HORIZON 2020
Information and Communication Technologies
Integrating experiments and facilities in FIRE+

Deliverable D4.2
Testbed As A Service integration and SDK 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: 30/04/2018
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.
Abstract

This deliverable describes the work done in task 4.2 “Testbed as a service integration and SDK”. We describe the software interfaces exposed by the Testbed As A Service (TBaas), which are used by the F-Interop Session Orchestrator and GUI to request, provision and configure testbed resources.

We describe the contents and usage of the software development kit (SDK) in detail, and provide a manual on how it can be deployed.
# Table of Contents

Table of Contents

List of Figures

List of Acronyms

1 Introduction
   1.1 About F-Interop
   1.2 Deliverable Objectives
      1.2.1 Work package Objectives
      1.2.2 Task Objectives
      1.2.3 Deliverable Objectives and Methodology

2 Design of the 'Testbed as a Service'
   2.1 Introduction
   2.2 Architectural view of F-Interop TBaaS

3 Implementation of the 'Testbed as a Service'
   3.1 Overview
   3.2 The High-Level API
      3.2.1 General overview
      3.2.2 API Resources
      3.2.3 SpeaksFor
   3.3 The Low-Level API
      3.3.1 Authentication
      3.3.2 Create
      3.3.3 Status
      3.3.4 Renew
      3.3.5 Delete
      3.3.6 Graphical testbed user interface

4 Software development kit (SDK)
   4.1 Overview of the SDK
   4.2 Contents of the software development kit
      4.2.1 AMQP-server
      4.2.2 Session Orchestrator
      4.2.3 GUI
      4.2.4 Supporting scripts

5 Deploying the software development kit
   5.1 Registering an Fed4FIRE account
   5.2 Install jFed
   5.3 Start the SDK in jFed
   5.4 Access the SDK resources

6 Developing a third-party testing tool using the SDK
   6.1 index.json
   6.2 supervisord configuration file
   6.3 Docker images
   6.4 Testing tool interface
   6.5 Sending and receiving network traffic
   6.6 Executing a testcase
   6.7 Analyzing the network messages
List of Figures

Figure 1: High level architecture with testbeds ................................................................................. 9
Figure 2: High level architecture with multiple tests on testbeds running in parallel ....................... 10
Figure 3: Speaks-for mechanism ....................................................................................................... 17
Figure 4: jFed GUI for easy interaction with testbed resources ......................................................... 21
Figure 5: Registration form on the imec Authority website ............................................................... 25
Figure 6: Login screen of jFed ........................................................................................................... 26
Figure 7: Login screen for Fed4FIRE-credentials in jFed ................................................................. 26
Figure 8: Local login screen in jFed ................................................................................................... 27
Figure 9: Opening an ESpec in jFed .................................................................................................... 27
Figure 10: Configuring an experiment for the F-interop SDK in jFed ............................................. 28
Figure 11: An SDK being deployed by jFed ....................................................................................... 29
Figure 12: An SDK with a Zolertia Re-mote on the imec w-iLab.t testbed ...................................... 36
**List of Acronyms**

