HORIZON 2020

Information and Communication Technologies

Integrating experiments and facilities in FIRE+

Deliverable D3.1
Performance test tools detailed framework report

Grant Agreement number:	**687884**
Project acronym:	**F-Interop**
Project title:	**FIRE+ online interoperability and performance test tools to support emerging technologies from research to standardization and market launch** **The standards and innovations accelerating tool**
Type of action:	**Research and Innovation Action (RIA)**
Project website address:	**www.finterop.eu**
Due date of deliverable:	**31 August 2016 (M10)**
Dissemination level:	**PU**

This deliverable has been written in the context of the Horizon 2020 European research project F-Interop, which is supported by the European Commission and the Swiss State Secretariat for Education, Research and Innovation. The opinions expressed and arguments employed do not engage the supporting parties.

Document properties

Responsible partner	EANTC AG
Author(s)/editor(s)	Eduard Bröse (EANTC AG), Eldad Zack (EANTC AG), Maria Rita PALATTELLA (University of Luxembourg), Luca Lamorte (University of Luxembourg), Eunah Kim (Device Gateway), Michael Hazan (Device Gateway), Matteo Filipponi (Device Gateway), Cedric Crettaz (Mandat International)
Version	2.0
Keywords	Remote testing, Online testing, Network testing, Performance testing, Scalability testing, Privacy testing, Energy efficiency testing, Internet of Things (IoT), CoAP, spatial and map representation, GUI

Abstract

This document, **Deliverable D3.1 - Performance test tools detailed framework report,** provides an overview of the design approach adopted for developing performance and privacy test tools.

The document describes the framework functionality, the common properties shared by all test instances as well the specific properties which differentiate one tool from another. The deliverable overviews the already existing tools on which the F-Interop tools will be leveraged on, and provides gap analysis, identifying the missing component to be implemented. It describes the mechanism by which the different tools can communicate to provide a coherent test environment. Finally, it presents the preliminary view of the spatial and map representation of online tests.

Table of Contents

Table of Contents 3
List of Figures 5
List of Tables 6
List of Acronyms 7
1 Introduction 8
1.1 About F-Interop 8
1.2 Objectives 8
1.2.1 Work package Objectives 8
1.2.2 Tasks Objectives 8
1.2.2.1 T3.1: Performance test tools (QoS, scalability and energy) 8
1.2.2.2 T3.2: Privacy tools and data management 9
1.2.2.3 T3.3: Spatial and map representation of experiments 9
1.2.3 Deliverable Objectives and Methodology 9
2 Overview of the high-level design concept 10
3 Performance and Privacy Tools Design Approach 14
3.1 Overview 14
3.2 Test Scenarios 14
3.3 Generic Test Module Topology 15
3.4 Design Approach for Testing Tools 16
3.4.1 Scalability Testing - Passive Monitoring 16
3.4.2 Scalability Testing - Active Emulation 17
3.4.3 Resiliency Testing - Network Impairment 18
3.4.4 Energy Consumption Testing 19
3.4.5 QoS Testing - Monitoring of SDN/NFV-Based Networks 20
3.4.6 Privacy Risk Testing - Data Leakage Analysis 22
4 Framework Functionality 24
4.1 General Functionality 24
4.1.1 Common Functionality 24
4.1.2 Timeline Controller Module 25
4.2 Performance Tools Functionality 26
4.2.1 Scalability: Passive Monitoring 26
4.2.2 Scalability: Active Emulation 28
4.2.2.1 Scalability: Active Emulation - CoAP Client Emulation 29
4.2.2.2 Scalability: Active Emulation - CoAP Server Emulation 30
4.2.3 Resiliency: Network Impairment 31
4.2.4 Energy Consumption: Platform-specific Measurement 33
4.2.5 QoS Monitoring in SDN/NFV-Based Networks 34
4.2.5.1 The troubleshooting QoE approach 37
4.3 Privacy Tools Functionality 37
4.4 Gap Analysis 39
5 Module Intercommunication Model 41
5.1 Module Intercommunication Requirements 41
5.2 Module Intercommunication Design 42
5.2.1 High Level Intercommunication Description 42
5.2.2 Message Routing Keys 42

5.2.3 Common Message Fields 43
5.2.4 Timeline Control 43
5.3 Spatial and Map Representation of online tests 44
6 Conclusion 48
7 References 49
8 Annex 1: Timeline Control Structure Example 50

List of Figures

Figure 1: High-level architecture with testbeds and the scope of testing tools 10
Figure 2: High Level Performance Tools Building Blocks 11
Figure 3: Test Scenarios Concept 14
Figure 4: Generic Test Module Topology 15
Figure 5: Passive Analysis Module Schematic 17
Figure 6: Active Emulation Module Schematic 18
Figure 7: Network Impairment Module Schematic 19
Figure 8: Energy Consumption Sampler Module Schematic 20
Figure 9: Synthetics Measure Module Schematic 21
Figure 10: Privacy Module Schematic 23
Figure 11: QoS Monitoring Tool 36
Figure 12: Privacy Module Working Flow 38
Figure 13: Network Visualization 44
Figure 14: Map visualization 45
Figure 15: Data set graph visualization 45
Figure 16: High-level building block of Visual representation tool 46
Figure 17: Timeline Example 50

List of Tables

Table 1: Framework Modules - Common Functionality 25
Table 2: Framework Modules - Timeline Controller Functionality 26
Table 3: Framework Modules - Passive Monitoring Functionality 28
Table 4: Framework Modules - CoAP Client Emulation Functionality 30
Table 5: Framework Modules - CoAP Server Emulation Functionality 31
Table 6: Framework Modules - Network Impairment Functionality 33
Table 7: Framework Modules - Energy Consumption Functionality 34
Table 8: Framework modules - QoE Synthetic measurement 37
Table 9: Framework modules - Privacy Functionalities 39
Table 10: Existing Tools and Gap Analysis 40
Table 11: Routing Key Association for Module Intercommunication 42
Table 12: Common Message Fields 43

List of Acronyms

AMQP	Advanced Message Queueing Protocol
CoAP	Constrained Application Protocol
DUT	Device Under Test
FIRE	Future Internet Research and Experimentation
GDPR	General Data Protection Regulation
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
ISO	International Standards Organization
IUT	Implementation Under Test
JDK	Java Development Kit
JSON	JavaScript Object Notation
NFV	Network Function Virtualisation
NFVI	Network Function Virtualisation Infrastructure
NFVO	Network Function Virtualisation Orchestrator
PDP	Personal Data Protection
QoE	Quality of Experience
QoS	Quality of Service
R&D	Research & Development
SASL	Simple Authentication and Security Layer
SDN	Software Defined Networking
SLA	Service Level Agreement
SME	Small Medium Enterprise
SUT	System Under Test
TBaaS	Test Bed as a Service
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
VIM	Virtual Infrastructure Manager
VNF	Virtual Network Function
VNFM	Virtual Network Function Manager
WP	Work Package

1 Introduction

1.1 About F-Interop

F-Interop is a Horizon 2020 European Research project, which proposes to extend the European research infrastructure (FIRE+) with online and remote interoperability and performance test tools supporting emerging technologies from research to standardization and to market launch. The outcome will be a set of tools enabling:

- Standardization communities to save time and resources, to be more inclusive with partners who cannot afford travelling, and to accelerate standardization processes;
- SMEs and companies to develop standards-based interoperable products with a shorter time-to-market and significantly lowered engineering and financial overhead.

F-Interop intends to position FIRE+ as an accelerator for new standards and innovations.

1.2 Objectives

1.2.1 Work package Objectives

- Research and develop performance test tools.
- Research and develop tools for privacy risk assessment
- Research and develop spatial representing tools to support experimenters.
- Research and integrate testing tools with network virtualization technologies such as OpenFlow / OpenDayLight based SDN / NFV environments.

1.2.2 Tasks Objectives

1.2.2.1 T3.1: Performance test tools (QoS, scalability and energy)

The main goal of this task is to provide the required software to enable remote online testing with respect to QoS, scalability and energy. This work will be based on the results of WP1 T1.1 “Testing tools requirements and analysis”. The tools will enable to test and measure:

- The Quality of Service in the interaction with the device under test, in particular in terms of latency.
- The scalability of the implementation under test (IUT), by analysing its behaviour and reaction to a growing number of interactions in order to identify a potential limit as well as to check if the linearity of the relation between the number of requests per seconds and the processing time of the device.
- The energy consumption impact of the tested device.

For QoS and performance testing in SDN/NFV networks / testbeds, the particular goals are a detailed analysis of performance and QoS parameter in OpenFlow v1.3 (e.g. SLA parameters, load/ delay/jitter control), the integration of the module in the

OpenDayLight network controller, and the definition and implementation of a northbound API to the network controller

This task will work in close cooperation with T3.3 to provide the required interfaces and API for the spatial representation module.

1.2.2.2 T3.2: Privacy tools and data management

This task will design methods for privacy analysis of the data exchanged while running different kind of tests on the F-Interop platform. First, the State-of-Art of traffic analysis on encrypted data flows will be studied, in order to check potential suitable solutions for F-Interop. Then, tools for checking how privacy issues may raise due to information leakage will be developed. In detail, it will be investigated how an adversary may get “sensitive” information (e.g., results of a test running on the shared platform) by passively observing patterns of encrypted communication, based for instance on packets size, packet ordering, packet direction, and timing. This task is also intended at researching and identifying personal data, such as user identifiers, leaked by device under test, keeping in perspective the newly adopted European General Data Regulation (GDPR) and its impact on personal data protection (PDP).

1.2.2.3 T3.3: Spatial and map representation of experiments

This task will research state-of-the-art technologies for enabling a flexible visual representation of experiments, and will provide a visualization tool supporting performance and privacy tests. The tool will enable the user to get a clear representation of the network configuration and node deployment used in the test, both on the user side and on the test-bed side. It will ensure that the developing visualization tool fulfils scalability requirements and eases the integration of the experiment parameters such as Quality of Service (latency, packet loss, etc.). The task will start by specifying the requirements in terms of experiment visualization in conjunction with the other work packages. It will then analyse the state of the art in terms of open source solutions in order to compare and select the most relevant one. It will then customize and extend it to address the specific F-Interop requirements by following an iterative methodology through several iterations.

1.2.3 Deliverable Objectives and Methodology

The main objective of this deliverable is to describe the general framework designed to implement the performance and privacy tools in scope for the work package tasks.

We have first identified the test scenarios to be implemented, and then designed a framework generic enough to enable all of them, for different test tools. In detail, in each scenario, a subset of the tools should be employed to realize its requirements as analysed in deliverable D1.1 and to align with the F-Interop Architecture.

As second step we have identified already existing components such as established open source projects to be integrated into the framework of F-Interop performance test tool. This approach allows us to leverage existing, tested code instead of duplicating the effort. Moreover, since the F-Interop Platform and tool will be extended by 3rd parties, another benefit of this approach is that it allows us to experience first-hand integration of code into the framework and refine the APIs.

2 Overview of the high-level design concept

The design of the performance and privacy testing tools is based on the requirements identified in D1.1. The design approach for the test tools is described in the Section 3.4, with specific test areas listed below:

- Section 3.4.1"Scalability Testing - Passive Monitoring"
- Section 3.4.2 "Scalability Testing - Active Emulation"
- Section 3.4.3 "Resiliency Testing - Network Impairment"
- Section 3.4.4 "Energy Consumption Testing"
- Section 3.4.5 "QoS Testing - Monitoring of SDN/NFV-Based Networks"
- Section 3.4.6 "Privacy Risk Testing - Data Leakage Analysis"

Each section includes specific references to the requirements of D1.1 and how they were applied in the context of WP3.

The design of the tools is also based on the F-Interop high-level architecture that is detailed in D1.3. The following figure is a high-level F-Interop architecture. The red-dotted box is indicating the role of WP2 and WP3 developing testing tools. Within F-Interop scope, WP2 is developing interoperability and conformance testing tools and WP3 is developing performance testing tools. Although the features of the tools are different, they are being developed using the same generic architecture as shown in the Figure 1.

