within the current statistical analysis approach and the evaluation of the results where the machine learning algorithms have been deployed. As the work is continuing for

---

\(^6\) https://www.opnfv.org/
\(^7\) https://www.openstack.org/
\(^8\) https://www.opendaylight.org/
the integration of the machine learning algorithms into the DSE testbed it is only possible to report on the results from the statistical analysis approach.

<table>
<thead>
<tr>
<th>Evaluation Area</th>
<th>Delays</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traffic flow metrics</td>
<td>&lt; 5 seconds</td>
<td>Every 64th packet generates an SFlow packet and is sent directly to the CSE for analysis via the Kafka queue. With traffic on the network there is no delay so the metric data is ready instantly.</td>
</tr>
<tr>
<td>Attack Detection Accuracy Temporal</td>
<td>15 &lt; detection &lt; 30 seconds</td>
<td>As SFlow data is received it is processed and batched into blocks of 15 seconds and processed once the time threshold is reached. Max delay is &lt; 30 seconds for attack detection.</td>
</tr>
<tr>
<td>Corrective Action Execution Metrics</td>
<td>&lt; 30 seconds</td>
<td>Once attack is detected the policy document is drafted and sent to the Policy Manager for processing via another Kafka queue. The policy manager is continuously polling the queue and retrieves messages almost instantaneously. The conditions are evaluated and the corrections action(s) are executed, in this case the API calls in to the DSE-App. Max delay: processing time of the policy document + API calls into the DSE-App + DSE-App MANO activities including rules generation and deployment &lt; 30 seconds for corrective action execution.</td>
</tr>
</tbody>
</table>

Table 4 Evaluation Metrics based on statistical analysis approach

These values are all valid for the set of implemented attack types to date which are captured in Table 3. As more attack types are added and the method of analysis is migrated from the statistical analysis method to a machine learning approach the set of evaluation metrics will be expanded based on the accuracy, false positive rates, precision and sensitivity or the algorithms.

2.1.6. Conclusions and next steps

The results attained so far have shown that it is possible to utilize statistical analysis and machine learning to monitor a sample of network traffic in order to determine if there is some malicious
intent in that traffic and if so to take corrective actions against the originating sources. As attacks are detected the corrective actions can vary from the brute force type (where we simply remove the offending node(s) from accessing the network via access control lists and firewall rules) to the complex (where malicious traffic can be quarantined for further analysis). By isolating rouge traffic it is possible to determine the nature of the traffic and perhaps generate an attack signature that can be used to further enhance the set of algorithms for a more rapid detection.

The expansion of the DSE testbed itself where the strategy of the online training can be enhanced to provide training in parallel to live detection where new and updated training thresholds are generated based on the latest set of input SFlow packets and the assurance that an attack hasn’t been detected thereby validating the new trained thresholds. This would cover the ability to scale the subscriber base without triggering false positives.

As additional attack types are identified the CSE can be scaled out to include those new detection algorithms into the testbed. This coupled with the ability of the DSE Tested to deal with a vast array of data points and the pace of detection provides significant reassurances about the resilience of the network, consistency of the tenant’s network performance and the seamless experience for the subscribers.
2.2. Honeynet Testbed

2.2.1. Evaluation Scope

The initial scope defined in the deliverable D5.2 explained the identification and classification of some security attack patterns in the data plane, especially relating to encrypted traffic, or what is similar, inspecting data packets from Layer 2 to layer 4, avoiding payload analysis (encrypted or not), and thus, improving the privacy. Interim results shown in deliverable D5.3 included an expansion of the ML experiments with new datasets from specific honeynet attacks, some malware samples and illegal traffic.

The last stage activity and the following results focus the experiments with the integration of new malware samples and commercial traffic generator (Breaking Point) that improve datasets realism, through customization of the Mouseworld environment. The evaluation scope for the testbed has included the test and evaluation of different ML algorithms with promising results in this type of network threats. Three final ML algorithms candidates were evaluated, in the semi-supervised and unsupervised anomaly detection category:

- iForest (Isolation Forest)[1]:
- LOF (Local Outlier Factor)[2]
- OCSVM (One Class Support Vector Machine)[4]

2.2.2. Evaluation Metrics

Validation process was based on fresh datasets injection. Cross validation test has been done, but the final evaluations has been made using new dataset benefiting from Mouseworld generation capacity and replicability.

Some valuable metrics used for evaluation includes: precision, recall, f1 score or accuracy, but in the case of unsupervised anomaly detection researched here, the nature and the volume of the datasets recommend other metric to be used. It is common accepted that, in large datasets, identify all anomalies between the first ones in a long list, is more valuable than identify only some anomalies in a short list. This is very relevant in our area of research, where a Network operator can focus and investigate the first detected abnormal flows, knowing that all anomalies are there, despite the fact that there could be some false anomalies. This is seized using the True positive rate, versus false positive rate represented by the Area Under the Receiver Operating Characteristic Curve (ROC AUC). The AUC represent the probability that an anomaly detection algorithm will assign a randomly chosen normal instance a lower score than a randomly chosen anomalous instance [2].

2.2.3. Evaluation Tools

The tools used in this testbed can be categorized in two areas: the traffic generation tools, and the evaluation tools. The traffic generation includes several tools already introduces in D5.2 and D5.3 to generate traffic. Here are the ones used in this phase:
- Commercial Security traffic generator. IXIA Breaking Point [5]. This traffic generator allows for the production of synthetic traffic related to applications and network attacks (DDoS, malware, etc.). Its high versatility allows it to execute two main tasks. First one is what we call "white noise" or normal traffic, i.e. Service provider mix traffic: https, ssh, email, p2p, etc., to be used as the ISP normal traffic. Second one is specific malware or attacks, such as DDoS, ransomware communications, ssh attacks that should be detected by the ML algorithms.

- TCPReplay tool and malicious samples. Combining both we were able to inject in the Mouseworld lab different type of malware and malicious traffic.

The evaluation tools used are based on scikit-learnt python tool [6]. The metric library between others, allows us to evaluate the different metrics mentioned in Section 2.2.2, and specifically the AUC score.

It is important to highlight that being a realistic traffic in a controlled environment, the traffic can be precisely classified for this evaluation process. The timestamp and the source IP address is being used as the labels to identify the anomalies in the evaluation scripts. Next example shows the evaluation process used to test a trained ML algorithm with a capture dataset from Mouseworld where a malware sample has been introduced with service provider normal traffic:

```bash
$ python ./security_test.py -i dataset_26_01_malware.csv -m model.plk -a ips.txt -d dates.txt
Input dataset: dataset_26_01_malware.csv
Output train: model.plk
IPs: ips.txt
Dates: dates.txt
('Initial dataset size', (60882, 10))
('Processed dataset size', (28538, 10))
Results exported correctly
('TPR (Recall):', 1.0)
('ACC (Accuracy):', 0.9922909804471232)
('TNR (Specificity):', 0.9922871967465994)
('AUC :', 0.9961435983732997)
```

where the parameters are:

- `dataset_26_01_malware.csv`: dataset captured from Mouseworld with malware sample.
- `model.plk`: trained ML algorithm to be loaded in the evaluation test
- `ips.txt`: list of malware source IP address
- `dates.txt`: time windows of the start and end time of the malware
This example shows that with a dataset of 60682 flows (around 9 hours) we obtain a AUC score of 0.996. Detailed analysis of the output file (results.csv) showed that the algorithm detected around 234 anomalies flows, including the real ones (14 flows).

The complete set of script evaluation has been published in the CogNet Github repository: https://github.com/CogNet-5GPPP/WP5-CSE-Final/tree/master/MouseWorld/Evaluation

2.2.4. Testbed description

The Honeynet testbed was introduced in D5.2 and D5.3 as one of the potential applicability of NFV into Networks threats detection fostered by Machine Learning technologies. The scope of this testbed, as a subset of the distributed security enablement, has the aim of being able to generate useful datasets, as close as possible to the reality, without compromising privacy and test different types of algorithms in the detection based only on the network traffic flows. The Figure 7 shows the testbed implemented based on TID’s Mouseworld Lab.

![Figure 7 Honeynet case in Mouseworld](image)

2.2.5. Evaluation Results and their assessment

Two main evaluations have been made. The first is based on different ML algorithms performance with different security types of malicious traffic to choose the better one. The second evaluation was related to the performance results over the winner algorithm if we change the training dataset size (traffic capture size during training).

Malicious traffic performance evaluation

Three families of Malware were selected to evaluate the ML algorithms:
- Malware Family #1: RIG EK. RIG *Exploitation Kit*, used in 3 malware implementations: Bunitu [7], Dreambot [8] and Chthonic [9].

- Malware Family #2: Jaff Ransomware, used in 3 malware extensions: .zip [10], pdf and word [11].

- Malware Family #3: Wannacry Ransomware (eternalblue) [12].

Also, additional types of malicious traffic were included:

- Illegal traffic #1: Cardsharing protocols for illegal decryption of payTV. Two different families were used: CCCAM protocol and Engel IKS protocol.