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMQP</td>
<td>Advanced Message Queuing Protocol</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>CA</td>
<td>Consortium Agreement</td>
</tr>
<tr>
<td>CoAP</td>
<td>Constrained Application Protocol</td>
</tr>
<tr>
<td>Comsoc</td>
<td>Communications Society</td>
</tr>
<tr>
<td>DESCa</td>
<td>Development of a Simplified Consortium Agreement</td>
</tr>
<tr>
<td>DHCP</td>
<td>Dynamic Host Configuration Protocol</td>
</tr>
<tr>
<td>DHT</td>
<td>Distributed Hash Tables</td>
</tr>
<tr>
<td>DNS</td>
<td>Domain Name System</td>
</tr>
<tr>
<td>DNSsec</td>
<td>Domain Name System Security Extensions</td>
</tr>
<tr>
<td>DPA</td>
<td>Data Protection Authorities</td>
</tr>
<tr>
<td>DPO</td>
<td>Data Protection Officer</td>
</tr>
<tr>
<td>DUT</td>
<td>Device Under Test</td>
</tr>
<tr>
<td>EC</td>
<td>European Commission</td>
</tr>
<tr>
<td>ENISA</td>
<td>European Union Agency for Network and Information Security</td>
</tr>
<tr>
<td>ETSI</td>
<td>European Telecommunications Standards Institute</td>
</tr>
<tr>
<td>EU</td>
<td>European Union</td>
</tr>
<tr>
<td>FP7</td>
<td>Seventh Framework Programme</td>
</tr>
<tr>
<td>GA</td>
<td>Grand Agreement</td>
</tr>
<tr>
<td>GA</td>
<td>General Assembly</td>
</tr>
<tr>
<td>GPS</td>
<td>Global Positioning System</td>
</tr>
<tr>
<td>HTTPS</td>
<td>Hypertext Transfer Protocol Secure</td>
</tr>
<tr>
<td>ICT</td>
<td>Information and Communication Technologies</td>
</tr>
<tr>
<td>ID</td>
<td>Identifier</td>
</tr>
<tr>
<td>IEC</td>
<td>International Electrotechnical Commission</td>
</tr>
<tr>
<td>IEEEn</td>
<td>Institute of Electrical and Electronics Engineers</td>
</tr>
<tr>
<td>IERC</td>
<td>European Research Cluster on the Internet of Things</td>
</tr>
<tr>
<td>IETF</td>
<td>Internet Engineering Task Force</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>IPC</td>
<td>Intellectual Property Committee</td>
</tr>
<tr>
<td>IPM</td>
<td>IPR Monitoring and Exploitation Manager</td>
</tr>
<tr>
<td>IPR</td>
<td>Intellectual Property Rights</td>
</tr>
<tr>
<td>IPSEC</td>
<td>Internet Protocol Security</td>
</tr>
<tr>
<td>IPv4</td>
<td>Internet Protocol version 4</td>
</tr>
<tr>
<td>IPv6</td>
<td>Internet Protocol version 6</td>
</tr>
<tr>
<td>ISO</td>
<td>International Standards Organization</td>
</tr>
<tr>
<td>ISP</td>
<td>Internet Service Provider</td>
</tr>
<tr>
<td>IT</td>
<td>Information Technology</td>
</tr>
<tr>
<td>ITU</td>
<td>International Telecommunication Union</td>
</tr>
<tr>
<td>IUT</td>
<td>Implementation Under Test</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>KPI</td>
<td>Key Performance Indicator</td>
</tr>
<tr>
<td>LSPI</td>
<td>Legal, Security and Privacy Issues</td>
</tr>
<tr>
<td>MAC</td>
<td>Media Access Control</td>
</tr>
<tr>
<td>MSc</td>
<td>Master of Science</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>M2M</td>
<td>Machine to Machine</td>
</tr>
<tr>
<td>OASIS</td>
<td>Organization for the Advancement of Structured Information Standards</td>
</tr>
<tr>
<td>OECD</td>
<td>Organization for Economic Cooperation and Development</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>OSN</td>
<td>Online Social Network</td>
</tr>
<tr>
<td>PC</td>
<td>Project Coordinator</td>
</tr>
<tr>
<td>PCP</td>
<td>Partner Contact Person</td>
</tr>
<tr>
<td>PDPO</td>
<td>Personal Data Protection Officer</td>
</tr>
<tr>
<td>PERT</td>
<td>Program Evaluation Review Technique</td>
</tr>
<tr>
<td>PhD</td>
<td>Doctor of Philosophy</td>
</tr>
<tr>
<td>PM</td>
<td>Person Month</td>
</tr>
<tr>
<td>PMB</td>
<td>Project Management Board</td>
</tr>
<tr>
<td>PPR</td>
<td>Periodic Progress Report</td>
</tr>
<tr>
<td>PRAAT</td>
<td>Privacy Risk Area Assessment Tool</td>
</tr>
<tr>
<td>P&amp;T</td>
<td>Post &amp; Telecom</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>RAND</td>
<td>Reasonable and Non Discriminatory</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>RFC</td>
<td>Request For Comments</td>
</tr>
<tr>
<td>RSpec</td>
<td>Resource Specification</td>
</tr>
<tr>
<td>R&amp;D</td>
<td>Research &amp; Development</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>SME</td>
<td>Small Medium Enterprise</td>
</tr>
<tr>
<td>SO</td>
<td>Session Orchestrator</td>
</tr>
<tr>
<td>SOTA (or SoA)</td>
<td>State Of the Art</td>
</tr>
<tr>
<td>SSL</td>
<td>Secure Sockets Layer</td>
</tr>
<tr>
<td>TBaaS</td>
<td>TestBed as a Service</td>
</tr>
<tr>
<td>TC</td>
<td>Technical Coordinator</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>TL</td>
<td>Task Leader</td>
</tr>
<tr>
<td>TLS</td>
<td>Transport Layer Security</td>
</tr>
<tr>
<td>Tor</td>
<td>The Onion Router</td>
</tr>
<tr>
<td>TRL</td>
<td>Technology Readiness Level</td>
</tr>
<tr>
<td>TT</td>
<td>Testing Tool</td>
</tr>
<tr>
<td>UN</td>
<td>United Nations</td>
</tr>
<tr>
<td>UNCTAD</td>
<td>United Nations Conference on Trade and Development</td>
</tr>
<tr>
<td>UPRAAT</td>
<td>Universal Privacy Risk Area Assessment Tool</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>US</td>
<td>United States</td>
</tr>
<tr>
<td>VoIP</td>
<td>Voice over Internet Protocol</td>
</tr>
<tr>
<td>WES</td>
<td>Women’s Engineering Society</td>
</tr>
<tr>
<td>WITEC</td>
<td>Women in science, Engineering and Technology</td>
</tr>
<tr>
<td>WoT</td>
<td>Web of Trust</td>
</tr>
<tr>
<td>WP</td>
<td>Work Package</td>
</tr>
<tr>
<td>WPL</td>
<td>Work Package Leader</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
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 Deliverable Objectives