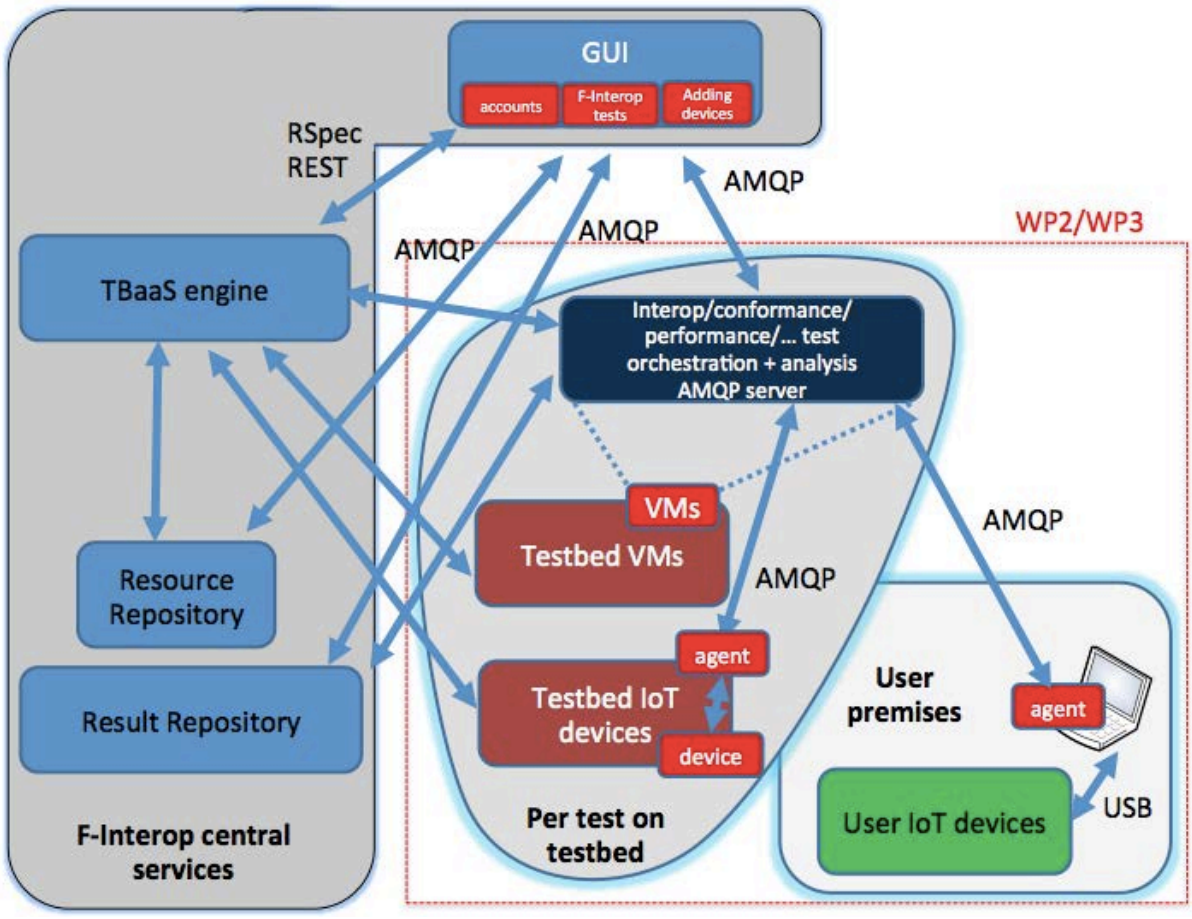

Figure 1: High-level architecture with testbeds and the scope of testing tools

Based on the general architecture view, the high level building blocks for the performance tools in WP3 are depicted in the following figure:

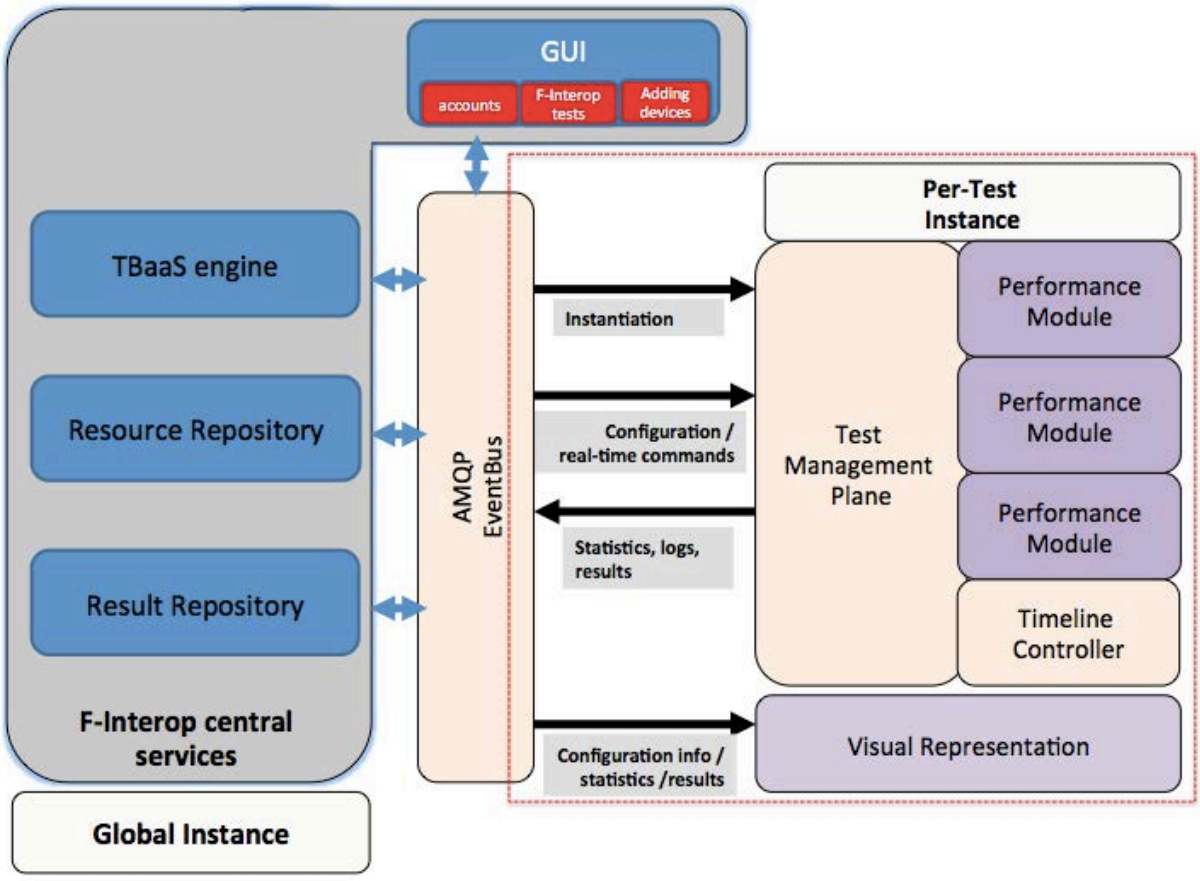

Figure 2: High Level Performance Tools Building Blocks

The Figure 2 illustrates simplified building blocks of performance tool modules no matter if the test is performing on user premise or on testbeds. It also shows the interacting information with F-Interop central services (global instance) though AMQP EventBus. In the F-Interop general architecture, the EventBus is shown as AMQP message exchanges among F-Interop components. In the Figure 2, it is illustrated as a functional building block in order to display the information to be exchanged between WP3 components and F-Interop central services. As shown, the components are divided in global instance components indicating F-Interop central services and per-test instance components indicating individual test runs from testing tools.

The global instance components representing F-Interop central services are shared between all F-Interop components and are out of scope for this work package (WP3). The TBaaS[1] in the F-Interop central services is responsible to instantiate the topology and the different modules/components of a given test, while the configuration of the actual test parameters and real-time commands, such as start and stop, is

[1] Note that the TBaaS design and implementation is out of scope for this document, being the focus of work package 4 (WP4).

communicated over the F-Interop EventBus toward the test management plane of the per-test instance.

The per-test instance components always include the test management plane (using AMQP to communicate between the components) and a timeline controller, which is responsible to change real-time parameters of modules according to a user-configured timeline. The other modules (depicted as "Performance Module" blocks above) are user-selectable. Each module may be instantiated more than once, allowing horizontal scaling of the modules in case higher performance is required or a different connectivity scheme is used for each module.

The per-test instance is buffered from the global instance for two reasons:

1. In performance testing, many messages are exchanged between components, due to timeline controller emitting messages to change the real-time controls of the modules.
2. While the test management plane and the timeline controller are not directly accessible to the user, performance modules may be provided by a user. This allows filtering of messages between the per-test instance and global instance, such that user experiments do not negatively impact the global F-Interop platform.

The setup process and information flow between the global instance and the per-test instance are as followed:

- TBaaS instantiates the modules. It provides the initial configuration of the test modules, allowing it to communicate to the test management plane node.
- The GUI (via the F-Interop EventBus) communicates the configuration and real-time commands from the user towards the components via the test management plane.
- The statistics, logs and results from the different modules is provided via the test management plane to the user GUI.

The methodology is designed to be suitable to test performance in SDN/NFV environments[2] as well as traditional network environments. The NFV architecture (Network Functions Virtualisation (NFV); Infrastructure Overview, 2015) is composed from the management and orchestration components — NFV orchestrator (NFVO), VNF Manager (VNFM) and Virtual Infrastructure Manager (VIM) — along with NFV infrastructure (compute nodes, storage nodes and network nodes) and virtual network functions (VNFs). Since the components of the framework are distributed per design, the performance modules are suitable to be deployed inside an SDN/NFV stack as VNFs. This is also the approach that test equipment vendors focus on for testing in NFV environment, with major test vendors offering virtual counterparts to their traditional physical testers such as Ixia IxNetwork VE (IxNetwork VE) and Spirent TestCenter Virtual (Spirent TestCenter Virtual).

Moreover, an OpenDayLight controller will be integrated within the QoS performance module, to the final aim of collecting active and passive statistics in SDN networks, useful to operate network monitoring and fault detection.

[2] Note that F-Interop doesn't aim to test conformance and interoperability of SDN/NFV protocols (e.g., OpenFlow) but rather aims to provide tools for checking performance of SDN-NFV networks.

Finally, an F-Interop Spatial and Map Visual Representation tool aims to provide to the F-Interop users a clear representation of the network configuration and node deployment used in the tests, and statistics of collected data during performance and privacy tests. The communication between the performance test instances and visualization tool is as followed:

- F-Interop Visual Representation tool obtains the configuration information of the tests via the F-Interop EventBus.
- Per-test instances provide test statistics, logs and results through the F-Interop EventBus and the visual representation tool receive the collected data via F-Interop EventBus.

3 Performance and Privacy Tools Design Approach

3.1 Overview

This section presents a design for a common framework, which addresses all of the test scenarios, while the selection of the modules determines the applicability to the test scenario. The framework addresses both generic network testing and protocol-specific testing. Generic network testing tools allow testing IUTs in a given network condition (e.g., QoS testing), while protocol-specific testing calls for emulation of protocols, which is performed assuming the role of peers, clients or servers. Protocol specific testing allows testing of protocol scalability and QoE. A test scenario may also be implemented with a juxtaposition of tools from the two different classes.

3.2 Test Scenarios

The following Figure addresses the Performance Testing Tools topologies C1 through C5 as discussed in the Section B1.3 of F-Interop description of action (DoA) and detailed in D1.3, the initial architecture design.

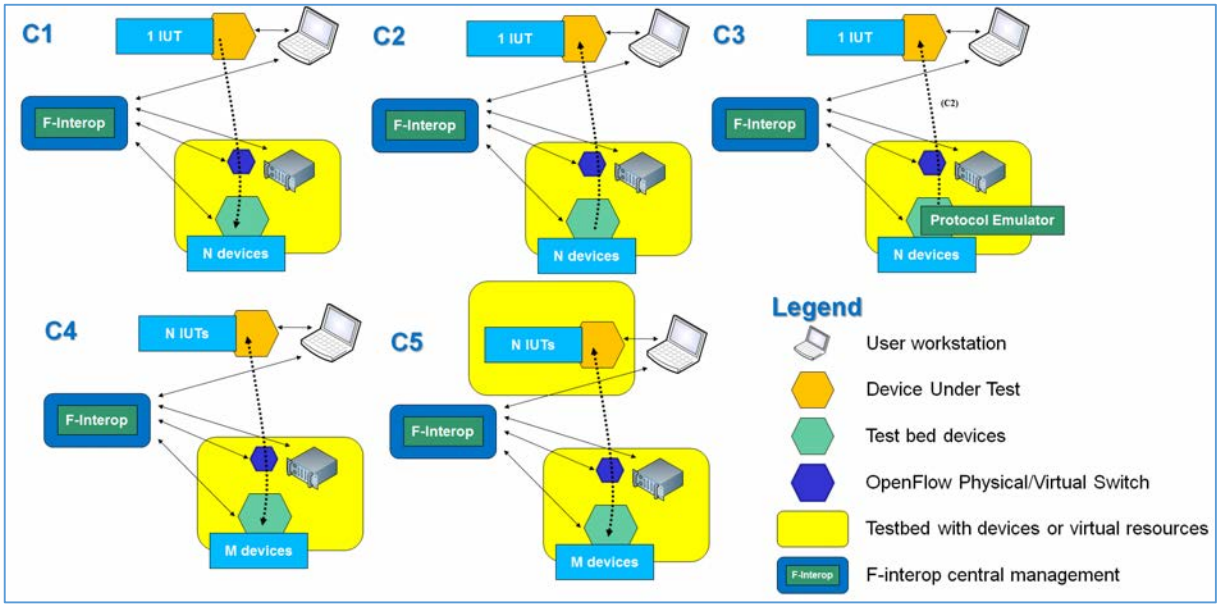

Figure 3: Test Scenarios Concept

- **C1**. The IUT is located at the vendor and interact as a client with a large number of devices running on FIRE+ connected testbeds.
- **C2**. Nodes in FIRE+ connected testbeds are programmed with to interact with the IUT. Tests verify whether this implementation can form satisfactory networks at scale.
- **C3**. The IUT interacts with virtual nodes emulated on the test server. This can useful for simple nightly non-regression tests, as well as to generate well calibrated requests in quality, size, volume and frequency.
- **C4**. The IUT and verified implementation run on the same network for example when testing that a new implementation of a routing protocol interoperates with other verified implementation.