- Illegal traffic #2: SSH brute force attack. As it was defined in D5.3

The Table 5 AUC score for each algorithm and type of malicious traffic shows the different results with the 3 types of ML algorithm. The contamination is fixed to 1% (number of estimated outliers or anomalies). Other contamination factors were used (5% and 10%) with no improvement in the numbers.

<table>
<thead>
<tr>
<th>Type</th>
<th>OCSVM</th>
<th>iForest</th>
<th>LOF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Malware Family #1: Bunitu</td>
<td>0.99539</td>
<td>0.55758</td>
<td>0.495036</td>
</tr>
<tr>
<td>Malware Family #1: Dreambot</td>
<td>0.995070385</td>
<td>0.795044024</td>
<td>0.495004481</td>
</tr>
<tr>
<td>Malware Family #1: Chthonic</td>
<td>0.995457486</td>
<td>0.553876638</td>
<td>0.494999191</td>
</tr>
<tr>
<td>Malware Family #2: jaff(zip)</td>
<td>0.995032582</td>
<td>0.495005875</td>
<td>0.828357013</td>
</tr>
<tr>
<td>Malware Family #2: jaff(pdf)</td>
<td>0.995023936</td>
<td>0.745013437</td>
<td>0.745013437</td>
</tr>
<tr>
<td>Malware Family #2: jaff(doc)</td>
<td>0.995015084</td>
<td>0.745009192</td>
<td>0.4950033</td>
</tr>
<tr>
<td>Malware Family #3: Wannacry</td>
<td>0.996032799</td>
<td>0.494978777</td>
<td>0.494997877</td>
</tr>
<tr>
<td>Illegal traffic #1: CCCAM</td>
<td>0.997997305</td>
<td>0.961675709</td>
<td>0.689670121</td>
</tr>
<tr>
<td>Illegal traffic #1: IKS</td>
<td>0.99854683</td>
<td>0.875496109</td>
<td>0.507710883</td>
</tr>
<tr>
<td>Illegal traffic #2: SSH</td>
<td>0.500008028</td>
<td>0.520044937</td>
<td>0.4949988</td>
</tr>
</tbody>
</table>

Table 5 AUC score for each algorithm and type of malicious traffic

The results clearly show that One-Class SVM has a better performance than the others algorithms. The unique exception is the SSH brute force attack, where the detection capacity is clearly inadequate in any of them.

Training dataset size evaluation

This evaluation phase fixes the One-Class SVM algorithm and evaluate the performance of the algorithm when we modify the available dataset or flows size during the training phase. This evaluation has the aim to clarify how much training is needed to consider a specific algorithm trained.

The Figure 8 shows the evolution of the AUC with different dataset size.
The Figure 8 show that AUC stabilizes with the size of the training, and from 40K flows the AUC keep constant with most of types of malicious traffic.

### 2.2.6 Conclusions and next steps

We have been able to train and test multiple malware families and other type of malicious traffic with positives results. These experiments demonstrate that the netflow protocol, with adequate pre-processing (features selection) and normalization, allow to identify with good confidence several malicious behavioural.

On the other hand, the precision of these algorithms are not so good by themselves to implement in production in stand-alone mode. But the knowledge generated can be combined with other source of information, to increase the precision. Some examples are reputation IP or domain list, or supervised algorithms.

In this respect TID has started collaborations and activities in:

- Collaborate with other business units, such as 11Path, Telefonica Cybersecurity company, to increase the experiments and trials in real environment to integration in cybersecurity existing tools.
- Work in progress with legal department in TdE around card sharing potential trials to identify the impact of this traffic in networks.
- Bring these results and evolve the algorithms in several research projects. One potential project is SHIELD-H2020 project [13] where TID expect to enhance these algorithms with
other source of information. Also, several EIT Digital Infrastructure action proposals are open and TID foresee the transfer of Mouseworld and Network threats algorithms developed here.
2.3. NFV Security Anomaly Detection Testbed

2.3.1. Evaluation Scope

This testbed is dedicated to show the possibilities provided by Software Defined Networking (SDN) based data center networking for network security, concentrating on the deployment of SDN components as substrate of the cloud infrastructure, as well as to use the substrate to monitor and to make policy actuation in regard to security threats.

This evaluation of this testbed showcases the advantage of using machine learning for security purposes within the data center substrate level.

The testbed includes a virtualized datacenter with a modified OpenStack instance that integrates the SDN Controller and Switch of Fraunhofer OpenSDNCore into the network layer of OpenStack. The SDN control plane has a central view of the network and uses the OpenFlow protocol to communicate with the data plane devices. The central view gives us the opportunity to make automated decisions based on network level, distributed data, and to have complete control over the method and place of actuation over the whole network which is a key advantage compared to node by node actuation. Through the OpenFlow protocol we can push customized flow rules as reaction to activated policies and we can selectively monitor network traffic with monitoring actions added on flow rule level. A full description of the testbed is available in D5.3.

Monitoring can be easily customized and optimized as it is implemented fully in software, as a module that can be run on top of the OFS Software Switch. It is a simple task to add new monitored traffic flow features or remove features that, with time, turn out not to be useful for anomaly and attack detection purposes. This monitoring module logs the flows live with a configurable logging period, which is a key advantage over tools that log only when the monitoring is stopped, since this live monitoring enables immediate reaction to detected threats.

In the following points several metrics are shown to prove the benefits of automated policy actuation when used to protect network or compute resources.

2.3.2. Evaluation Metrics

The evaluated metrics are dedicated to show, that we can protect compute and network resources with the automatically activated machine learning triggered policies, and that the reaction time of the system is much faster than what we could expect from local, firewall like, policy actuation.

We measured the following metrics:

1. CPU usage of a server located in the datacenter. By monitoring the CPU activity on a server during normal activity, under attack and with the attack blocked at the switch, we can indicate how our policy actuation based on machine learning can protect compute resources in the datacenter showcasing that the proposed solution is reducing the effect of attacks on the active components.

2. Traffic throughput reaching a server. By measuring the amount of traffic reaching a server during normal activity, under attack and with the attack blocked at the switch, we can
indicate how the policy actuation can result in the protection of internal network resources, when the blocking rules are applied at the attack entry point.

3. Traffic throughput reaching a server and its quarantine counterpart. By measuring the amount of traffic reaching a server and its quarantine counterpart during normal activity, under attack and with the attack redirected at the switch from the default zone to the quarantine zone, we can show the possibility to send suspicious traffic to a dedicated server for further inspection while not disturbing the flow of normal traffic to the default server. By these means, the data traffic does not have to be dropped immediately when a possibility to be malicious is detected.

4. Response time of a web server. By measuring the time, it takes for a web server to answer to a page request during normal activity, under attack and with the attack blocked at the switch, we can indicate how the machine learning based policy actuation can help maintain quality of user experience by protecting the server from malicious traffic.

5. Reaction speed of the system. By measuring the time, it takes for the system to react after an attack is started, we can show the ability to accelerate detection and actuation compared to previous, non-machine learning based solutions.

As the complexity of the experimentation was high and in order to avoid side effects in the measurements and to properly control the experiments, the metrics from one to four were monitored on a statically basis, as they are not directly impacted by the automation of the attack. For the fifth metric, all measurements were executed automatically.

**2.3.3. Evaluation Tools**

The following are the evaluation tools:

1. The testbed is built upon the integration of Fraunhofer OpenSDNCore into OpenStack, utilizing the concepts behind SDN to control network traffic in the datacenter. The setup is described in detail in Section 2.3.4.

2. For the CPU usage test one UDP and one TCP Iperf server was started on the server VM hosted in OpenStack. The Iperf servers are quite CPU intensive and are hence good candidates for this test. UDP and TCP traffic was started from a machine outside of OpenStack and the UDP traffic was declared “malicious” and blocked after a certain time with a manually applied block rule. The results can be seen on Figure 10 in Section 2.3.5.

3. The throughput test was done using TCP Iperf between the server VM and an outside host. TCP Iperf produces the maximal possible traffic throughput between the client and the server, which makes it a perfect tool for this test. In this test the attack was a SYN Flood generated by the traffic generator implemented by Fraunhofer. The Flood traffic was blocked after a while with a manually added block rule. The results can be seen on Figure 11 in Section 2.3.5.
4. In the third test the response times were measured using a simple script sending HTTP requests periodically to the server VM from an outside host. A time stamp was taken before sending the request and after receiving the response. The difference between these two gave a good approximation of the actual response time. The results can be seen on Figure 12 in Section 2.3.5.

Traffic redirection was shown starting multiple Iperf UDP flows between the server and two different outside hosts. After some time, the traffic from one of the hosts was declared “suspicious” and redirected to a second server residing in the quarantine zone. We measured the throughput the two servers received, while the redirection rule was applied and when not. The results can be seen on Figure 13 in Section 2.3.5.