- **C5**. Two networks running on different FIRE+ testbeds interact with one another through the Internet. One network would consist of the IUT; the other would run a verified implementation.

Note that the generic testing tools accommodate test scenarios C1, C3, C4, and C5. Scenario C2 requires protocol-specific emulation tool that can be instructed to generate the required transactions. The protocol-specific tools together with the generic testing tools provide a complete coverage of the test scenarios C1 through C5.

3.3 Generic Test Module Topology

The Figure 4 shows more detail building blocks of a performance tool. Each performance testing tool shares a generic topology. Each performance testing module has at least one network interface used for management and one or more test plane network interfaces used within the system under test.

The following diagram depicts the generic test module topology:

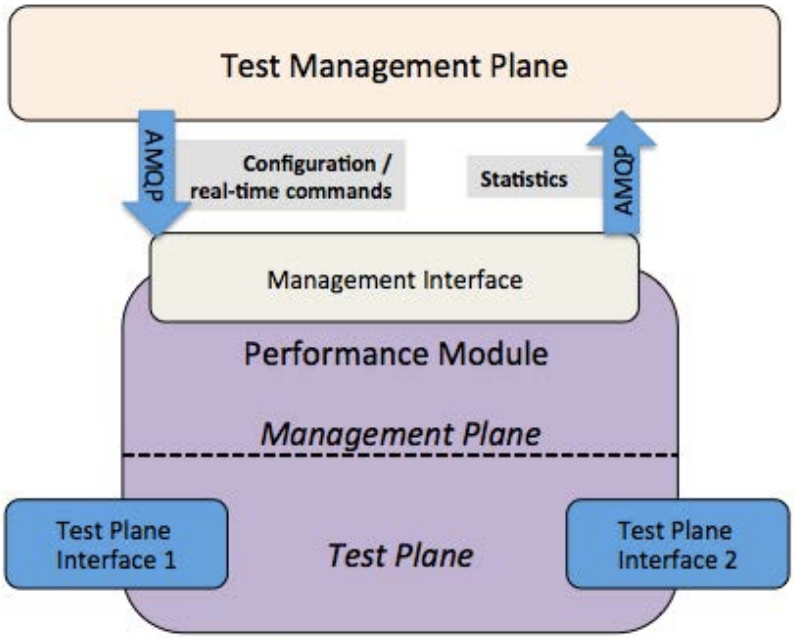

Figure 4: Generic Test Module Topology

Both management and test plane interfaces are network interfaces that are used by the performance module. For clarity, we separated the management functions and the test plane within each performance module. The management functions are used to configure the modules, to receive commands and to provide statistics. The test plane functions are module specific and their functionality varies between the modules.

3.4 Design Approach for Testing Tools

3.4.1 Scalability Testing - Passive Monitoring

The core requirements stated in D1.1 (section 3.1.5.1) for scalability testing tools are to provide a simple method of testing tool that users are able to control the scalability parameters with a real-time analysis tool. In order to fulfil the requirements stated in D1.1, the scalability testing module shall enable near real-time traffic analysis of the interactions from multiple implementations. The intent is to provide a tool for the case that none of the implementations in a given test scenario are directly under control of F-Interop. For example, a user might have developed both client and server parts and requires the analysis of the transaction rate.

The module shall provide generic conversation accounting and protocol-specific accounting. Conversation means in this context the exchange of data between two end-points using the same transmission protocol (e.g., TCP, UDP) and matching source/destination transmission protocol ports, when applicable.

The module shall provide statistics for the following:

- Network layer throughput, in bits/second. This includes all the IP header, transport layer and payload bits in each packet, but does not include the physical layer bits (i.e., Ethernet header is excluded).
- Application layer throughput (also named "Goodput"), in bits/second, defined as follows: "The number of bits per unit of time forwarded to the correct destination interface of the DUT/SUT, minus any bits lost or retransmitted."[3]
- Application layer transactional rate, if applicable to the application layer protocol.

As it is stated in the section 3.1 in D1.1, we have chosen to focus on the application layer protocol CoAP for remote scalability testing as our first step. Application layer throughput and transactional analysis will be available for supported protocols. The targeted protocol, CoAP shall be supported by the application layer throughput analysis. CoAP is a transactional protocol, thus transactional analysis shall also be included in the statistics produced by this module.

The schematic of the module is as follows:

[3] See RFC2647 (Benchmarking Terminology for Firewall Performance, 1999) section 3.17. The definition is also adopted by a current IETF benchmarking working group draft relating to data center benchmarking. (Data Center Benchmarking Terminology (Work in Progress), 2016)

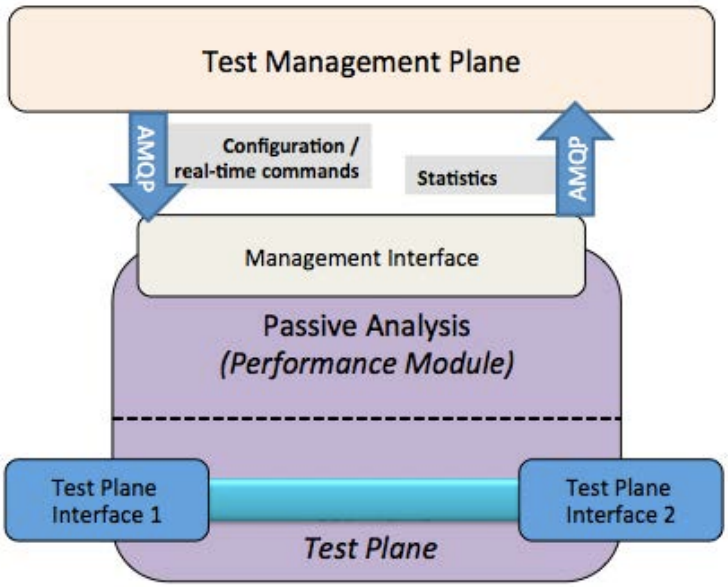

Figure 5: Passive Analysis Module Schematic

The module has three network interfaces. The management interface is used to communicate with the test management plane, receiving configuration and real-time commands and providing back aggregated statistics. Network layer and application layer statistics shall be reported separately. The other two network interfaces (network interface 1 and network interface 2) are used in the data plane of the test. The module shall forward traffic between network interface 1 and 2 (and vice versa) transparently, acting as a virtual link.

3.4.2 Scalability Testing - Active Emulation

The purpose of active emulation is the study of performance scalability, including - but not limited to - latency (round trip transactional delay), amount of endpoints (clients/servers) and transaction rate. For the active emulation, two requirements are identified in D1.1 (section 3.1.5.2) on F-Interop user side: users are able to control scalability parameters and timeline of the test.

This module emulates a single protocol endpoint or more commonly multiple protocol endpoints. For each protocol and according to applicability for the respective protocol, an endpoint may be one of the following: peer, client or server. In this stage, we consider CoAP as a targeted protocol for active emulation. CoAP is a client-server protocol. Support for other protocols may follow and extending the framework with additional protocols is a specifically enabled by the methodology.

In order to fulfil the user requirements identified in D1.1, this module shall provide the user means to setup an endpoint with protocol-specific parameters. The available endpoint settings differ according to the endpoint type. A server may only need to be configured with resources which clients can act upon. A client or a peer may include a simple atomic operation or a set of atomic operations to be performed in a loop.

For clients and peer endpoints, the module shall allow real-time changes to the scalability parameters, with the most important parameters being the number of

emulated endpoints and the per-client transaction rate. Server endpoints are commonly provisioned without change throughout a performance test.

The module shall deliver protocol-specific statistics regarding latency, throughput, successful transaction count/rate, failure count/rate and a breakdown by error codes if applicable.

The schematic of the module is as follows:

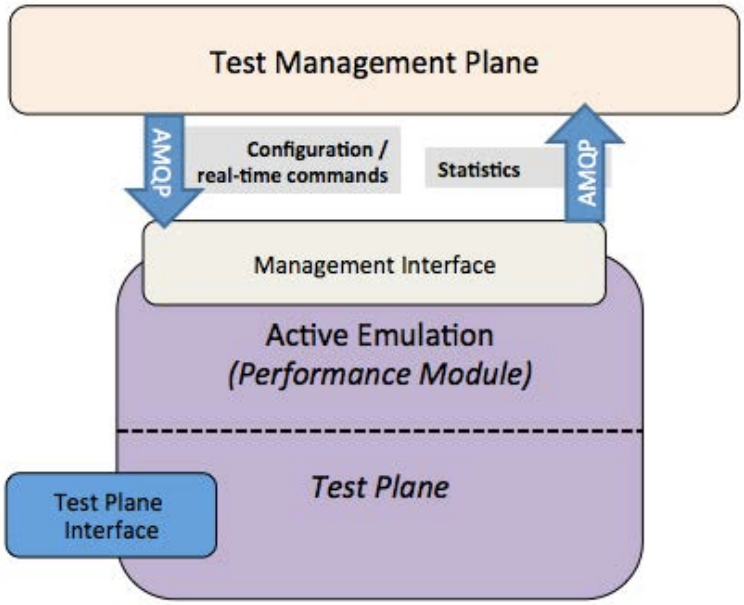

Figure 6: Active Emulation Module Schematic

The module has two network interfaces. The management interface is used to communicate with the test management plane, receiving configuration and real-time commands and providing back protocol-specific statistics. The network interface in the data plane of the test may be assigned multiple IP addresses if deemed necessary.

3.4.3 Resiliency Testing - Network Impairment

As described in D1.1, in the scope of F-Interop, we have chosen to focus on relevant scenarios applicable to IoT and implement the emulation of network conditions in software for the online resiliency testing. Infrastructure failures are out of scope for F-Interop.

Network impairment provides an emulation of sub-optimal network conditions, such as packet loss, packet duplication, latency and latency variance. The application of network impairment provides possibility for the resiliency testing of the IoT devices which are expected to operate in such degraded network conditions.

The main requirements identified in D1.1 (section 3.2.5) are to provide users to control the setting of impairment and to check the real-time statistics of the tests. In order to fulfil the requirements that stated in D1.1, this module shall provide the user with means to apply network impairment on a link between two components, whether they are located at the same test bed, different test beds in the test bed federation or located on user premises, as long as at least one component is located in a test bed in the test federation.

The module shall provide means to configure different impairment profiles applied to specific traffic flows identified by traffic classifiers. The traffic classifier is a set of parameters that can be matched in the ingress packets to identify a flow, for example IP addresses or port numbers. The impairment profile is a set of impairment parameters applied to packets matching a traffic classifier, for example delay or packet loss ratio.

The impairment module provides a list of impairment parameters and a list of traffic classification parameters available for configuration via interactive user interface or automated test suites. The module is instantiated in the test bed, and the network configuration (connectivity to the endpoints) is provided by the TBaaS layer.

The schematic of the module is as follows:

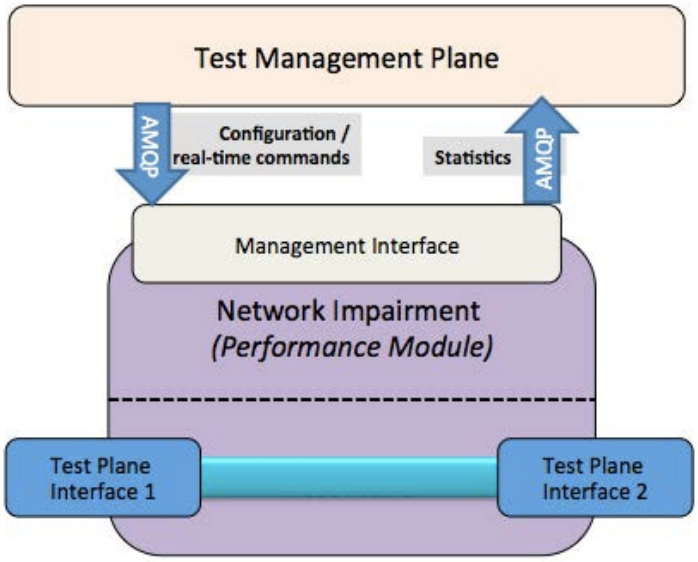

Figure 7: Network Impairment Module Schematic

The module has three network interfaces. The management interface is used to communicate with the test management plane, to receive configuration and real-time commands and to provide statistics back in the granularity level of a profile. The other two network interfaces (network interface 1 and network interface 2) are used in the data plane of the test. The module shall forward traffic between network interface 1 and 2 (and vice versa) transparently, acting as a virtual link. Data that does not match to any profile shall be forwarded and not dropped. Data matching a profile shall be altered, duplicated or delayed per profile configuration as set by the user.

3.4.4 Energy Consumption Testing

Provision of energy consumption or energy efficiency modelling will be highly beneficial for IoT devices. As it is described in D1.1 (section 3.4.5), these tools aim to provide energy consumption measurements or estimations. Two different classes of energy consumption test tools have been identified:

- Platform specific energy consumption modelling, as reported by an IUT. This requires both the support from the IUT and an adapter to collect the data from the IUT.
- Generic energy consumption modelling, derived from traffic statistics and estimated with a microwatt per bits/s (µW/bits/s) provided by the user.

If the platform-specific energy consumption measurement is desired, the energy consumption sampler module described below shall be included in the test setup. The module will communicate with the IUT via separate interface to retrieve the energy consumption statistics.

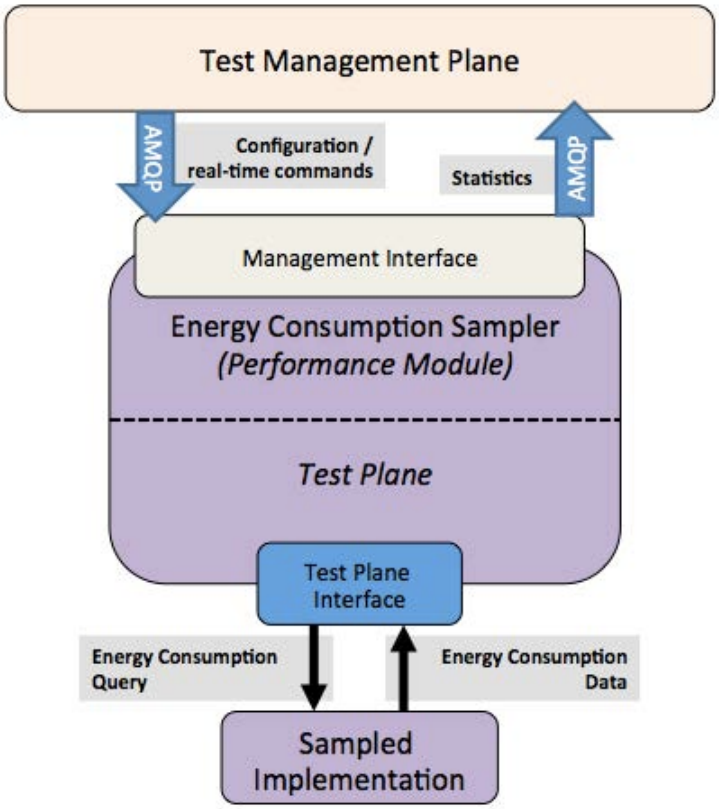

Figure 8: Energy Consumption Sampler Module Schematic

The module has two network interfaces. The management interface is used to communicate with the test management plane, receiving configuration and real-time commands and providing back energy consumption statistics. The network interface in the data plane of the test is used to communicate with a sampled implementation (the targeted IUT) in order to retrieve the energy consumption data from it.

For the generic energy consumption modelling, the estimation will be performed by the passive monitoring tool described above. Its functionality shall be extended to apply energy estimation calculation to the measured traffic statistics for the flows associated with the IUT. In this case, the performance measurement module must be included in the test and configured with the metadata information of the IUT providing parameters for the calculation model.

The statistics provided by platform-specific measurement and generic estimation shall be treated as distinct types of measurements and transmitted as separate statistic items.

3.4.5 QoS Testing - Monitoring of SDN/NFV-Based Networks

As stated in D1.1, QoS SDN/NFV-based testing is to qualify and quantify QoS performances of a given system. As SDN paradigm is emerging as a promising solution for simplifying network management, F-Interop provides QoS tests based on

SDN/NFV architecture for collecting statistics related to node and links, and NFV approach for emulating network conditions. We choose synthetic measurement as it provides a tool for monitoring device and network conditions by leveraging on the Software Defined Networking (SDN) approach.

The module identifies network degradation considering the quality of experience (QoE) perceived by users (such as host/device) while they are using a service. A Service Level of Agreement (SLA), that defines the acceptable network performance level, is defined through thresholds.

Once the SLA conditions is not respected, the module tries to isolate the cause of the problem by combining passive statistics collected by the SDN Controller and active measurements, such as end to end latency, calculated by agents connected to the network nodes.

As defined in chapter 3.5.5 of D1.1 both FI-User and FI-Contributor should be able to configure the module in order to provide the topologies scenarios he/she want to test. Moreover the module should provide a configuration interface to:

- Set thresholds, which define, as mentioned before, the SLA for a given service.
- The number of agents required for actively monitoring the network.
- The information to access the SDN Controller (credential, location).

The module can be executed as remote resource in the testbed or downloaded and run in the local testbed of the end user. The schematic of the module is as follows:

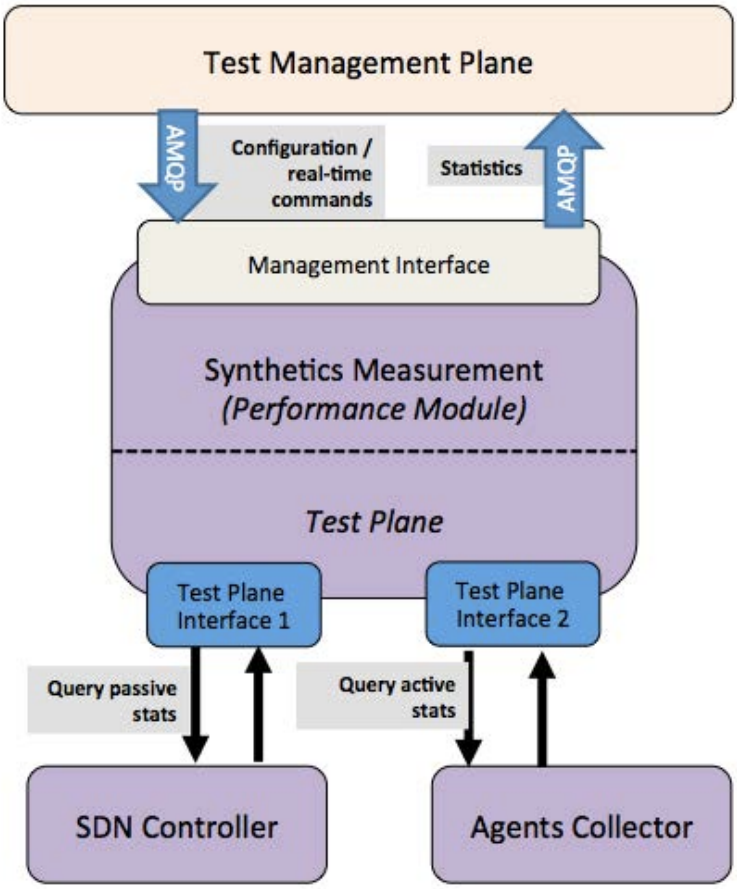

Figure 9: Synthetics Measure Module Schematic

The module has two software interfaces that communicated with the SDN controller and the agents' collector. The management interface is used to communicate with the test management plane, receiving configuration and real-time commands and providing statistics back regarding the state of the network. The other network interface is used in the data plane of the test. The module shall read both passive statistics, collected by the SDN Controller, and active measurements, collected by an Agent central repository, namely Agent Collector (AgCo). By combining active and passive measurements, the module shall evaluate QoE performance in the network, identify potential underperforming link, and define a mitigation strategy to restore the quality of the Service (QoE) according to the SLA (defined by a threshold).

3.4.6 Privacy Risk Testing - Data Leakage Analysis

The integration of IoT devices/sensors, such as the F-Interop System Under Test (SUT) (see definition in D1.1) may lead to concerns about both security and privacy aspects. This is due to the fact that those devices can process sensitive data (privacy issues) using insecure wireless channels (security issues).

Regarding the security aspect, according to D1.2 this was the key aspect adopted in the design of F-Interop architecture. Secure communication protocols, data encryption and the use of the AMQP Event Bus should provide a good level of security. Therefore, this aspect is considered provided a priori from the platform and not anymore investigated by this module.

The module will focus on privacy aspects to prevent leaks of sensible or confidential data while tests are performed. As it is stated in D1.1, we mainly focus on traffic analysis techniques, aiming to extract not only user/node location, but also other personal information.

Following the identified requirement in D1.1 (section 3.3.5), FI-User must have access to the data exchanged on the encrypted channel between the IUT and the FI-Platform. This module enables to perform traffic analysis on encrypted data exchanged between one local IUT and other remote ones. The intent is to provide a tool that investigates what kind of information can be inferred in case a malicious observer manages to sniff encrypted traffic generated while running a test on the F-Interop platform. For example, a user might want to test an implementation against other resources remotely connected using F-Interop. Even though all F-Interop communications should be secured by Transport Layer Security (TLS) protocol encryption, an attacker by analysing the traffic exchanged may identify what type of test the user is performing, and even worst, the result of the test (passed or failed), which should be confidential.

The module shall analyse the encrypted data between two end-points using the same transmission protocol (encrypted AMQP packets using TLS). The module is instantiated in the test bed.

The schematic of the module is as follows:

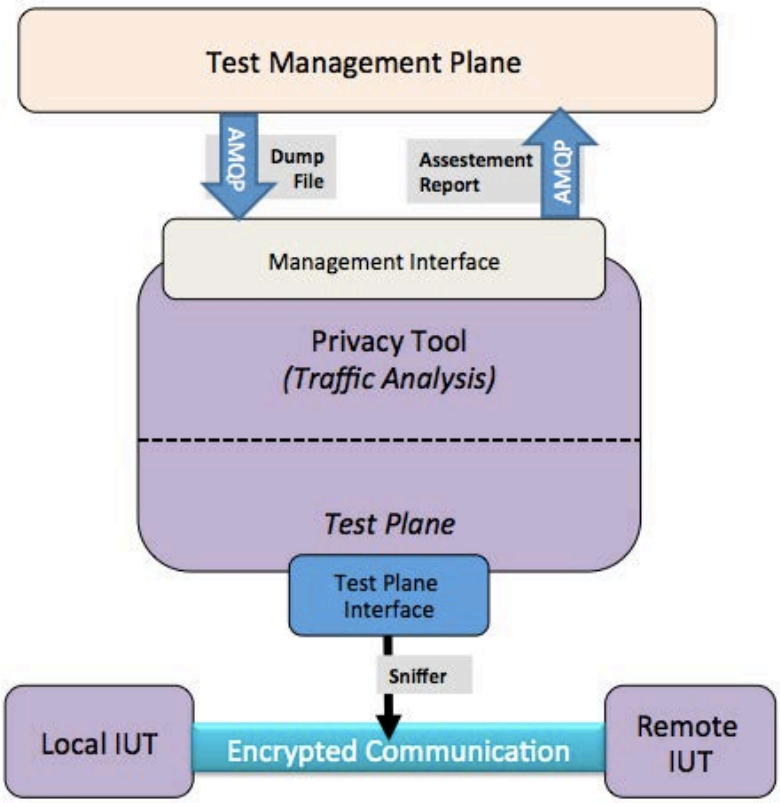

Figure 10: Privacy Module Schematic

The module has two network interfaces. The management interface is used to communicate with the test management plane, to receive configuration and real-time commands and to provide aggregated statistics back. The network interface in the data plane of the test is used to sniff encrypted traffic generated by a local IUT tested against others, remotely connected.

4 Framework Functionality

The framework for the performance tests and privacy tests is designed to fulfil the requirements of the different test scenarios as well the different classes of performance tests. The detail functionalities are described in the following subsections.

4.1 General Functionality

4.1.1 Common Functionality

To enable a coherent test environment, we have defined a common functionality that shall be provided by each module.

Test configuration: Each module shall provide a set of configuration items. This set is dependent on the objective of the module and shall be specified for each module.

A module shall provide a manifest describing the configuration commands in a machine readable form. The configuration manifest is used by the F-Interop GUI to generate the configuration items.

These commands are used to configure the initial state of the module before starting a test, for example, configuration of IP addresses, command sequences, filters and profile names.

Test control: Each module shall provide a function to control its running state in the test. The running state is aligned for all modules for a certain test instance. It is used to signal that the test starts or that the test is stopped (gracefully or non-gracefully):

- Before "START" is received, only configuration is performed. Real time commands are not valid in this stage and no statistics are provided. Before "START" command is received, only "START" and "ABORT" signals are valid.
- When "START" is received, the module becomes active in the test plane. Configuration commands are not valid for this state while real time commands are valid. Statistics are provided once the test started. After "START" is received, only "STOP" and "ABORT" signals are valid.
- When "STOP" is received, the module shall gracefully stop. Configuration and real time commands are not valid and statistics are provided until the module completely stopped. It is possible to abort a test that is in stopping state by issuing the "ABORT" signal. After "STOP" is received, only "ABORT" signal is valid.
- When "ABORT" is received, the test shall immediately stop and all resources of the test instance are released.

The stimulus for starting, stopping or aborting the test is user action on the F-Interop GUI.

Statistics: Each module shall provide real-time statistics with data applicable to the functionality of the module. The statistics stream shall be described in a module manifest.

Statistics shall be stream upon test start, i.e., when the START command is issued). If STOP command is used, the statistics will continue to be streamed until the module has shut down completely. If ABORT is used, the statistics streaming will be cease immediately.

A module shall provide a manifest describing the statistics metadata in a machine readable form. The statistics manifest is used to generate templates to be used by the visualization tool.

Real time commands: A module shall provide a manifest describing the real-time commands it provides in a machine readable form.

These commands are used to control parameters in real-time, for example, transaction rates (protocol emulator) or latency values (impairment generator).

The following table provides an overview of the common module functionality:

Scope	Functionality
Test Configuration	Module specific
Test Control	• START: Start the test • STOP: Stop the test gracefully • ABORT: Stop the test without delay (non-gracefully)
Module Manifests	• Configuration manifest • Real-time command manifest • Statistics manifest

Table 1: Framework Modules - Common Functionality

4.1.2 Timeline Controller Module

This module shall be instantiated once for each test instance and therefore considered an integral part of the general functionality of the framework.