5. System reaction speed was tested by adding a threat detection policy with a threshold that applies a block rule automatically when the threshold, the number of UDP flows started from the same source, is passed. With the threshold set to 12, we started 13 UDP flows from an outside host using Iperf. On the server, TCP dump was used to see the timestamps of the received UDP packets that were part of the Iperf flows. TCP dump receives packets from the start of the flows until the block rule is applied. Hence the time difference between the first and last received packet gave a good approximation of the actual system reaction time. The results can be seen on Figure 14 in Section 2.3.5.
2.3.4 Testbed description

Figure 9 NFV Security Anomaly Detection Testbed

- **Overview**: The testbed as shown in Figure 9 is built upon the Fraunhofer OpenSDNCore integrated into OpenStack. OpenStack networking is handled by the OpenSDNCore switches (OFS) and controller (OFC). The controller resides in the control node and is responsible for all of the switches. A new controller module provides the interface to push access control policies as allow, block and redirect rules to the switches. The switch performs real-time monitoring of network traffic flows and stores the flow information in a database residing outside of OpenStack. The data is handled by the attack and anomaly detection modules and the policy engine decides which policy to apply.

- **OpenStack Integration**: For the purpose of better control and enablement of dedicated, customised functionalities, an integration of OpenSDNCore into OpenStack was initiated. This means handling the OpenStack networking functions using Fraunhofer OFS Switch and OFC Controller instead of the several bridges OpenStack uses originally. The network architecture is greatly simplified by using a single switch per node. Dedicated controller modules and an OFS Agent are responsible to control the switches and act on the events of OpenStack.

- **Switch, Monitoring and OpenFlow**: OFS is an OpenFlow enabled software switch. OpenFlow is the communication protocol between the controller and OFS defining messages, actions, tables, flow rule matching and the packet processing pipeline. It provides experimenter extension possibilities, which are used in the testbed to provide functionality not defined by the protocol standard.

  Hence OFS can handle traffic forwarding or routing depending on the controller settings. It is also able to selectively monitor the forwarded traffic, which makes it possible to monitor only
allowed traffic and not blocked or redirected flows. This monitoring can be further improved and customised for different use cases.

- **Controller Modules and Rules:** A new OFC module translates ACL policies into flow rules that are pushed to the switch. Rules can allow traffic from source hosts or sub-networks to one of the services residing in the data centre, block traffic from certain sources targeting a service or redirect traffic to a counterpart of the service VM placed in a different zone. Header matching is done mainly on the IP, protocol and port fields of the packets to make forwarding decisions.

- **Detection and Policy Engine:** Detection and policy selection in the testbed is static threshold based, concentrating on the effect and speed of actuation instead of the machine learning aspects. However, full integration into the CogNet Common Infrastructure allows future research regarding the use of machine learning for network attack and anomaly detection as well as policy selection. The data collected during monitoring the network traffic is stored and available for modules in the CSE for processing using machine learning. Flow classification and clustering algorithms combined with dynamic statistical modelling, and time series processing are promising candidates to be used for attack and anomaly detection. More sophisticated detection algorithms enable the creation of more complex events towards the Policy Engine, which allow further research in automated policy selection and policy creation.

### 2.3.5. Evaluation Results and their assessment

In this section, the evaluation results are presented followed by their assessment. In order to obtain these results a set of experiments were executed in laboratory conditions. Albeit a very large number of measurements were executed, the results are presented in a simplified, single measurement form, as to enable the reader to easily comprehend the advantages of the proposed solution.
The Figure 10 shows the CPU usage of a server processing "normal" and "malicious" traffic. When the malicious traffic is blocked by the Switch/firewall the CPU usage noticeably decreases to the level experienced with only normal traffic to an average of about seven percent, and when the block rule is removed it jumps back up to the level with an average of about 15 percent. This shows the potential to save computing resources by blocking malicious traffic targeting CPU intensive applications.

The percentages obtained are highly dependent on the type of service, on the scaling level of the infrastructure and on the type of attack. To prove the advantage of a security solution protection at the data center level for a specific service, it should be deployed in the same setup with that specific service. Otherwise, the measurements here presented address a specific case which cannot be easily generalized.

This evaluation result was highly predictable, however being necessary to prove the advantage of the protection of individual network functions against DDoS types of attacks which may be initiated by externals or by malicious software network functions within the same system.

The Figure 11 shows the traffic throughput reaching the Server in normal state and when experiencing an attack. When there is no attack, the throughput is around 800 Mbits/sec (first high period). When the TCP SYN Flood attack is started, the throughput decreases to about 40 Mbits/sec (first low period). Adding a block rule, blocking the subnet where the attack originates from, results in the throughput getting back to close to the original level, and removing the block rule decreases the throughput again (from second high period). This shows the potential to save network resources by blocking high volume of malicious traffic at its entry point to the controlled network.
The Figure 12 shows the time it takes for the web Server to send the HTTP response after receiving a request during normal traffic, attack and blocked attack. The first low period is the response time without attack (3-5 ms). The peaks show the response times during attacks, when the malicious traffic is not blocked. It is well noticeable that the average response time increases significantly (to about 800 ms), having peaks reaching a few seconds. After adding the block rule for malicious traffic, the response time falls back to the original level. This shows the potential to improve user experience by blocking malicious traffic targeting applications.

**Figure 11 Traffic throughput in normal state and during an attack**
The Figure 13 shows the UDP throughput received by a server in the “normal zone” and its counterpart in the quarantine zone. Two UDP Iperf flows were started from two different outside hosts, one “normal” flow with 80 Mbits/sec throughput and one “suspicious” flow with 420 Mbits/sec throughput. At certain points in time, a redirection rule was added and later removed that redirected part of the traffic to the quarantine zone. After applying the redirection rule, the suspicious traffic arrives to the quarantine machine, while normal traffic continues to reach the normal server. This shows the possibility of redirecting suspicious traffic to a quarantine zone, where it can be further inspected to decide if it is malicious or normal or find out more about the origin and other details.
In the previous examples the blocking and redirection rule was added manually to show the difference between normal, attacked and blocked attack periods. However, the actuation can be done autonomously. The Figure 14 shows the average time it takes for the system to react after an attack is started. The monitoring and the detection modules do reporting and detection periodically, and this period can be configured. The reaction time depicted by the figure includes the following steps after the attack starts:

1. One monitoring period (tick) finishes and the monitored live data is sent to the database
2. One detection period (tick) finishes and the new live data is polled from the database
3. The detection module processes the data using machine learning models, statistical models or static thresholds and fires events if an attack is detected
4. The policy engine receives the event and activates the appropriate policy
5. New flow rules are pushed to the Switch according to the activated policy to block or redirect the traffic

Figure 14 shows the time these five steps take, from the moment when the attack starts, until the moment when the new rules are applied on the switch. The time is shown as a function of the length of the monitoring and detection periods, which were set to the same value in the four cases, to half a second in the first test, one second in the second test, one and a half seconds in the third test and two seconds in the fourth test.

It can be seen on Figure 14, that the average time it takes for the system to react is nearly proportional to the length of reporting/detection period (tick), with this proving that such a system is more dependent on the monitoring granularity than on the detection and mitigation duration.
Overall a time of 1.4 seconds is considered acceptable for the most of the currently available cloud deployed services, this reaction time being faster than the ones of the current firewall solutions. However, for telecom software network functions, it would be necessary to have a smaller monitoring window or an event based system from the monitored components in order to compensate the long window and ultimately to be able to bring the monitored events in real-time to the decision point. With this solution, there would be a larger resource consumption from the monitoring, as a larger amount of data will be gathered, however reducing the overall response time to around 0.4 seconds.

We have to mention in several points that suggest that further research and optimization is necessary, especially because the current implementation is testbed based and does not include neither the carrier-grade features, nor the processing optimization expected from a live system.

Inspecting the headers of each packet and collecting timestamps for each packet increases the time the packet processing threads of the switch spend on one packet and hence decrease the maximal throughput the switch can handle. This calls for further improvement of the monitoring method.

When the database, used to store the monitored flows, is not on the same physical machine as the switch sending the information, sending the data itself consumes network resources, which becomes a significant overhead, when dealing with a large amount of flows.

The reaction of system is still not instantaneous, taking several hundred milliseconds at best, since setting the monitor reporting period lower than this would further reduce performance and increase network resource consumption as explained above.

![Figure 14 - Time taken by system to react after an attack started](image-url)
2.3.6 Conclusions and next steps

The following conclusions can be derived:

- The tests performed in this testbed show the benefits of automated actuation in data centers networking. We have complete control over the network layer of the data center using easily extensible software control and monitoring modules. We could clearly show how network and compute resources can be saved and how we can redirect traffic for deeper inspection. These results allow us to state that we have successfully shown the potential impact of centrally controlled software based firewalls and that the implemented functionalities should be at least partially included in the feature set of future software firewalls.

- On the other hand, there are still some weak points in the system, needing further research and improvement. There are also several potential use cases that could not be completely examined in the scope of this project such as intrusion detection or policy selection using complex machine learning as explained in Section 2.3.4.

- Further research is needed to find the most useful flow features for attack and anomaly detection. Decreasing the amount of monitored flow features will result in better switch performance, since the number of counters and necessary header comparisons decrease. It will also decrease the network usage overhead introduced by the reports of monitoring, since the length of each reported flow entry decreases.