Test control: The test control commands are issued by the FI-User via the F-Interop GUI. This module shall forward the commands toward all the modules involved in a particular test instance.

The module shall observe the semantics for the test control signals and forward configuration to the modules only before the test start is signalled, and real-time commands only after test was started and not yet stopped.

When a test is stopped, the module shall wait for all other modules to finish their graceful shutdown and once all modules have reported back as completely stopped. The module shall then signal the completion of the test instance toward the F-Interop GUI.

Test configuration: The configuration for this module is composed of timeline definitions. The timeline controller shall support multiple timelines. All of the timelines are aligned to the beginning of the test (when the test start is signalled).

The module shall handle the setting of real-time parameters according to a user pre-configured timeline.

The timeline shall be divided into segments. Each segment is configured with an initial value, end value, segment duration (in seconds) and interpolation method (linear, exponential and others).

Statistics: The module shall forward statistics streamed from performance modules towards the FI User. The module shall provide a possibility for the FI User to include or exclude specific statistics to be forwarded. The statistics are processed by the visual representation tool that detailed in the Section 5.3.

The module may aggregate the statistics, if more than one module instance of the same type is used. For example, when multiple instances of protocol emulation are used, all of the statistics are streamed together and the module ensures the aggregated statistics are coherent in relation to the test execution time.

The following table provides an overview of the module's functionality:

Scope	Functionality
Test Configuration	Timeline definition
Test Control	• START: Start the test • STOP: Stop the test gracefully • ABORT: Stop the test without delay (non-gracefully)
Statistics	Forwarded from modules towards the F-Interop EventBus. May aggregate statistics. FI User may select the statistics to be forwarded to it.
Timeline Control	Support multiple timelines. Each timeline is divided into segments. Each segment is defined with the following: • Initial value • End value • Segment duration • Interpolation method

Table 2: Framework Modules - Timeline Controller Functionality

4.2 Performance Tools Functionality

4.2.1 Scalability: Passive Monitoring

The passive monitoring tool enable near real-time traffic analysis of the interactions from multiple implementations.

Test configuration: This module shall provide named profile configuration support.

The named profile shall be a unique key that associates a set of **traffic classifiers** (filters). Statistics generated by this module shall be aggregated by the named profile key. The profile name shall be selected by the F-Interop User.

This module shall accept filter configuration, supporting the following identifiers:

- Protocol family: IPv4 and IPv6
- Matching source IP addressing
- Matching destination IP address
- Matching the lower layer protocol (Protocol field for IPv4; Next Header for IPv6), i.e., TCP, UDP, ICMP
- Matching of source and/or destination port numbers, where applicable (UDP/TCP).

Protocol configuration: The module shall provide both generic protocol analysis and protocol specific analysis.

In the generic protocol case, the module shall support the IP protocols (IPv4, IPv6) and transport protocols (TCP, UDP).

Protocol specific analysis is provided as an extension of the generic analysis. At this point we are targeting the IoT protocol CoAP and other protocols may be added as extensions to the module.

Energy consumption estimation: Based on the requirement of D1.1 (section 3.4.5), this module must provide energy consumption estimation functionality, available as an alternative to the platform-specific energy consumption measurement (which is described in the section 4.2.4 below). The estimated energy consumption statistic is calculated based on the traffic statistics and a device-specific estimated consumption, specified as a parameter in terms of µWatts per bits/s. The estimated statistic is provided as a time-series and is distinct from the platform-specific measurements.

Statistics: The module shall provide statistics per interval for each of the named profiles. The module shall deliver statistics for generic and specific protocols (see *protocol configuration* section above). The module shall deliver generic energy consumption utilization if selected by the user based on traffic statistics. The module shall provide the statistics at a granularity of a named profile.

The following statistics shall be provided for generic protocols per interval:

- Number of active nodes observed in the interval
- Number of active conversations in the interval
- Number of packets per interval
- Network layer throughput in the interval
- Estimated energy consumption

For CoAP, the following statistics shall be provided per interval:

- Total transactions
- Successful transactions
- Failed transactions
- One-way transactions (when no response from server is observed)
- Statistics per CoAP response code (e.g., 2.05, 4.04)
- Application layer throughput ("Goodput")

Real time commands: There are no real-time commands for this module.

Note that statistics are streamed as soon as the test START signal has been indicated and cease to be streamed at the test STOP/ABORT according to the semantics described for the common functionality (section 4.1.1).

The following table provide a summary of the functionality provided by this module:

Scope	Functionality
Test Configuration: Filters	• Protocol family: IPv4 and IPv6 • Matching source IP addressing • Matching destination IP address • Matching the lower layer protocol (Protocol field for IPv4; Next Header for IPv6), i.e., TCP, UDP, ICMP • Matching of source and/or destination port numbers, where applicable (UDP/TCP)
Generic Protocol Support	The generic protocol support encompasses: • IP Protocols: IPv4, IPv6 • Transport protocols: TCP, UDP The following statistics shall be provided for generic protocols per interval: • Number of active nodes observed in the interval • Number of active conversations in the interval • Number of packets per interval • Network layer throughput in the interval
Protocol Specific Support	CoAP: • Total transactions • Successful transactions • Failed transactions • One-way transactions (when no response from server is observed) • Statistics per CoAP response code (e.g., 2.05, 4.04) • Application layer throughput ("Goodput")[4]

Table 3: Framework Modules - Passive Monitoring Functionality

4.2.2 Scalability: Active Emulation

Active emulation is strictly dependent on the specific protocol that is targeted. At this point, we selected CoAP as the target protocol for active emulation. It shall be possible to extend the following module to support other protocols or to create new modules based on the design and functionality of the CoAP module (e.g., for HTTP).

[4] See **Erreur ! Nous n'avons pas trouvé la source du renvoi.** for description of "Goodput".

Active emulation shall be supported for server-side and client-side. Server-side emulation and client-side emulation may be integrated into a single module.

4.2.2.1 Scalability: Active Emulation - CoAP Client Emulation

To emulate CoAP clients, we selected a well-known implementation: the Eclipse Californium Java-based JDK (Eclipse Californium).

This module shall provide named configuration profile support. The named profile shall be a unique key that associates a set of **clients**. Each client shall have an associated command sequence. Statistics generated by this module shall be aggregated by the named profile key. The profile name shall be selected by the F-Interop User.

Configuration Profile:

This module shall accept a set of configuration data prior to execution of the test. The configuration profile of the clients shall include:

- the number of emulated clients
- IPv4/IPv6 address set/range to be used as the source address
- Optional port number set/range to be used as the source port. If omitted, default random port number assigned by the underlying OS shall be used.
- A sequence of CoAP commands to be executed by each client.

The commands in the sequence shall contain following parameters:

- CoAP primitive. GET, POST, PUT, DELETE and OBSERVE primitives shall be supported. The primitive are described in RFC 7252 (The Constrained Application Protocol (CoAP), 2014) (GET, POST, PUT, DELETE) and RFC 7641 (Observing Resources in the Constrained Application Protocol (CoAP), 2015) (OBSERVE).
- URI
- Payload in case of POST and PUT primitives
- Expected return code
- Command execution timeout.

Real-time commands: The module shall allow controlling the rate of attempted transactions and the target number of emulated clients during the execution of the test.

This module shall provide the following real-time commands:

- Set target rate of attempted transactions
- Set target number of emulated clients

Note: that the actual transaction rate and the number of emulated clients may lag behind the target numbers, depending on resource availability and the performance of the IUT. The actual achieved transaction rate and the number of emulated clients shall be provided by the module in the statistics.

Statistics: This module shall provide aggregated statistics as well as per profile statistics.

The following statistics items shall be provided:

- Traffic rate, (incoming/outgoing, in bits/s)
- Transaction rate, (attempted/successful/failed, in transactions/s)
- Transaction round-trip time (minimum/average/maximum, in milliseconds)
- Response codes (total number and rate for each code)
- Number of active clients *(note: this is distinct from the target number of emulated clients, which is a parameter)*.

The following table provide a summary of the functionality provided by this module:

Scope	Functionality
Test Configuration	• Named profile • Network configuration • Command sequencing
Real-time commands	• Rate of attempted transactions • Target number of emulated clients
Statistics	• Traffic rate • Transaction rate • Transaction round-trip time • Response codes • Number of active clients

Table 4: Framework Modules - CoAP Client Emulation Functionality

4.2.2.2 Scalability: Active Emulation - CoAP Server Emulation

To emulate CoAP servers, we selected a well-known implementation: the Eclipse Californium Java-based JDK (Eclipse Californium).

This module shall provide named profile configuration support. The named profile shall be a unique key that associates a set of **servers**. Each server shall have an associated resource description. Statistics generated by this module shall be aggregated by the named profile key. The profile name shall be selected by the FI User.

The servers shall implement the CoAP primitives GET, POST, PUT, DELETE and OBSERVE. The primitive are described in RFC 7252 (The Constrained Application Protocol (CoAP), 2014) (GET, POST, PUT, DELETE) and RFC 7641 (Observing Resources in the Constrained Application Protocol (CoAP), 2015) (OBSERVE).

Test configuration: This module shall accept an address range to be used for multiple servers. Optionally, this module shall accept a source port configuration. When a source port is not specified, the default source port for CoAP shall be used (5683) (Service Name and Transport Protocol Port Number Registry).

This module shall accept a resource description for a given named profile. A resource is defined by a URI path and content. The content shall be allowed to be customized by the user. The module shall also provide ready-made contents to allow rapid testing. The ready-made contents shall be differentiated by content size (in bytes).

Real-time commands: This module shall allow modification of resources to facilitate the OBSERVE primitives.

Statistics: This module shall provide aggregated statistics as well as per profile statistics.

Note that the servers cannot calculate a transaction round trip time since they are unaware of the time when the request was initiated and the time the request was delivered to the client.

The following statistics items shall be provided:

- Traffic rate, (incoming/outgoing, in bits/s)
- Transaction rate, (attempted/successful/failed, in transactions/s)
- Response codes (total number and rate for each code)
- Number of observed clients

The following table provides a summary of the functionality provided by this module:

Scope	Functionality
Test Configuration	• Named profile • Network configuration • Server resources
Real-time commands	• Modification of server resources
Statistics	• Traffic rate • Transaction rate • Response codes • Number of observed clients

Table 5: Framework Modules - CoAP Server Emulation Functionality

4.2.3 Resiliency: Network Impairment

Many IoT devices are expected to operate in sub-optimal network conditions, for example due to lossy nature of the wireless connections and aggressive power management of the devices themselves. In order to test IUT's resiliency and correct function of the protocols in poor network conditions, the FI Platform must provide means to simulate network impairments. In accordance with the requirements of D1.1 (section 3.2.5), the impairment function shall provide multiple impairment models.

The QoS impairment module provides the means to emulate sub-optimal network conditions, including packet loss, duplication, delay and delay variation. A set of impairment parameters applied to the traffic constitutes an **impairment profile**. The FI User must have the possibility to enable, disable and modify the impairment profile in real-time during the test. The parameters of the impairment profile must also be able to be dynamically modified by the Timeline Controller.

The module must support a convenient method to associate a certain traffic flow with the desired impairment emulation, using named profile configuration. The named profile is a unique key that associates **traffic classifiers** with **impairment profiles**. The traffic classifiers are a set of attributes identifying packets of a flow, such as IP

address, transport protocol and other values that can be matched in the packet header or payload. The design of the impairment module should provide a possibility to extend both traffic classifier and impairment profile definitions in the future.

The impairment module must collect the **impairment statistics** and deliver them to the Timeline Controller in real-time. The module provide to the FI User the possibility to define which of the available statistics are collected and delivered by the module as stated in D1.1.

We selected a well-known implementation of network impairment: Linux NETEM (Linux NETEM). Linux NETEM uses the standard Linux traffic control mechanism. Linux Traffic Control and Linux NETEM provide the functionality on the test plane and this module provides the test management plane.

Traffic classifiers: The module shall perform classification using the Linux traffic control mechanism using a classifier called *tc-u32*.

The following list describes the traffic classifiers that the module shall provide:

- Matching protocol family: IPv4 or IPv6
- Matching source IP address
- Matching destination IP address
- Matching the DSCP/ECN field (IPv4 specific)
- Matching the IHL field (IPv4 specific)
- Matching the transport layer protocol (Protocol field for IPv4; Next Header for IPv6), i.e., TCP, UDP, ICMP
- Matching ICMP types and/or ICMP codes
- Matching of source and/or destination port numbers
- Matching fragmentation options
- Matching of traffic class (IPv6 only)
- Matching of flow label (IPv6 only)

The module must ensure the configuration is valid. For example, if ICMP type/code is used as a match filter, a protocol filter must also be used and set as ICMP. If a source/destination port number is matched, a suitable protocol must be matched (e.g., TCP or UDP).

Impairment profiles: The impairment module shall provide impairment parameters, as described in the following list:

- Delay (in milliseconds, constant transit delay applied to all matched packets)
- Delay variation (in milliseconds, uniformly distributed random delay)
- Packet loss (as percentage from the total number of matching ingress packets)
- Packet duplication (as percentage of matched ingress packets)

Impairment statistics: The module shall provide statistics for the traffic that matched a traffic profile. Statistics generated by this module shall be aggregated by the named profile key. The profile name shall be selected by the FI User.

The module shall provide the following statistics fields per profile:

- Matched number of packets
- Matched number of bytes
- Number of dropped packets

- Number of duplicated packets

Real time commands: This module shall provide commands to change the impairment parameters in real-time, per named profile. The commands shall allow changing all the impairment parameters used in the test configuration (delay, delay variation, packet loss and packet duplication).

The following table provide a summary of the functionality provided by this module:

Scope	Functionality
Test Configuration: Filters	• Matching protocol family: IPv4 or IPv6 • Matching source IP address • Matching destination IP address • Matching the DSCP/ECN field (IPv4 specific) • Matching the IHL field (IPv4 specific) • Matching the lower layer protocol (Protocol field for IPv4; Next Header for IPv6), i.e., TCP, UDP, ICMP and so on. • Matching ICMP types and/or ICMP codes • Matching of source and/or destination port numbers • Matching fragmentation options • Matching of traffic class (IPv6 only) • Matching of flow label (IPv6 only)
Test Configuration: Impairment Parameters	• Delay • Delay variation • Packet loss • Packet duplication
Real-time commands	• Per-profile parameter change • Enable/disable impairment profile
Statistics	• Matched number of packets • Matched number of bytes • Number of dropped packets • Number of duplicated packets

Table 6: Framework Modules - Network Impairment Functionality

4.2.4 Energy Consumption: Platform-specific Measurement

This module provides an implementation-specific energy consumption sampling. This module shall provide named profile configuration support. The named profile shall be a unique key that associates a set of **target implementations** (IP address based). The profile name shall be selected by the FI User.

Test configuration:

Implementation-specific support: The module requires explicit support for each implementation target, since the APIs to interface and retrieve the data vary and may

be provided using different protocols altogether (for example, one implementation allows retrieving the data using the command line interface, while another provides the information over a web interface).

We are currently targeting the following implementations, which are IoT-specific:

- Contiki OS[5]
- OpenWSN[6]

Support for additional implementations may be added at a later stage by extending the module.

Note that the functionality of the energy consumption **estimation** is provided by the passive monitoring module described in the section 4.2.1.

Real-time commands: There are no real-time commands for this module.

Note that statistics are streamed as soon as the test START signal has been indicated and cease to be streamed at the test STOP/ABORT according to the semantics described for the common functionality (section 4.1.1).

Statistics: The module shall provide the statistics at a granularity of a named profile. The statistics shall provide the energy consumption in the interval.

Scope	Functionality
Test Configuration	• Target implementation list
Statistics	• Energy consumption

Table 7: Framework Modules - Energy Consumption Functionality

4.2.5 QoS Monitoring in SDN/NFV-Based Networks

This module exploits SDN capabilities to perform network monitoring and fault detection. It provides a graphical and analytical overview of the network, focusing on the Quality of the Experience (QoE) perceived by an end user, connected to a SDN network. The module is based on the SDN-POC (Proof of Concept) developed by the University of Luxemburg.

The module aims to check the SDN network performances by combining passive and active measurements gathered respectively from the SDN controller and agents. The approach is based on revealing any network degradation, from the user point of view (QoE), leveraging on the SDN centralized view of the data plane. (See section 4.2.5.1)

Based on this work, the QoE Synthetic Measurements shall provide a tool to simulate and/or evaluate the performance of a specific SDN network topology.

---

[5] Contiki OS (Contiki OS) provides means to estimate power consumption.

[6] Currently power estimation is not yet implemented for OpenWSN (OpenWSN Wiki) ; it is however on the roadmap. The founder and coordinator of the project (Thomas Watteyne, INRIA) is a member of the F-Interop consortium.

The simulation will be possible by using Mininet (Mininet), an emulation network environment widely used in the research community that creates realistic virtual network running real kernels, switch and application code. Mininet uses Open vSwitch, an open source software that supports OpenFlow. By using these tools, the F-Interop user can define, and thus simulate the network topology that he/she would like to test. Moreover, Mininet allows customizing a given interface, setting up link parameters, such as link capacity, link packet loss, etc. An instance of that will be available in the testbed if required by the user.

The module will be able to gather all the information about the network topology by querying the SDN controller, the brain of an SDN network that besides controlling the data plane, it offers APIs to implements software network services. The module uses a specific controller implementation, OpenDayLight (ODL), an open source project, under the Linux Foundation, offering a community-led and an industry-supported framework, including support for OpenFlow protocol. The topology information and the per-port statistics owned by the controller allow determining the end to end path followed by each data flow in the network (included the probe requests sent by the agents). A pre-configured instance of this controller will be available in the testbed.

The monitoring and fault detection module leverages on e2e latency measurements, perceived by end users (hosts) while accessing a service available on a given server. Those (active) latency measurements are generated by agents, namely QoE agents, which produces some ping requests from the specific vSwitch where they are connected, to the target server. Such measures represent the experience that an user would have seen if connected to that specific point of the network.

In the SDN-RADAR POC (Gheorghe, Avanesov, Palattella, Engel, & Popoviciu, 2015) a on the shelf solution, called v6Sonar[7], was used. The v6Sonar platform provides not only the agents but also a centralized collector (v6Sonar controller) which keeps the history of the probes, and it offers a console and API to fetch those data. The module will use for the first iteration the same solution, providing those agents as resources in the testbed, and letting the user instantiating and attaching them to specific switches in a given topology (using tunnels).

Erreur ! Nous n'avons pas trouvé la source du renvoi.Erreur ! Nous n'avons pas trouvé la source du renvoi. shows the main building blocks of the monitoring tool described above. If the e2e delay perceived by the end users is higher than what expected (according to SLA), it is possible to conclude that some links in the network may be underperforming (e.g., they are congested). A heuristic method has been defined to identify the links which may be experiencing an issue (which impacts the whole network). The accuracy of the fault detection method can be improved considering not only the e2e delay measured by the agents, but also per links, and per ports statistics collected by the controller.

The monitor and fault detection tool will be further extended including a mitigation module, which allow redirecting the traffic, avoiding the underperforming links.

[7] v6Sonar is considering to release the agents as open-source. In any case, we plan to simulate the agents' behavior with scripts, integrated in the SDN-based QoS tools.

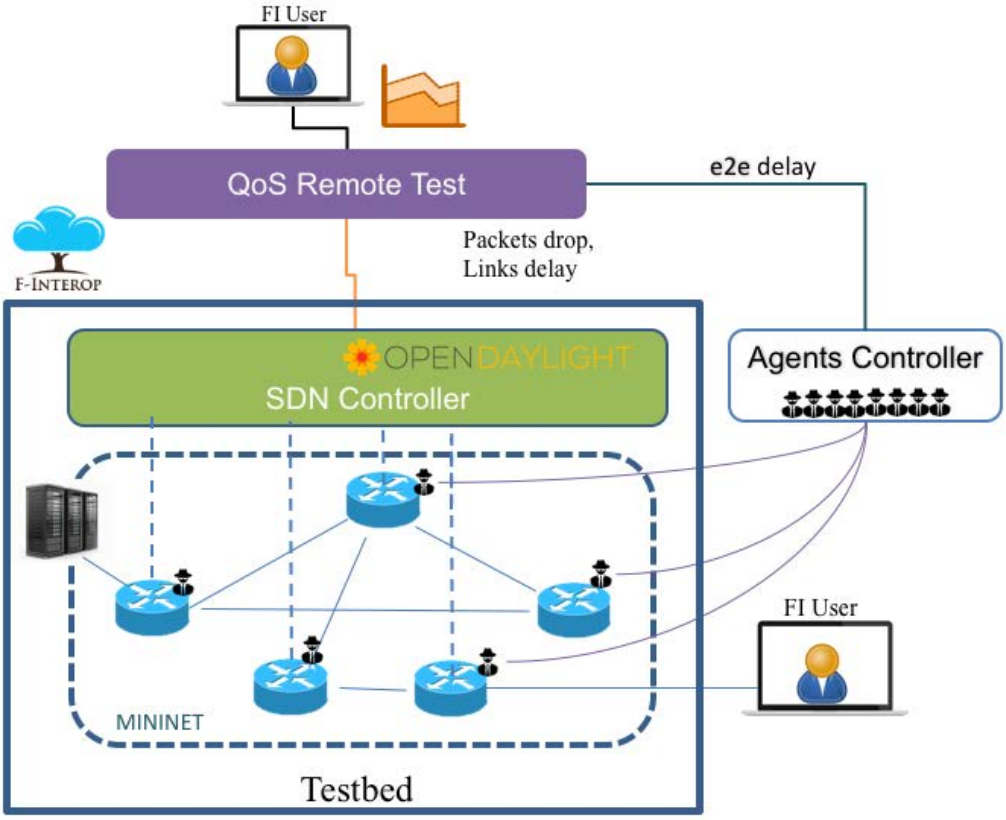

Figure 11: QoS Monitoring Tool

The Synthetic Measurement module shall provide a web graphical interface to visualize the network topology, the agents, and display all the available information about the status of the network, and the achieved performances.

Some API will also be released for third party use, such that the modules can be used for testing a real or a simulated SDN network.

Considering the requirements defined in section 3.5.5 of D1.1 the functionality provided by this module are described in the following table:

Scope	Functionality

Module Configuration:	The module MUST provide a configuration page where all the components involved SHOULD be configured:
	• Network Topology (to instruct the simulator)
	• Number of agents required
	• Access to the SDN Controller:
	  o Credential
	  o Location IP
	The configuration shall be provided by the FI-User or FI-Contributor.

Real Time Commands	This module shall provide a web user interface that will graphically display the overall network status. It will allow to configure: • networks parameters (e.g., average link

	bandwidth, delay, packet drop) • Thresholds for QoE metrics (according to SLA) • Start/Stop measurements • Activate/Disable Agents
Statistics	The module shall deliver statistics about the computed information collected by the controller and the agents: • Link Bandwidth (in/out) • Node Packet Loss • Link Packet Loss • End to End Latency

Table 8: Framework modules - QoE Synthetic measurement

4.2.5.1 The troubleshooting QoE approach

The troubleshooting algorithm used by this module is quite unique. It starts from a user perspective and progressively works its way towards the source of the fault. Normally the way users experience a web-based distributed service should be homogeneous from all locations. However, users experience the service conditioned by the quality of the underlying network links, which varies across domains and communication protocols. It is possible to put a threshold over the acceptable end-user perceived degradation, and start troubleshooting once such a threshold is surpassed. The choosing of such a threshold value can stem from an existing service level agreement (SLA). SLAs are usually specifically tailored to the service context and define limits of service delivery quality. A possible example, the average speed to answer a request is a common metric; some SLAs in VoIP contexts can refer to notions such as the “expectation factor” the amount of service degradation that users accept in service quality, in exchange for service access. Since the agents offered by the Sonar tool were already available, our solution is novel in that it combines Sonar’s continuous monitoring capability, with the central and local network knowledge delivered by the SDN infrastructure, in order to assist in the fault localisation process.

4.3 Privacy Tools Functionality

Privacy module will allow checking the level of protection against a particular eavesdropper. Even though the F-Interop platform has been designed to keep all the information secure through the use of special protocols and methodologies, there are still techniques that allow inferring information exchanged in encrypted communications. The confidentiality of a test is one of the main requirements in F-Interop and it is also the most important aspect required by a User that wants to use such platform.

Based on this consideration, the module will provide an assessment of what kind of information (and its granularity) can be extrapolated from the analysis of the encrypted traffic exchanged among IUTs remotely connected, or simple from one IUT

and the central platform. It aims to infer information about the content of the encrypted connections by observing patterns of data flows.

The module, based on (Panchenko, et al.), utilizes meta-information, such as packet size and direction of the traffic, without breaking the encryption. Traces collected during different test execution will be used for identified and classified the distinct classes of test. A class is considered as a collection of abstract objects that shares a common characteristic.

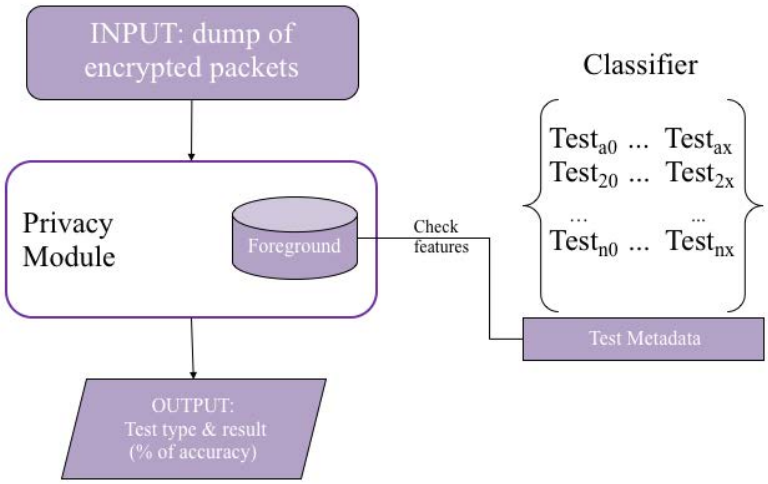

Figure 12: Privacy Module Working Flow

The module abstracts the execution of a specific test by generating a cumulative behavioural of the trace. From this result, features will be extracted and used by the module's classifier as a "foreground", that is the term used to identify the set of well-known tests which we will be able to reveal.