- We did not explore the potential behind SDN providing a complete network view. If selective monitoring would be done on many network devices and the centrally collected distributed data would be used for higher level detection and decision making, based on advanced machine learning algorithms, it would provide an opportunity to discover attacks and anomalies that single node detection cannot.

- We also did not explore the case when the data center hosts services of multiple tenants that might internally cause unexpected or malicious network behaviour or perform attacks inside the network. These type of anomalies can potentially be avoided as well using SDN based monitoring and detection.

A next step is the integration of the system with the CogNet common infrastructure and the common infrastructure anomaly detection module in order to be able to showcase an end-to-end infrastructure while using a reference Machine Learning implementation for all the use cases.
2.4. LSP-CLUSTER: Network Traffic Clustering towards Anomaly Detection

2.4.1. Evaluation Scope

In Deliverable 5.3 (D5.3), we presented the LSP-cluster module for supervised clustering and applied it to the network data. In brief, we represent the input data set as a graph where the input data points are nodes. For example, in our case, the input dataset is a sequence of the network transmissions $t_1, ..., t_n$, thus each transmission becomes a graph node. Then, we draw edges between the $i$-th transmission node, $t_i$ ($i=1,n$), and the preceding transmission nodes, $t_1, ..., t_{i-1}$. The LSP-Cluster approach predicts the weights of the edges and outputs the maximum spanning trees on the resulting graphs as the final clustering. Please refer to D5.3 for the detailed description.

Here, we test whether LSP-cluster can be adapted to capture anomalous traffic. Previously, we tested the clustering ability of the module on the data from the ECML-PKDD Network Classification Challenge (NetCla)\(^9\), in a way that transmissions generated by a certain type of applications were considered as belonging to the same cluster. Here, we will slightly modify the task formulation and cast it as an anomaly detection task. We will still cluster the transmissions crossing the data plane according to their application type, only now we will regard any incoming transmission which cannot be clustered (or added) to one of the preceding clusters, i.e., application types, as a potentially anomalous transmission.

Our formalization of the anomaly detection task is similar to that of [14], who accurately formalize anomaly detection as a clustering problem under the assumptions that "new data that do not fit well with the existing clusters are considered anomalies" and "in a clustering which has clusters of various sizes, the smaller and sparser ones can be considered anomalous and the thicker ones normal". They propose an approach based on unsupervised clustering. Our LSP approach is based on the supervised clustering which does not require a pre-defined number of output clusters, which unsupervised algorithms crucially depend on. Another advantage of the graph-based LSP-cluster model is its incremental nature, which enables a new coming data point to be clustered at a cheap cost within a current clustering without the need to reconsider the previous clustering decisions.

We evaluate the LSP-cluster capability for anomaly detection in two scenarios. First, we simulate anomalies in NetCla data. And, second, we use an intrusion detection NSL-KDD\(^10\) dataset [15] (a new version of KDD Cup 1999\(^11\) dataset thus enabling comparison to the state of the art).


\(^10\) [http://unb.ca/cic/research/datasets/nsl.html](http://unb.ca/cic/research/datasets/nsl.html)

2.4.1.1 Data Preparation

**NetCla:** We divide the 19 classes (0 class apart) into two disjoint sets, $C_N$ and $C_A$, so that $C_N$ contains more populated classes, and $C_A$ – less populated, respectively. Since generally the portion of the normal traffic is considerably large compared to the anomalous traffic, we consider the data points belonging to one of $C_N$ classes (most of the traffic) to be normal transactions, whereas those from $C_A$ – anomalies. $C_A$ is further divided into two non-intersecting parts $C_{A1}$, $C_{A2}$, and we inject “anomalous” data points from $C_{A1}$ into traffic samples for training, and from $C_{A2}$ – into those for testing. Therefore, in the test data we have anomalies from classes not seen in the training data.

We split all the data into the samples of $N = 1000$ transmissions, among which most are normal ($\in C_N$), and few (up to some small $k$ from each of “anomalous” classes ($\in C_A$ in correspondence to their train/test partitions) – abnormal. More specifically, in our experiments $C_N$ contains 7 out of 19 classes with the highest number of representatives. The remaining twelve classes are divided into two groups of six and six and constitute $C_{A1}$ and $C_{A2}$, respectively.

**NSL-KDD:** In the NSL-KDD data, the connection records attributed to normal traffic are annotated as one class type. The attack connections are labelled with several types, among which some appear only in test data. This is similar to the scenario that we adopted above for simulating data for anomaly detection in NetCla, with $|C_N| = 1$. Due to the fact that the normal traffic is presented as one class only and the proportion of attack connections is quite high with respect to the normal traffic, we do not split the data into samples but treat it as a continuous sequence, and for each data point we consider $N = 10$ candidate links to preceding data points in the clustering graph, as illustrated in Deliverable 5.3.

2.4.2. Evaluation Metrics

To evaluate the results, we will use the standard metrics adopted in the community [14], that measure ratios between the rates of correct and erroneous predictions:

$$\text{Precision} = \frac{TP}{TP + FP},$$

sensitivity (or true positive) rate

$$\text{Sensitivity} = \frac{TP}{TP + FN},$$

false positive rate

$$\text{False positive rate} = \frac{FP}{FP + TN},$$

and overall accuracy

$$\text{Overall accuracy} = \frac{TP + TN}{TP + TN + FP + FN},$$

where,

$TP$ – correctly identified anomalies,

$FP$ – normal traffic incorrectly identified as anomalous,
2.4.3. Evaluation Tools
We provide this component as a standalone component, which can be actuated if required. We evaluated it by means of scripts, which implement the evaluation metrics mentioned above, on the NetClia and NSL-KDD datasets.

2.4.4. Testbed description
No testbed foreseen for this component.

2.4.5. Evaluation Results and their assessment
We present the results of LSP-cluster model on the NetClia and NSL-KDD data in Table 6. The percentage of the anomalies in the two testing scenarios differs (see the rightmost column of Table 6). The experimental results show that the model can be adapted to capture anomalous transmissions (or connections) in either case, regardless of whether their ratio is high or very low with respect to normal traffic. This is also due to the fact, that the model is based on evaluating similarity between the data points rather evaluating similarity to the training data distribution as the classification-based models do. In the latter case, the new data points are attributed to one of the classes seen on training, while LSP-cluster is able to detect a new pattern not seen during training and handles it as a potential anomaly.

As Table 6 shows, LSP-cluster obtains very low false positive rate. As for sensitivity, a real-life scenario might require higher sensitivity levels. They can be increased, for example, by fine-tuning the models’ parameters, as the models listed in Table 6 are not fine-tuned. There is yet appreciable room for improvement by means thorough parameterization and model selection, and by balancing between false positive and false negative error rates.

<table>
<thead>
<tr>
<th>Data</th>
<th>Precision, %</th>
<th>Sensitivity, %</th>
<th>FP rate, %</th>
<th>Overall accuracy, %</th>
<th>Anomaly percentage, %</th>
</tr>
</thead>
<tbody>
<tr>
<td>NetClia</td>
<td>37.93</td>
<td>61.11</td>
<td>0.60</td>
<td>99.17</td>
<td>0.60</td>
</tr>
<tr>
<td>NLS-KDD</td>
<td>94.17</td>
<td>70.37</td>
<td>5.75</td>
<td>80.66</td>
<td>56.92</td>
</tr>
</tbody>
</table>

Table 6 LSP-cluster performance on NetClia and NSL-KDD datasets

In Table 7, we compare LSP-cluster to two state-of-the-art classification-based intrusion detection systems (IDS): an SVM-based dynamic IDS [16], and an Artificial Neural Network (ANN) based IDS [17]. Please note this is not a strict comparison, since we report the SVM/ANN performance as
published in [16] and [17], and do not re-implement them. Moreover, SVM and ANN employ different feature selection methods.

<table>
<thead>
<tr>
<th></th>
<th>LSP-cluster</th>
<th>SVM</th>
<th>ANN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision, %</td>
<td>94.17</td>
<td>93.81</td>
<td>98.86</td>
</tr>
<tr>
<td>Sensitivity, %</td>
<td>70.37</td>
<td>88.83</td>
<td>95.77</td>
</tr>
<tr>
<td>FP rate, %</td>
<td>5.75</td>
<td>7.74</td>
<td>-</td>
</tr>
<tr>
<td>Overall accuracy, %</td>
<td>80.66</td>
<td>90.30</td>
<td>95.77</td>
</tr>
</tbody>
</table>

Table 7 Comparison of LSP-cluster to other systems on NLS-KDD dataset

The LSP-cluster performs comparably to and even slightly outperforms the SVM model. ANN, the neural network based system, provides a more accurate model, however, it has higher complexity. In general, LSP-cluster, a simple linear-perceptron-based model that has linear running time, is faster to train and tune than both SVMs and neural networks (even shallow ones).

### 2.4.6 Conclusions and next steps

The combination of the state-of-the-art accuracy, precision and FP rate on the one hand, and its computational efficiency on the other hand, make LSP-cluster a good candidate for anomaly detection in the real-life environment with large amounts of data to be processed.

Before putting LSP-cluster into a particular real time deployment, one can explore the following several directions of enhancing the model:

- What the best method is for selecting the preceding candidate data points (or representatives of the preceding clusters) for a data point in the cluster graph, against which it can be tested on whether to fall into the same cluster or not. We cannot afford to check all possible preceding points, as this would be very computationally expensive,
- The choice of effective feature representations,
- Optimization to the metrics most crucial in a particular environment,
- Determining the conditions on which the model should be re-trained.
3. Network Resilience Testbeds

3.1. Performance degradation (Dense Urban Area Testbed)

3.1.1. Evaluation Scope

Anomaly detection is useful in various application domains (e.g., health, security for fraud/intrusion detection, systems) and is particularly beneficial for 5G network operators to avoid or detect in timely manner performance degradations, which can otherwise lead to service outages and increased costs for maintenance. The objective is to detect the degradations either in a proactive (i.e., prepare before they occur) or reactive (i.e., react after they occur) manner to allow to apply corrective actions at an early stage to minimize their impact and reduce cost.

The Open5GCore testbed was described in D5.3 where anomaly detection model and MME-LB could be used to identify possible anomalies and tackle the problem of performance degradation and system failures. For the evaluation scope, a dataset of over 50000 records collected in an interval of 5 seconds are fetched from the Zabbix Server and then filtered using python script for MME machine. Each record contains the hostname, timestamp, metric name, and metric value in csv format. This procedure is repeated for collection of both normal and anomaly datasets. The Anomaly Detection with LSTM module is trained with these datasets. The large dataset ensured proper training. The output result seems to be promising with the detection of the set of anomalies and found to be similar which when compared with the true occurrence of anomalies during the tests. The results are discussed in below sections. Another purpose is also to stream the data (as json object) of active MME to Kafka queue on some specific topic and to investigate the ability of the model to detect the possible anomalies and avoid possible system failures.

3.1.2. Evaluation Metrics

3.1.2.1 Anomaly Detection Evaluation Metrics

We utilize the ADE: Anomaly Detection with LSTM module from WP3 deployed on the Network Resilience for Performance Degradation testbed in order to detect potential performance degradations in the server. We evaluate the anomaly detection module based on the precision, recall and F1-score, which is defined as:

\[ F_1\text{-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \]

The F1-score represents the harmonic mean of precision and recall. The precision is defined as the number of true anomalies discovered, divided by the total number of anomalies discovered. The recall of a technique is defined as the number of true anomalies discovered, out of the total number of true anomalies.
We consider an anomaly as detected if the approach detected the anomaly within the anomaly window introduced in [18] and defined as 10% of the dataset length divided by the number of anomalies contained. This is to allow flexibility for methods that detect the anomaly a bit earlier or a bit later than its occurrence.

Moreover, as ADE utilizes an explicit generalization model to learn the normal behaviour of the system, we also evaluate how accurate its predictions are with the following well-known error metrics:

- **Root-mean squared error (RMSE):** is an error metric frequently used to quantify the differences between the actual values observed in the data and the ones predicted by a model or estimator. RMSE is computed as the square root of the mean of the squares of the deviations between the predicted and actual values.

- **Mean Absolute Error (MAE):** is another metric used to measure the errors of the predicted values computed to the actual. This error is computed as the mean of the absolute differences between the predicted and actual values.

- **Coefficient of determination (R2):** provides a measure of how well observed actual values are replicated by the model, based on the proportion of total variation of actual values captured by the model. It also ranges from 0 to 1, where the higher the better. The error is computed as the difference between 1 and the sum of squares of residuals divided by the total sum of squares.

### 3.1.2.2 Testbed Evaluation Metrics

The common metrics such as CPU usage, network usage of the components in the testbed have been collected and used for evaluation purposes. The CPU util. metric of MMEs are exclusively used to train the anomaly detection model. Also, the different QoS parameters are measured and compared in different scenarios. The results are shown in the Section 3.1.5.2.

### 3.1.3. Evaluation Tools

In Open5GCore, the BT is used as input tool for generating traffic and to excite the system by attaching/detaching UEs. This synthetic traffic can be used as a source for normal traffic with less attachments/detachments and very large values of attachments/detachments can be considered as anomalous traffic.

**Actuation:**

The nodejs script in the configuration and orchestration stub machine establishes web socket connection with the module that is supposed to send the anomaly detection result. The stream data (json object) containing CPU usage metric information are sent periodically to the Kafka queue on some specific topics ‘metrics_mme1’ and ‘metrics_mme2’ for MME1 and MME2. The data is consumed by the Kafka consumer present in the module running in CI. The periodic result is then forwarded back to the configuration and orchestration stub via web socket. The anomaly label value (0 or 1) in the result is then analysed and based on it, if the value is 1, a trigger is sent to the currently active MME where the possible anomaly could have occurred. A python script listening on the active MME gets the trigger when then stops the MME service and restarts the service again.
In the meantime, a trigger is received by MME-LB which causes it to forward the control plane requests to the hot standby MME. Now, this MME becomes active and the former one registers itself to MME-LB in the hot standby state.

The ADE predicts the possibility of occurrence of anomaly in MME and acts pro-actively and helps MME-LB to switch to hot standby MME which keeps the system functional.

### 3.1.4. Testbed description

In deliverable D5.3, it has been discussed about how we can deal with indicators of the performance degradation in systems and using machine learning techniques to take corrective actions in advance to avoid any performance issues.

The Performance degradation testbed for Network Resilience is shown in the Figure 15. In this deliverable D5.4, we employ two scenarios. In one scenario which doesn’t involve machine learning section, a fixed threshold for the CPU utilization is set and when the system is excited with BT generating traffic, we check if the threshold is exceeded and a trigger is sent to stop the active MME. When the active MME is not in action, the MME-LB switches to the hot standby MME to handle the action. A complex procedure of transferring the load to the hot standby MME is executed to bring the system to the active and running state.

Another scenario involves the CI and CSE docker container along with Open5GCore. The deployed anomaly detection in CSE predicts the anomaly in CPU utilization metric and reports back to Open5GCore. The collected CPU usage metric data is sent as a stream to the Kafka server in the CI which is processed by CSE. It then sends back the output result that contains the anomaly label value. A web socket connection is already established between the CSE and configuration and orchestration stub. The result is pushed to this machine where the anomaly label value is analysed by a script and sends the UDP message to the active MME if there is anomaly. The script running in active MME receives the trigger and then stops the service. It eventually triggers the MME-LB to switch to the hot standby MME to make it as active MME and to forward all the requests. The former MME is started again and registers itself as a hot standby MME to the MME-LB.
These two scenarios will be compared on basis of metrics and output data to show that the deploying of machine learning techniques is better and have a proactive nature of handling the anomalous behaviour to take corrective actions and keep system alive and running.

The normal and anomalous data are collected from the Fraunhofer Testbed which will be used to train the anomaly detection model. It can be used to proactively detect anomaly in the MMEs to avoid system failure.

3.1.5. Evaluation Results and their assessment

3.1.5.1 Anomaly Detection results

The ADE anomaly detection module can function either in a batch or in a hybrid mode, where the latter implies a model trained in the batch layer and deployed for scoring the near-real time layer. In this section, we present the results of the module working in the batch mode, i.e., trained on batches of data and deployed for scoring on batches of data, since we utilize offline collected data from the testbed for the evaluation of precision, recall and F1-score, as the evaluation required labelling the anomalous class of the data in order to calculate the metrics.

Figure 16 and Figure 17 show the application of ADE to the data collected from MME1 for system CPU utilization and system CPU load, respectively. The upper part of both Figure 16 and Figure 17 show the predicted versus the actual values for the CPU utilization and load, respectively. The predicted are obtained from the model trained on the normal dataset collected. The lower part of the plot illustrates the $S_{\text{error}}$ (i.e., the squared error) which represents the measurement used for quantifying the deviation between the actual and predicted values, and in addition the $\text{AnomalyLabel}$ which is 1 when an anomaly is discovered and 0 when the data point is considered a normal instance. The dashed lines represent the true anomalies as labelled during the data collection phase.
Figure 16 ADE applied on MME1 system CPU utilization collected from the Network Resilience testbed.
As described in the Evaluation Metrics Section 3.1.2, we evaluate the precision, recall and F₁-score of ADE on the test dataset above based on whether the detection of an anomaly within the anomaly window of the true anomalies illustrated by dashed lines in Figure 16 and Figure 17 above. Based on this, we observe for the CPU utilization that ADE was able to detect all the nineteen true anomalies, leading to a recall of 1, and from the anomalies reported, there were two false positives reported (i.e., the other anomalies were either detected slightly after or before the anomaly occurred, but within the anomaly window), leading to a precision of 0.9, finally leading to a F₁-score of 0.95. Moreover, we observe for the CPU load that ADE was able to detect all the seventeen true anomalies, leading to a recall of 1, and from the anomalies reported, similarly to CPU utilization, there were two false positives reported leading to a precision of 0.89, which in turn lead to a F₁-score of 0.94. This leads to an overall average precision, recall and F₁-score of: 0.895, 1, and 0.945, respectively.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>RMSE</th>
<th>MAE</th>
<th>R²</th>
</tr>
</thead>
<tbody>
<tr>
<td>MME1 CPU Utilization</td>
<td>0.1426</td>
<td>0.0636</td>
<td>0.8238</td>
</tr>
<tr>
<td>MME1 CPU Load</td>
<td>0.0068</td>
<td>0.0024</td>
<td>0.9146</td>
</tr>
</tbody>
</table>

Table 8 ADE Prediction Evaluation on the test data
With regards to the accuracy of the predictions, we observe the errors reported in Table 8 for the CPU utilization and CPU load ADE predictions. The low errors and high variance captured indicate that the model successfully learnt the underlying patterns of the datasets under observation and was able to produce close to actual predictions.