The module shall automate the recording of a testing session trace, by synchronizing the sniffer used to collect the encrypted traffic, with the control messages exchanges during its execution. Multiple instances of a single test session are required for training the classifier.

Besides the described privacy tool based on traffic analysis techniques, another privacy tool is envisaged, aiming to check if any personal data (such as IP address, MAC address, etc.) can be leaked from the device under test. The tool will leverage on the F-Interop packet Analyser used for the conformance and interoperability tests to look for data string that looks as personal data identifier.

Scope	Functionality
Test Configuration	• Encrypted trace (input) • Initialization with Foreground DB
Real Time commands	• Start/Stop recording traces • Execute evaluation
Statistics	• List of the tests executed • the result of the test (passed / failed)

Table 9: Framework modules - Privacy Functionalities

All the statistics generated by the module has to be considered as the probability that a certain test, and its result, has been performed.

4.4 Gap Analysis

This section provides a gap analysis for the implementation of the framework modules we identified. We performed a survey of existing tools we can integrate or leverage into the framework. We evaluated the candidates and selected the best fitting tools for each block, if one was available. When a candidate was available, we also performed an analysis to assess how the tools can be adapted or extended to fit the project objectives and methodology.

Category	Implementation Gap
Message Broker	Selected RabbitMQ. No implementation gap.
Timeline Controller	Implementation required.
QoS Test Tools: Synthetic Measurement	Selected SDN-RADAR, a Proof of Concept (PoC) of a monitoring tool, which collects active and passive measurements in a SDN network. Requires implementation of a set of APIs for third party uses Requires implementation of mitigation policies for operating network troubleshooting. Requires implementation of a suitable GUI for F-Interop users/contributors
QoS Test Tools: Network Impairment	Selected Linux NETEM for the low-level implementation. Provides all the required data plane functionality. Linux NETEM configuration is non-trivial. Requires implementation of named profile configuration. Requires implementation of statistics reporting according to named profile. Additional statistics reporting for amount of packets duplicated.
Scalability Test Tools: Passive Monitoring	Implementation required.
Scalability Test Tools: Active Emulation - CoAP	Selected Eclipse Californium (Eclipse Californium), an established open source implementation of CoAP. Provides a library for CoAP client and CoAP server. Requires implementation of dynamic endpoint pools and

	command sequencing.
Energy Consumption Test Tools (Implementation Specific)	Implementation required.
Privacy Test Tools	Implementation required. Traffic Analysis Tool provides a classifier that extracts features associated to an encrypted traffic dump. Requires adaptation of the module to the features of the traffic collected during F-Interop tests Requires training and tuning of the classifier, for distinguishing different classes of tests. Requires implementation of a repository to contain the classifier knowledge.

Table 10: Existing Tools and Gap Analysis

Note that the implementation gap **does not** include the integration of the tool into the framework by itself, i.e., the communication between the module and the F-Interop management layer is required for all the selected test suites.

5 Module Intercommunication Model

5.1 Module Intercommunication Requirements

As we described in section 1.2.3 "Deliverable Objectives and Methodology", the modules used for running performance tests are pluggable and interchangeable. For each test instance there will be a module in charge of co-ordination and mediation between the F-Interop central service and F-Interop user interface layers towards the test instance. The test mediator, the timeline controller, needs to communicate in a simple and extendible manner with the worker modules which provide the actual functionality, such as protocol emulation, network impairment, privacy measurement, and so on. The design approach specifically targets to support multiple instances of the same module and multiple modules.

There are several reasons for these requirements:

- A module performing protocol emulation may require several instances to reach its performance goal.
- Several modules may be needed due to the geographical location diversity. If a test is performed on two or more test beds, it is beneficial to co-locate performance modules nearer the resources to reduce latency.
- A module performing network impairment and passive analysis may require several instances due to the location diversity described above.
- Implementation-specific energy consumption modules may require one instance per targeted implementation type.

An intercommunication model based on a message broker is used to support the heterogeneous modules. Corresponding to the generic architecture of F-Interop AMQP (Advanced Message Queuing Protocol) is used for this purpose in WP3 as well.

This model allows coordinating distributed components over a single point of communication for each of the components. While the all components communicate with the message broker as far as the transfer protocol is concerned, the messages can be exchanged between any module in a one-to-one model (between two particular modules), one-to-many model (multicasting from one to many modules) and many-to-many module (broadcasting to all modules).

RabbitMQ (RabbitMQ), a well-established open-source (RabbitMQ Server Source Code Repository) AMQP server implementation is agreed to be used for F-Interop due to the following benefits:

- RabbitMQ provides a secure authentication scheme based on TLS certificates and alternatively Simple Authentication and Security Layer (SASL (Simple Authentication and Security Layer (SASL), 2006)), providing data protection layer on the test management plane.
- RabbitMQ supports clustering and high-availability, in case performance limitation is reached in the future.
- RabbitMQ provides APIs for an extensive selection of programming languages: Java, .NET, C, C++, Python, PHP, Ruby, Perl, Go, Haskell, Erlang

and others. This is especially beneficial to the open call process and integration of contributions into F-Interop.

Corresponding to the project decision, inter-communication of the modules in WP3 is also via the AMQP Eventbus (implemented with RabbitMQ). The following section elaborates more on the intercommunication design of WP3.

5.2 Module Intercommunication Design

5.2.1 High Level Intercommunication Description

The communication between the modules is facilitated through message exchange via RabbitMQ. Generally, the messages are sent for a certain **topic**, distinguished by a **routing key**, to an **exchange**. The RabbitMQ server then distributes the messages to individual **queues** that are bound to a specific **exchange** and **routing key.**

For the intercommunication between the modules we shall use a single exchange for a test instance, while the topic and the routing key shall be synonymous. The modules shall use anonymous queues bound to the topics and the single exchange.

All messages shall be formatted as JSON (The JavaScript Object Notation (JSON) Data Interchange Format, 2014) objects.

5.2.2 Message Routing Keys

The topic shall use distinctive routing keys and will determine the applicability of the message, as described in the following table:

Routing Key (Topic)	Description
Heartbeat	Used for heartbeat messages (see section 4.1.1)
Control	Used for test control messages, such as START, STOP and ABORT (see section 4.1.1)
Configuration	Used for initial module configuration. Messages to this exchange are valid only when sent before a test START is signalled (see *control* routing key).
Command	Used for real-time commands. Messages to this exchange are valid only when sent after a test START is signalled and before a STOP or ABORT is signalled (see *control* routing key)
Stats	Used to stream statistics. Messages to this exchange are valid only when sent after a test START is signalled and before a STOP or ABORT is signalled (see *control* routing key).

Table 11: Routing Key Association for Module Intercommunication

5.2.3 Common Message Fields

The messages contain common fields and module specific fields. The common fields are described in the following table:

Routing Key (Topic)	Field	Description
All routing keys	name	The module name. The name must be unique for a specific test instance. The name is provided to the module during instantiation.
heartbeat	timestamp	A string containing the time since the so-called "Unix epoch" being the number of seconds that elapsed since midnight, January 1st, 1970, UTC. The string shall have a decimal precision of at least 6 digits after the decimal point, meaning the timestamp is expressed with at least microsecond precision.
control	signal	A string specifying the event. The valid signals are "START", "STOP" and "ABORT" (see section 4.1.1).
configuration	dst	Destination module name for this specific message.
command	dst	Destination module name for this specific message.
command	method	The method name for this command. This string is module specific.
stats	timestamp	A string containing the time since the so-called "Unix epoch" being the number of seconds that elapsed since midnight, January 1st, 1970, UTC. The string shall have a decimal precision of at least 6 digits after the decimal point, meaning the timestamp is expressed with at least microsecond precision.

Table 12: Common Message Fields

5.2.4 Timeline Control

The timeline module controls multiple timelines starting at the same time when the START command is given over the *control* topic.

The timeline module shall calculate the desired real-time value according to the timeline configuration, and if a change is required, shall emit a message to the respective module over the *command* topic.

The timeline module is unaware of the underlying configuration of the module that is controlling. Instead, it is provided with a template that describes the method and parameters to be adjusted.

This design maximizes flexibility and extendibility and allows easy integration of contributed performance module as timeline controlled functional blocks in the test bed.

An example to the data structure used as input for the timeline control is described in *Annex 1: Timeline Control Structure Example*.

5.3 Spatial and Map Representation of online tests

In order to provide enrich services for the F-Interop users on the performance testing tools, they will be supported by F-Interop spatial and map Visual Representation tool.

The F-Interop Visual Representation tool is designed under the following principles:

- Flexibility: the different types of visualization views will be modulated and new service will be seamlessly added as needed.
- Scalability: it will support large scale of online scalability tests in real-time.
- Simplicity: the user interface will be designed simple to use.
- Privacy protection: all visualized data will be filtered by F-Interop privacy policy and the stored data for the Visual Representation tool will keep as secured and privacy protected.

The Visual Representation tool will support three types of visualization of the F-Interop online tests:

- Real time network visualization: it represents the instantaneous and recent data flow and monitors the activity in terms of data reception.
    - Outlook of the network visualization:

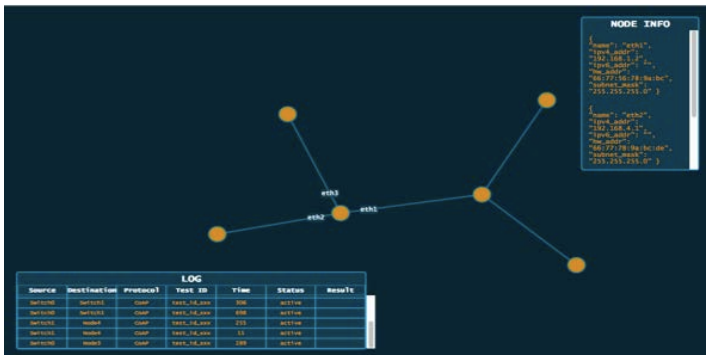

Figure 13: Network Visualization

- Real time world map visualization:
    - Represents 2D macro network visualization of the interactions and packet transfers among the various nodes/devices connected to the F-Interop platform.
    - May support modelling of 3D micro network visualization of detailed interactions and packet transfers among the test nodes/devices on specific sites as needed.
    - Outlook of the map visualization:

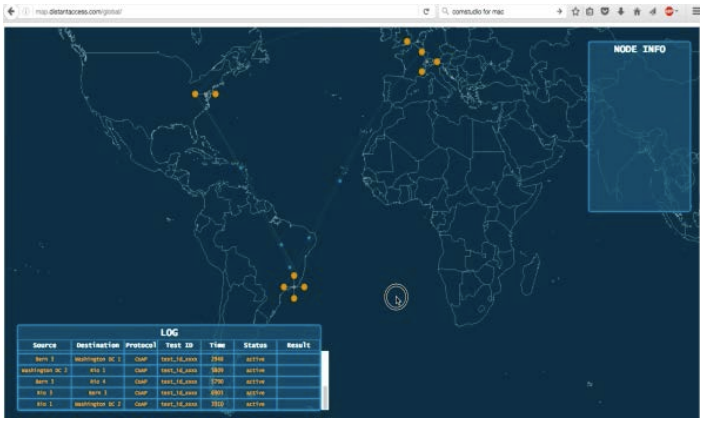

Figure 14: Map visualization

- Data set graph representation: it represents the values of collected data over a period of time. It will be particularly useful for scalability test that represents in a graph indicating the statistics of the collected data (e.g., latency in the function of the number of requests sent.)
    - Outlook of the data set graph visualization:

Figure 15: Data set graph visualization

Figure 16 illustrates the integrated view of the high level building blocks of the performance test with visual representation tool. As it is explained in the module intercommunication design in the Section 5.2, the modules in WP3 will intercommunicate with AMQP and the visual representation tool will also obtain the necessary information via F-Interop AMQP Eventbus. The testing tools transfer test configuration and statistics of the experiments to the Eventbus and visual representation tool collects the data on the Eventbus. The following Figure shows the high level building blocks of the F-Interop Visual Representation.