### 3.1.5.2 Testbed evaluation results

The rate of reaction is the total roundtrip time (RTT) taken from the time the message is sent using Kafka to the CI to the time the result is received by the component in the testbed if ADE is involved in the process. Otherwise, without ADE, a threshold is set within the component to trigger the MME-LB when the threshold is exceeded. In the Figure 18, with CI and ADE involved, the maximum RTT for average rate of reaction is below 100 ms which is quite quick to predict anomaly based on real-time data. For the rate of reaction of without ADE is quite less than 5-7 ms.

![Comparison of Rate of Reaction](image)

**Figure 18 Comparison of rate of reaction using RTT method**

<table>
<thead>
<tr>
<th>Attachment for MME1</th>
<th>Average duration(ms)</th>
<th>CPU Utilization</th>
<th>Inbound traffic on &quot;mgmt&quot; interface (Kbps)</th>
<th>Outbound traffic on &quot;mgmt&quot; interface (Kbps)</th>
<th>Anomaly Label(1/0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MME1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MME2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MME1 without ADE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MME2 without ADE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Attachment for MME1
- Average duration(ms)
- CPU Utilization
- Inbound traffic on "mgmt" interface (Kbps)
- Outbound traffic on "mgmt" interface (Kbps)
- Anomaly Label(1/0)
The Table 9 shows different metrics data related to CPU utilization and network usage along with occurrence of anomaly validated by Anomaly Label which is received from ADE running in CI for MME1 during attachment procedure. As we can see, the number of attachment of UEs are increased in each row keeping the number of operations constant at 15 ops/sec. We can observe that the as CPU usage increases (along with attachment duration as shown) which triggered the Anomaly Label to be 1. This would trigger the MME-LB to switch to MME2 and restart the MME1 service. Similar observations have been observed in MME2 as both are the virtual machines clones running the same service with different configuration files. We also observed that with less number of operations per second during attachment procedure, it is less likely to produce anomalies with Anomaly Label being most of the times 0. Also, sometimes during the attachment procedure, the MME1/MME2 service terminates before anomaly occurrence is detected due to internal error which are ignored during data collection.

<table>
<thead>
<tr>
<th>Attachment for MME1</th>
<th>Average duration(ms)</th>
<th>CPU Utilization</th>
<th>Inbound traffic on &quot;mgmt&quot; interface (Kbps)</th>
<th>Outbound traffic on &quot;mgmt&quot; interface (Kbps)</th>
<th>Threshold Exceeded(1/0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 15 0</td>
<td>600</td>
<td>0.933</td>
<td>28.65</td>
<td>37.38</td>
<td>0</td>
</tr>
<tr>
<td>200 15 0</td>
<td>619.88</td>
<td>1.4693</td>
<td>48.45</td>
<td>65.41</td>
<td>0</td>
</tr>
<tr>
<td>300 15 0</td>
<td>613.86</td>
<td>1.5154</td>
<td>43.78</td>
<td>58.33</td>
<td>0</td>
</tr>
<tr>
<td>400 15 0</td>
<td>639.42</td>
<td>1.6878</td>
<td>84.9</td>
<td>114.19</td>
<td>0</td>
</tr>
<tr>
<td>500 15 0</td>
<td>626.45</td>
<td>1.9207</td>
<td>76.65</td>
<td>111.16</td>
<td>0</td>
</tr>
<tr>
<td>600 15 0</td>
<td>712.40</td>
<td>2.43</td>
<td>89.22</td>
<td>109.22</td>
<td>0</td>
</tr>
<tr>
<td>700 15 0</td>
<td>812.11</td>
<td>2.59</td>
<td>102.3</td>
<td>111.33</td>
<td>1</td>
</tr>
</tbody>
</table>
Table 10 Observation of different metrics for MME1 during Attachment without ADE

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>800</td>
<td>15</td>
<td>0</td>
<td>889</td>
<td>2.78</td>
</tr>
<tr>
<td>900</td>
<td>15</td>
<td>0</td>
<td>945.8</td>
<td>2.98</td>
</tr>
<tr>
<td>1000</td>
<td>15</td>
<td>0</td>
<td>1012.4</td>
<td>3.16</td>
</tr>
</tbody>
</table>

The Table 10 shows different metrics data related to CPU utilization and network usage along with occurrence of anomaly validated by a threshold value 2.5 which would trigger the MME-LB to switch to hot standby MME if that threshold value is exceeded for MME1 during attachment procedure. So, here, ADE and C1 aren’t in use. The optimal value of threshold is chosen to conduct this experiment. The corresponding increase in CPU utilization values correspond to the number of attachments as shown in Table 10.

When the Table 9 and Table 10 are compared, the obvious win is the one with the ADE that efficiently predicts the possibility of anomaly on MME service and quickly reacts to trigger the MME-LB to switch to hot standby MME to avoid service failure.

3.1.6. Conclusions and next steps

The results shown in Section 3.1.5 shows that using the Anomaly Detection Enabler module as part of network component failure detection is promising. Considering the ability of ADE to cope with real-time data points to predict the possible anomalies, made it possible to take corrective actions instantly avoiding the potential degradation of the performance of the system.

The processing of the data point and prediction is done under 100ms which helps to take actions quickly, compared to the current reactive solution where we have 500ms for overall switching of context to the hot standby component after the event.

In the next steps, the ADE based solution will be further extended towards other research projects with industry partners aiming to cover the high availability use case for all the components of the Open5GCore and with this to provide a complete high availability solution for testbeds.
3.2. SLA

3.2.1. Evaluation Scope
The scope of our evaluation is twofold: Firstly, identifying SLA violations in an NFV-based environment, secondly, anticipating SLA violation before occurring thus improving the QoE.

3.2.2. Evaluation Metrics
We used four metrics, all are significant. In order to have an overall view on the ANNs performance we used accuracy metrics defined as follows:

\[ \text{Acc} = \frac{TP + TN}{P + N} \]

Where TP refers to True Positive as correctly identified as an SLOV, TN as True Negative as correctly discarded as a non SLOV, P stands for Positive as the number of SLOV, and N for Negative as the number of non SLOV.

Additionally, we added precision, recall and Beta F-score, with beta = 0.2

Precision and Recall are defined as:
\[
\text{Precision} = \frac{tp}{tp + fp} \\
\text{Recall} = \frac{tp}{tp + fn}
\]

\[
F_\beta = \left(1 + \beta^2\right) \frac{\text{precision} \cdot \text{recall}}{\beta^2 \cdot \text{precision} + \text{recall}}
\]

3.2.3. Evaluation Tools
We evaluated our approach using two standard Machine Learning techniques. Firstly, by splitting the data into 3 sets: Training data (70%), test data (20%) and validation data (10%). We have used Confusion Matrix to evaluate the classification performance of the ANNs. In Figure 19, we present the best classification performance.
3.2.4 Testbed description

Clearwater was designed to respect the architectural requirements for deployment in cloud environment while respecting the IMS requirements as standardized in 3GPP. The properties delivered by Clearwater that mostly interest us are the scale-out capability of all the VNFCs and its build-in stress generator (SIPp Stress-ng) to emulate SLOVs. Clearwater is also compatible with OpenStack and Cloudify which allow us to fully integrate it to our Cloud infrastructure.

As depicted in Figure 20, the testbed setup consists of 10 VMs (6 Clearwater VMs, 1 Monasca VM, 1 SIPp VM, 1 VM where we run the framework, 1 DNS VM) and over of 2 Gbits of collected Monitoring traces. In addition, 30,000 different SIP profiles were created and stored using Homestead in the local Cassandra Database. These profiles are used to generate traffic and anomalies when they are launched simultaneously. The monitoring tool for our framework is Monasca. Monasca is a MaaS, Monitoring-as-a-Service solution from HP, built as a highly scalable OpenStack service.
3.2.5. Evaluation Results and their assessment

We have tested the classification performance of our data set using randomly generated Neural Networks. More specifically we have generated Feedforward Neural Networks and Long Short Term Memory Networks. We have repeated each experimentation 20 times. The results are depicted as boxplots as in Figure 21, the small boxes represent the quartiles of the distribution. The red line in the middle represents the median, i.e. the point separating the data into half. The outliers are drawn as black crosses outside the box.
Overall as shown in Figure 21, LSTM is more performant than FFNN but is more prone to overfitting when incrementing the number of epochs. One solution is to use early stopping. On the other hand, FFNN can yield comparable results but requires more hyperparameters tuning. The power of Random ANN generation technique lays in its simplicity. Although, it cannot guarantee the optimal performance, it scales very well and is a good alternative when human intuition fails.

We have randomly tried different weight initialization methods for the ANN. The results over the initialization method in Figure 22 show that glorot uniform is the best initialization method. Orthogonal and Uniform initialization give the highest probability for converging to an acceptable accuracy. While, glorot normal, normal and zero should be avoided.

Figure 21 Comparison between LSTM and FFNN by accuracy over the validation set

Figure 22 ANN initialization method versus accuracy. GN: Glorot Normal, GU: Glorot Uniform, HN: He Normal, HU: He Uniform, LU: Lecun Uniform, N: Normal, O: Orthogonal, U: Uniform, Z: zero.
We consider the activation function (in the output layer) a relevant hyperparameter for the overall accuracy. As shown in Figure 23, the best activation function for SLO violation classification is the **linear function** followed by the **hard sigmoid** and **softmax**.

![Figure 23 Accuracy with respect to Activation functions in the output layer. This figure shows that the choice of the activation layer is amongst the most important ones.](image)

**Figure 23** Accuracy with respect to Activation functions in the output layer. This figure shows that the choice of the activation layer is amongst the most important ones.

![Figure 24 Top 5 ANNs based on the mean over validation score accuracy. All the best ANNs appear to converge on 94% average accuracy.](image)

**Figure 24** Top 5 ANNs based on the mean over validation score accuracy. All the best ANNs appear to converge on 94% average accuracy.
We observe that ANN successfully managed to learn the underlying complex patterns of the dynamic environment expected in future networks. Through our methodology, to combat the obstacle of expert knowledge required for tuning such complex deep neural networks, we show that the best architecture was selected. We had some expectations in mind on what the results would look like. We expect that deep layered ANNs will perform better than the classical ones. Even though this intuition was right, we get a broader and more complete perspective on what ANN can perform and how it is possible to utilize their hyperparameters for better results with reasonable resources and tight budget constrained.

The 4 shows the best ANNs according to the validation score. We refer to the ANN models name as "X" where X is the ID of the model.

Notice that the best ANN is \`\_71\` over validation score (Figure 20) distributions are focused in one point, the 94% accuracy point. We suspect that given the initial settings, the best results were guaranteed even after 20 different trials. Moreover, it seems that the ANNs couldn’t go further that 94% due to the hyperparameters range restrictions. In other words, if we have allowed the ANNs to explore higher depth, width and epochs, we could have surpassed the barrier of 0.94. However, this claim should be backed with more evidence.

### 3.2.6. Conclusions and next steps

We have investigated an empirical approach to machine learning that consists of generating multiple random ANNs with different factor of variations for the SLA use case in the context of Network Function virtualization. We provided an end-to-end data-driven methodology for network management. We used a real test case using a virtual IMS to instantiate the NFV framework. Generate data from experiment and stress the management opportunities and challenges brought by NFV and how a data-driven approach based on ANN can be leverage to tackle the management aspects.

Our results support the following conclusions:

- FFNN and LSTM if properly configured can yield high accuracy (up to 94%).
- ANNs are very sensitive to the hyperparameters. FFNN yields better results in Wide architectures, while LSTM requires Deep architecture.
- Overall, LSTM is more performant than FFNN but is more prone to overfitting when incrementing the number of epochs.
- The power of Random Search method lays in its simplicity. Although, it cannot guarantee the optimal performance, it scales very well and is a good alternative when human intuition fails.

To confirm our findings, we are currently running multiple experiments on the cloud with more resources and a wider range of hyperparameters. Moreover, we are considering adding additional ANN types from the ANN zoo to get a broader perspective on the power of ANNs for SLA management in a virtualized context.

We argue in document that the management of the programmable networks should necessary incorporate intelligent and cognitive solutions. We demonstrate how Machine Learning (ANN) can be leverage to this end.
For future work we are considering using different ANN types such as ConvNets. We are currently investigating a novel method based on a guided random search to improve ANN’s overall performance.
4. Conclusions and Further Work

4.1. D5.4 Conclusions

In this deliverable, the testbeds considered for the security and for the network resilience with machine learning were assessed. As the decisions of the management plane are rather not correlated, each addressing a specific feature of the fault, configuration, accounting, performance and security (FCAPS) the same approach was taken also by the work package. The following key conclusions could be taken for each of the testbeds:

1) Distributed Security Enablement – changing the processing path of flows is now representing the best alternative for network management in virtualized environments in order to be able to isolate specific data traffic. With dynamic statistics on the data flows, the processing of path flows can be adapted to the specifics of the data traffic. Key to the evaluation in this deliverable is that this adaptation is becoming faster and better when machine learning techniques are introduced in the decision point. With this we gain not only flexibility of the data path selection but also flexibility of decision towards dynamic statistics.

2) Honeynet testbed – a complex set of malicious behaviour can be identified using machine learning. The proposed solution is clearly less complex and providing similar results to Deep-Packet Inspection (DPI) solutions. Being less complex, the solution represents a cost effective alternative to the current DPI components.

3) NFV security anomaly detection – moving security related components to data center level provide several advantages in terms of functional placement in the network substrate. As the system is able to detect anomalies based on metrics which could pertain to other components (e.g. CPUs) compared to firewalls and as the enforcement could happen in any virtual switch on any server in the data center, the proposed solution is adding a new level of granularity needed for securing cloud networks. The evaluation results also show that the proposed solution has a minimal penalty for the processing delay (400ms without optimizations), which prove the suitability of developing a machine learning based distributed firewall at data center level.

4) LSP-Cluster – the accuracy of offline methods of addressing traffic detection is becoming better with optimized machine learning algorithms combining state of the art accuracy, precision and FP rate with computational efficiency. With this, high precision detection using ML can be used for offline processing, key to analytics of the system behaviour.

5) Performance degradation – anomaly detection using machine learning provides the means to predict possible failures and to execute transfer to hot standby component before the actual failure happens. Having a proactive behaviour, the service towards the subscribers is not impacted. With this addition, the high availability system reacts faster, not requiring additional engineering solution oriented towards delay reduction, making it feasible for software networks deployed on top of cloud.
6) SLA – Similar to performance degradation, machine learning enables a fast detection of SLA violations and through this the means to a fast recovery. A system using the anomaly detection module to determine SLA violations is able to execute proactively mitigation actions, virtually not impacting the subscribers.

In the next steps, the following activities will be executed for the different use cases:

- Within CogNet project, all the testbeds will be integrated and evaluated (if not yet completed) using the common infrastructure. This would provide a second evaluation as well as an overview whether a remote machine learning is sufficient for such solutions, opening the door for remote machine-learning services for network management.

- The results obtained will be disseminated towards the scientific community (if not yet completed) and through this to provide visibility of the project results towards other software networks management activities.

- The testbeds themselves will be used as basis for next projects in the area of network management either publicly funded or as basis for direct research activities with the industry.

- The industrial partners (Orange, IBM and TID) are aiming to use the testbeds to motivate the further development of internal products based on the technology knowledge acquired.

- The NFV security anomaly detection testbed and the performance degradation testbed will be integrated into the Fraunhofer offering of testbeds and will be licensed towards industry R&D partners.

- The LSP-Cluster will be used by the UNITN as basis for further research in the accuracy of machine learning solutions.

4.2. WP5 conclusions

This deliverable represents the final contribution from the perspective of WP5 in CogNet. With the development and the evaluation of six different testbeds, the WP5 has managed to cover successfully the development of real-time security and resilience machine learning mechanisms providing a wide variety of proof-of-concept implementation for the different features showcasing the advantages brought by machine learning. Being a research project, the prototypes are meant not to transform as such into products, but to provide an overview towards the business units of operators and manufacturers on how the machine learning technology could be used for providing the management of a more robust, dynamic network ecosystem.

Due to evolution of the research community in the direction of machine learning usage for handling security in the SDN/NFV environment through usage of a combination of data at data path and at service level, WP5 developed three security related testbeds, each concentrating on the different security options: distributed security at application level, replacement of the existing
security solutions with cost effective ones and the embedding of the security components into the cloud fabric with the possibility to early detect anomalies. Although the testbeds aim to cover as much as possible the item of security and prove the efficiency of machine learning to solve security problems, due to the complexity of the domain, it is foreseen that the large wave of technological advancements will be continued in the next years by the community. With developments within the CogNet project, the project partners as well as the other parties interested into the domain are equipped with the foundation of what could be achieved with machine learning in carrier-grade network management giving a boost to the research.