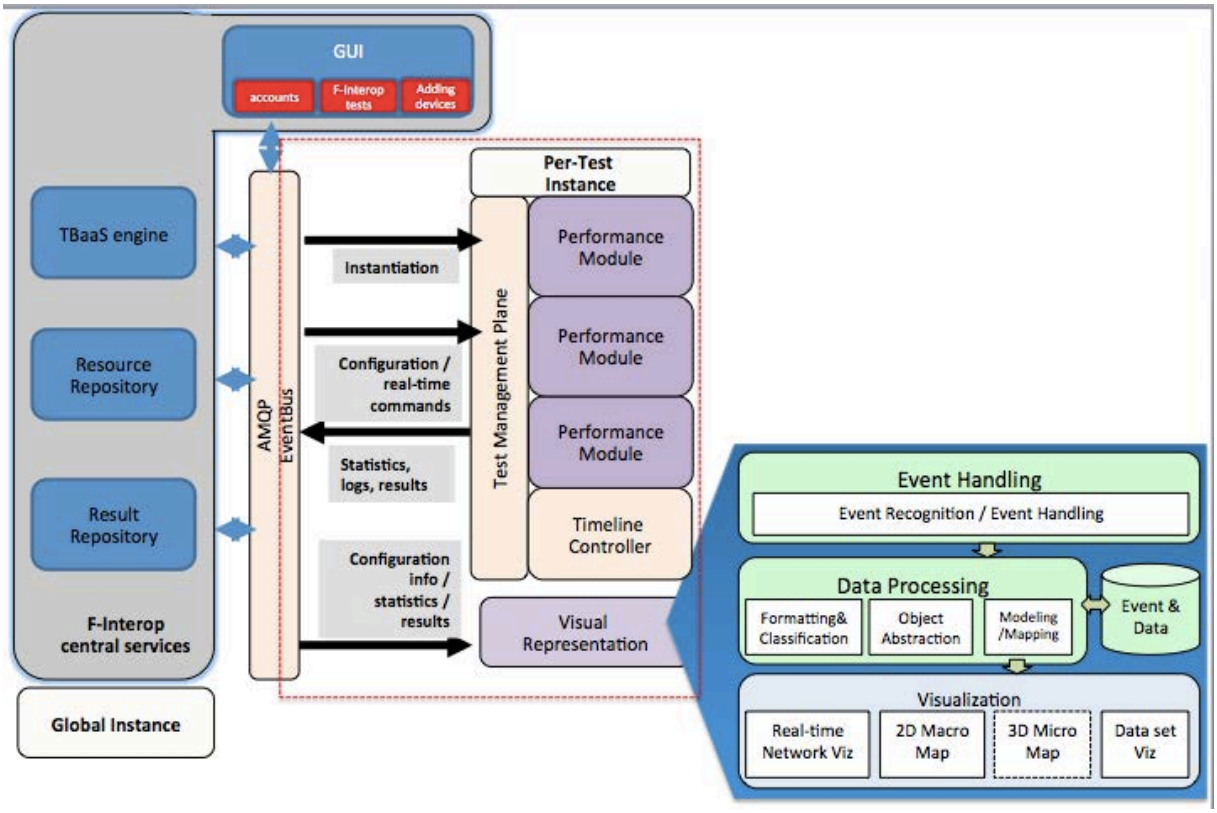

Figure 16: High-level building block of Visual representation tool

The “configuration” messages from the online test modules will be captured for modelling real-time network visualization and 2D macro map view. Note that precise location information will not be collected during the registration of the user information due to privacy protection policy of F-Interop. Thus the 2D map view will be served with country-level information unless users provide deeper information.

The obtained data from the configuration is changed to network data format of nodes and links. The nodes and links are networked and updated according to the traffic data captured during the online test. It will show as a network graph with animated real-time traffic flow visualization.

Location info from the user registration will be used for plotting the networks into 2D map view and will display the data exchanges among the current users.

3D micro map will need separate information from the users or it is possible to support when there is 3D location information of a specific site of F-Interop federated testbeds.

For a scalability test, “stats” message will be collected during the test and graphical views of the statistics will be displayed. The followings are examples of statistics information to be provided to the users:

- Incoming Throughput
- Outgoing Throughput
- Transaction rate
- Incoming notification rate for observed resources
- Response code

- Number of observed resources

According to the F-Interop user requirement in D1.1, the visualization tool will provide the F-interop users able to choose and control their most interesting data. It is noted that the visualization tool will be provided per user for privacy protection following the privacy requirements stated in D1.2.

When the user wishes to keep the test results, the desired information will be collected in F-Interop result repository and can be shown later in F-Interop Visual Representation tool. The publicly visualized information will be limited to the data that the corresponding user agrees to share.

The control signal “START” from the test instances will be captured and the F-Interop Visual Representation tool will provide the real-time network visualization. By triggering the control signal, the transaction messages from the online test instances are captured and used for the real-time network mapping that represents data flows during online performance tests or interoperability tests. This service will be continued per test until the control signal “STOP” or “ABORT” from the test instance is captured.

6 Conclusion

This document represents the foundation for the design and development of the performance and privacy test tools. Following the modular approach of the F-Interop Architecture, the common functionalities of the different tools will be implemented with modules that can be reused for performing several tests, and module intercommunication is performing via F-Interop AMQP EventBus.

Existing tools on which the F-Interop tools could be leveraged on have been identified. Besides this initial set of tools, others may be considered also in the future, if more suitable to accommodate the need, and requirements of the test tools.

This document will be updated by deliverables D3.2, D3.3, and D3.4 describing respectively the final performance test tool, privacy test tool, and the spatial and map representation, as implemented.

7 References

(1999). *Benchmarking Terminology for Firewall Performance.* IETF RFC 2647.

Contiki OS. (n.d.). From http://contiki-os.org/

(2016). *Data Center Benchmarking Terminology (Work in Progress).* IETF draft-ietf-bmwg-dcbench-terminology-05.

Eclipse Californium. (n.d.). From https://www.eclipse.org/californium/

Gheorghe, G., Avanesov, T., Palattella, M. R., Engel, T., & Popoviciu, C. (2015). *SDN-RADAR: Network troubleshooting combining user experience and SDN capabilities*. From http://orbilu.uni.lu/handle/10993/23892

IxNetwork VE. (n.d.). (Ixia) From https://www.ixiacom.com/products/ixnetwork-ve

Linux NETEM. (n.d.). From http://www.linuxfoundation.org/collaborate/workgroups/networking/netem

Mininet. (n.d.). From http://mininet.org/

(2015). *Network Functions Virtualisation (NFV); Infrastructure Overview.* ETSI GS NFV-INF 001 V1.1.1.

(2015). *Observing Resources in the Constrained Application Protocol (CoAP).* IETF RFC 7641.

OpenWSN Wiki. (n.d.). From https://openwsn.atlassian.net/wiki/

Panchenko, A., Lanze, F., Zinnen, A., Henze, M., Pennekamp, J., Wehrle, K., et al. (n.d.). *Website Fingerprinting at Internet Scale*. From http://hdl.handle.net/10993/24117

RabbitMQ. (n.d.). From http://www.rabbitmq.com/

RabbitMQ Server Source Code Repository. (n.d.). From https://github.com/rabbitmq/rabbitmq-server

Service Name and Transport Protocol Port Number Registry. (n.d.). (IANA) Retrieved July 19, 2016 from http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml

(2006). *Simple Authentication and Security Layer (SASL).* IETF RFC 4422.

Spirent TestCenter Virtual. (n.d.). (Spirent) From http://www.spirent.com/Products/TestCenter/DataCenter/Virtual

(2014). *The Constrained Application Protocol (CoAP).* IETF RFC 7252.

(2014). *The JavaScript Object Notation (JSON) Data Interchange Format.* IETF RFC 7159.

8 Annex 1: Timeline Control Structure Example

In this annex, we describe an example timeline control structure according to the current prototypes. The following graphic shows the values emitted by the timeline according to the test time.

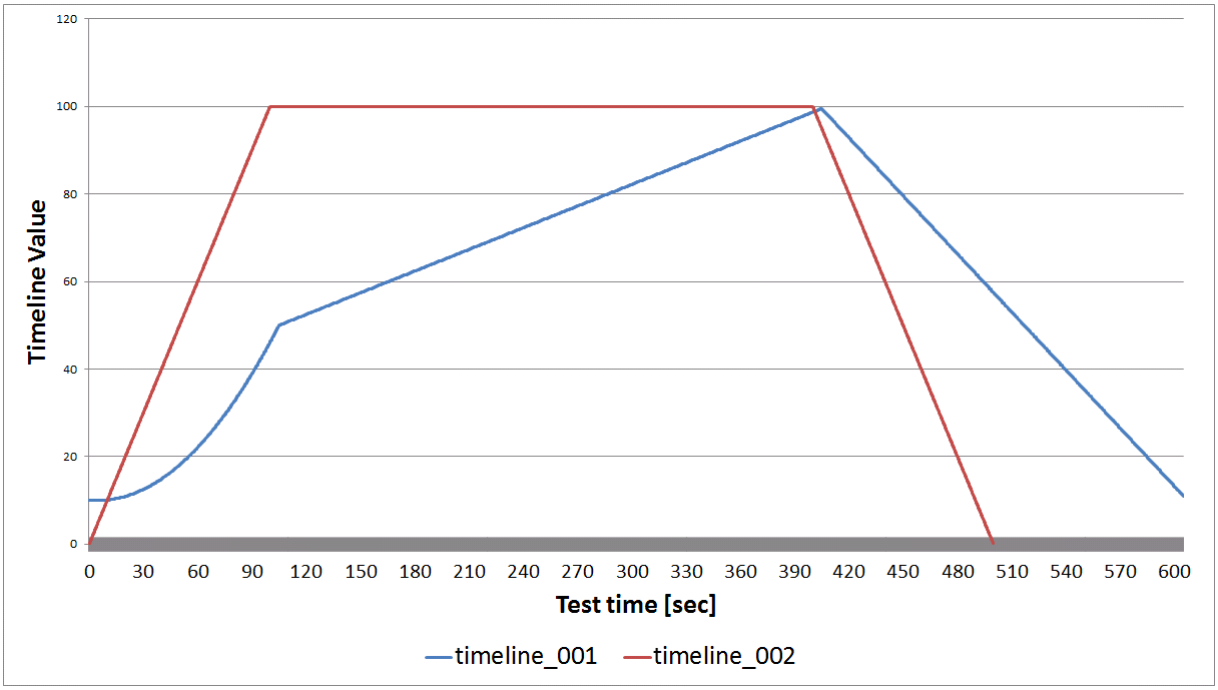

Figure 17: Timeline Example

The timeline presented below presents the matching information structure. The different fields are clarified further below the structure.

```
'timelines':
   [
      {
        'name':'timeline_001',
        'message':{
          'dst':'finterop2.netem',
          'method':'MODIFY_PROFILE',
          'profile':'coap1',
        },
        'value_path':['netem_parameters'],
        'value_format':'delay %.3fms',
        'value_type':'FLOAT',
        'start_value':10.0,
        'segments':[
          {
            'duration':5.0,
            'end_value':10.0,
            'interpolation':'LINEAR',
          },
          {
            'duration':100.0,
```

```
        'end_value':50.0,
        'interpolation':'POWER2',
      },
      {
        'duration':300.0,
        'end_value':99.5,
        'interpolation':'LINEAR',
      },
      {
        'duration':200.0,
        'end_value':11.0,
        'interpolation':'LINEAR',
      }
    ]
  },
          {
    'name':'timeline_002',
    'message':{
      'dst':'finterop4.coap',
      'method':'MODIFY_PROFILE',
      'profile':'coap6',
    },
    'value_path':['emulated_clients'],
    'value_format':'%d',
    'value_type':'INT',
    'start_value': 0,
    'segments':[
      {
        'duration':100,
        'end_value':100,
        'interpolation':'LINEAR',
      },
      {
        'duration':300,
        'end_value':100,
        'interpolation':'LINEAR',
      },
      {
        'duration':100,
        'end_value':0,
        'interpolation':'LINEAR',
      }
    ]
  }
]
```

There are two timeline below, *timeline_001* and *timeline_002* (user defined names). Each timeline controls a single parameter for a single module instance.

Timeline_001 controls the module named "finterop2.netem" and changes the value of the delay to profile *coap1* on interface *eth2*.

The timeline has 4 segments:

- Segment #1 starts using the timeline's *start_value* parameter (10.0) and ends with an *end_value* (10.0) with linear interpolation over 5 seconds. In this case, only one message is sent to set the start value, after which no change is done, since the start value and the end value are equal.
- Segment #2 starts using the end value of the previous segment (10.0) and ends with the *end_value* of segment #2 (50.0), using a cubic interpolation over 100 seconds. The value for each point in time *t* is evaluated as follows:

  $$Start\ Value + (End\ Value - Start\ Value)(\frac{tT}{})^2$$

  Where *T* is the duration of the segment.

  Thus, the value at 75 seconds after segment #2 has begun shall be:

  $$10 + (50 - 10)\left(\frac{75}{100}\right)^2 = 32.5$$

- Segment #3 starts with the end value of the previous segment (50.0) and ends with the *end_value* (99.5) with linear interpolation over 300 seconds. The value for each point in time *t* is evaluated as follows:

  $$Start\ Value + (End\ Value - Start\ Value)(\frac{t}{T})$$

  Where *T* is the duration of the segment.

  Thus, the value at 240 seconds after segment #3 has begun shall be:

  $$50 + (99.5 - 50)(\frac{240}{300}) = 89.6$$

- Segment #4 starts with the end value of the previous segment (99.5) and ends with the *end_value* (11.0) with linear interpolation over 200 seconds.

Timeline_002 controls the module "finterop4.coap" and sets the number of emulated clients to a profile named "coap6". It has 3 segments:

- Segment #1 starts using the timeline's *start_value* parameter (0.0) and ends with an *end_value* (100.0) with linear interpolation over 100 seconds.
- Segment #2 starts with the end value of the previous segment (100.0) and ends with the *end_value* (100.5) with linear interpolation over 300 seconds. The value is therefore not changed for the duration of this segment.
- Segment #3 starts with the end value of the previous segment (100.0) and ends with the *end_value* (0.0) with linear interpolation over 100 seconds.