- The honeynet testbed considered a classic approach to security, addressing mainly the functionality of a firewall which is placed at the entry of the network and is able to detect different attacks. The provided evaluation results prove that the implementation of machine learning based firewalls is feasible for a carrier-grade network and that it should be considered in the next generation of products. With this evaluations, operators are able to transmit to the vendors of software firewall components the message that firewalls could be improved by more dynamic data processing mechanisms and through this to reduce the final cost of such components.

- The distributed security enablement testbed, albeit addressing a firewall at the entry of the network has a further actuation implementation of security zones using SFC based routing, showcasing additionally to the large benefits of using machine learning for detection of attacks the benefit of dynamic networking solutions within the cloud infrastructures which allow with a minimal functionality addition to create different security zones. This is the main feature of interest of private network administrators which are still trying to discover how their current security zones are provided by the cloud infrastructures, one of the main limiting factors of adopting cloud infrastructures on a very large scale in enterprises.

- The security anomaly detection testbed made a further step into the direction of innovative features for security and to deploy a distributed security solution directly on the forwarding plane of the cloud, through this liberating the firewall functionality for the initial location into the infrastructure and enabling its placement between any two connected virtual machines. With this, security can be dynamically placed within the cloud environment and could address in a distributed manner (i.e. gradually) different types of attacks. Instead of concentrating on the placement of the network functions in different network locations to implement the security zones, the project proved that it is possible to place the security functionality at data path level and to use machine learning techniques to address the detection of only specific attacks in a specific location represents a new approach and new means to implement security in the cloud/NFV environment using SDN mechanisms.

From the perspective of resilience, another two distinctive testbeds were implemented and evaluated having the perspective of the software network components developers which are interested into determining the performance degradation and the operator perspective interested into the maintenance and guaranteeing of the SLAs towards the subscribers. Due to the simplicity of the proof-of-concept prototypes which aimed to underline the advantages of machine learning
for the specific items: increasing the resilience of the software components and the maintenance of the SLA by using the scaling mechanisms specific to cloud/NFV environments, the prototypes concentrated mainly towards the implementation and the integration of flexibility mechanisms as such, this being the basis for the further study of the network function placement functionality, which in itself requires knowledge on the exact operator topology targeted.

- The performance degradation testbed proved that machine learning represents a cost-effective means to determine when the performance of a network function is degrading and through this to predict the performance degradation of the network service. The proposed mechanism also covers abnormal behaviour (not only watchdog functionality) and proved that the anomalies can be detected and mitigated in due time before their effect is noticed by the users of the service. This represents a new approach to high availability where the specific machine learning helps to solve the problem of component functionality degradation on the long term.

- The Media SLA testbed addresses the performance degradation from the perspective of the SLA of the subscriber, proving that the same anomaly detection mechanisms should be used for the detection of the possible SLA degradation and to mitigate in real time such situations.

All the testbeds are profiting on the flexibility provided by the underlying cloud substrate of the SDN/NFV environment and are using the new provided mitigation mechanisms (e.g. re-routing, scaling, rebooting of network functions, etc.) to modify the network and the processing of the specific services according to the decisions taken by the machine learning algorithms. With the evaluation of the testbeds we have proven that machine learning techniques are feasible to be used as advanced means for the network management decisions in real-time as needed to be able to properly use the mitigation mechanisms proposed.

Furthermore, the proposed mechanisms were tested with local analytics, where the machine learning mechanisms were placed in close proximity to the network functions themselves. With this, a first evaluation of the algorithms was obtained providing to the reader the means to understand that the specific features are highly beneficial when placed locally into the network. However, this may be too complex as costs for real-life network deployments, as the distribution of such functionality would require large support from the analytics functionality providers. For assessing a more effective solution towards real-life deployments, most of the testbed algorithms will be also deployed in the framework of WP6 as part of the common infrastructure, which from a testbed perspective represents the centralized approach towards the network management. With this, a second evaluation of the mechanisms is provided as well as the means to integrate them into a single machine learning network management solution.

Since the start of the project, a very large momentum was seen for the machine learning usage in performance and security management. From this perspective, WP5 managed to provide innovation just in time for the needs of the software components providers, cloud providers and operators to be able to assess in an initial form the benefits of machine learning in network management and to make their decisions and roadmaps on the further integration towards the later product integration.
Considering that WP5 has reached its major objectives, that the proposed testbeds showcase a large number of performance and security related network management directions, all applied to cloud/NFV infrastructures and considering the research and the development community interest into the developed solutions, we consider WP5 as being successful, albeit being only an initial step into the integration of machine learning.
# Glossary, Acronyms and Definitions

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>5G</td>
<td>5th generation mobile networks</td>
</tr>
<tr>
<td>ACL</td>
<td>Access Control List</td>
</tr>
<tr>
<td>ADE</td>
<td>Anomaly Detection Engine</td>
</tr>
<tr>
<td>ANN</td>
<td>Artificial Neural Network</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>BT</td>
<td>Benchmarking Tool</td>
</tr>
<tr>
<td>CI</td>
<td>Common Infrastructure</td>
</tr>
<tr>
<td>CSE</td>
<td>CogNet Smart Engine</td>
</tr>
<tr>
<td>DDoS</td>
<td>Distributed Denial of Service</td>
</tr>
<tr>
<td>DNS</td>
<td>Domain Name System</td>
</tr>
<tr>
<td>DoS</td>
<td>Denial of Service</td>
</tr>
<tr>
<td>DSE</td>
<td>Distributed Security Enablement</td>
</tr>
<tr>
<td>ENISA</td>
<td>European Union Agency for Network and Information Security</td>
</tr>
<tr>
<td>EPC</td>
<td>Evolved Packet Core</td>
</tr>
<tr>
<td>FFNN</td>
<td>Feed Forward Neural Network</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>ICMP</td>
<td>Internet Control Message Protocol</td>
</tr>
<tr>
<td>iForest</td>
<td>Isolated Forest</td>
</tr>
<tr>
<td>ID3</td>
<td>Iterative Dichotomiser 3</td>
</tr>
<tr>
<td>LB</td>
<td>Load Balancer</td>
</tr>
<tr>
<td>LOF</td>
<td>Local Outlier Factor</td>
</tr>
<tr>
<td>LSSVM</td>
<td>Latent Structural Support Vector Machine</td>
</tr>
<tr>
<td>LSTM</td>
<td>Long Short Term Memory</td>
</tr>
<tr>
<td>MANO</td>
<td>NFV Management &amp; Orchestration</td>
</tr>
<tr>
<td>ML</td>
<td>Machine Learning</td>
</tr>
<tr>
<td>NAS</td>
<td>Network Access Server</td>
</tr>
<tr>
<td>NF</td>
<td>Network Function</td>
</tr>
<tr>
<td>NFV</td>
<td>Network Function Virtualization</td>
</tr>
<tr>
<td>NFVM</td>
<td>NFV Management</td>
</tr>
<tr>
<td>NFVO</td>
<td>NFV Orchestrator</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>OCSVM</td>
<td>One Class Support Vector Machine</td>
</tr>
<tr>
<td>OFS</td>
<td>OpenFlow Switch</td>
</tr>
<tr>
<td>OPNFV</td>
<td>Open Platform for NFV</td>
</tr>
<tr>
<td>P-CSCF</td>
<td>Proxy - <strong>Call Session Control Function</strong></td>
</tr>
<tr>
<td>RADIUS</td>
<td>Remote Authentication Dial-In User Service</td>
</tr>
<tr>
<td>RMSE</td>
<td>Root Mean Square Error</td>
</tr>
<tr>
<td>SDN</td>
<td>Software Defined Networks</td>
</tr>
<tr>
<td>SECaaS</td>
<td>Security as a Service</td>
</tr>
<tr>
<td>SFC</td>
<td>Service Function Chain</td>
</tr>
<tr>
<td>SFP</td>
<td>Service Function Path</td>
</tr>
<tr>
<td>SLA</td>
<td>Service Level Agreement</td>
</tr>
<tr>
<td>SLO</td>
<td>Service Level Objective</td>
</tr>
<tr>
<td>SP</td>
<td>Service Providers</td>
</tr>
<tr>
<td>SPAM</td>
<td>Unsolicited email</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>SYN</td>
<td>Synchronize message to establish TCP connection</td>
</tr>
<tr>
<td>TLS</td>
<td>Transport Layer Security</td>
</tr>
<tr>
<td>UE</td>
<td>User Equipment</td>
</tr>
<tr>
<td>VM</td>
<td>Virtual Machine</td>
</tr>
<tr>
<td>VNF</td>
<td>Virtual Network Function</td>
</tr>
<tr>
<td>VNFC</td>
<td>Virtual Network Function Controller</td>
</tr>
</tbody>
</table>
References

on Information and Communication Technology for Competitive Strategies (ICTCS’14),
Udaipur, Rajasthan, India, 2014.

for Intrusion Detection and attack classification,” in Twenty Second National Conference
on Communication (NCC), Guwahati, India, 2016.

-- The Numenta Anomaly Benchmark. 14th IEEE International Conference on Machine
Learning and Applications (ICMLA), 2015.