1.2.1 Work package Objectives

WP4 has the following goals:

- Research, develop and integrate the F-Interop Testbed as a Service
- Develop and demonstrate the open API and resources repository
- Develop the required SDK for third parties developments and extensions
- Develop and optimize the user interface

1.2.2 Task Objectives

Task 4.2 in WP4 describes two separate components that need to be developed:

1. It is responsible for integrating the various testbeds resources and enablers together.
2. It is responsible for developing an SDK for third parties extensions and developments.

For this, a milestone MS18 was reached in month 18 of this project, in which both components needed to be demonstrated. In this deliverable D4.2 we describe the inner workings of these components.

1.2.3 Deliverable Objectives and Methodology

This deliverable describes the software interfaces exposed by the Testbed As A Service, which are used by the F-Interop user interface to request, provision and configure testbed resources.

We describe the contents of the software development kit (SDK), and how it is deployed. We also describe how the software development kit (SDK) can be used by (open-call) contributors for the development of third party extensions.
2 Design of the ‘Testbed as a Service’

2.1 Introduction

This section will describe the “Testbed as a Service”-component of F-Interop in detail. This component was first defined in D1.3, section 3.2 ‘High level architecture with testbeds’ as ‘An extra component responsible for deploying the F-Interop framework on virtual machines in a testbed and for reserving and allocating devices needed for the test. The contributor has full access to the virtual machines and software in the middle circle and as such can work on the orchestration, analysis, AMQP messaging and result storage of a test’.

2.2 Architectural view of F-Interop TBaaS

The following figures have been adopted from F-Interop D1.3 to discuss the functions of the F-Interop TBaaS.

As can be seen on Figure 1, the TBaaS communicates with three other F-Interop components. The GUI and Session Orchestrator use the TBaaS to request, provision and manage resources on testbeds. For this they use a REST interface which allows them to perform these actions.

On the other hand, the TBaaS engine also feeds the Resource Repository with all devices available on the testbeds. It does this by querying the testbeds through the AM API with the Listresources call (see https://fed4fire-testbeds.ilabt.iminds.be/asciidoc/federation-am-api.html#ListResources) and converting them to the right JSON format and put it in AMQP messages towards the Resource Repository.

![Figure 1: High level architecture with testbeds](image-url)
Figure 2 shows how it looks like when multiple users in parallel deploy the F-Interop framework on testbeds. Each user has its own instance, with its own AMQP message bus and completely securely separated from the other instances. On the other hand, the user has full access and as such full flexibility on his instance.

Figure 2: High level architecture with multiple tests on testbeds running in parallel
3 Implementation of the ‘Testbed as a Service’

3.1 Overview

The TBaaS has two REST APIs on different levels:

- The low-level API is F-Interop independent. A testbed specific RSpec (Resource Specification) has to be specified (which contains the resources that are being requested, and how they must be linked with each other). It also supports ESpec (Experiment Specification), which allows for a richer definition of how the resources have to be configured: files that need to be uploaded, software which must be installed, scripts which must be executed, etc.

- The high-level API is specified specifically towards F-Interop and contains as such only the relevant items, e.g. instead of an XML RSpec, resources can just be specified by their unique ID.

The high-level API interprets the F-Interop specific request and converts it into an ESpec which is subsequently passed on to the low-level API. This hides the complexity of the low-level API from the other F-Interop services that need to use the TBaaS.

Both API’s use REST for structuring the API-calls, and use JSON-objects as input and output. As these are both widely adopted industry standards, this eases the integration into the other F-Interop components.

3.2 The High-Level API

This section describes the TestBed as a Service API version v2.

3.2.1 General overview

3.2.1.1 Authentication

All API access is over HTTPS. Users authenticate with the TBaaS API by using their client certificate to setup the SSL-connection. You get a client certificate (a PEM file containing an X.509 private key and a certificate = public key signed by the authority) when you register for an account on the F-Interop authority.

```
curl -X GET --cert-type pem --cert <YOUR AUTHORITY PEM> \ 
https://tbaas.ilabt.iminds.be/api/
```

When the provided credentials are incorrect, the SSL-connection setup will fail.

3.2.1.2 Schema

All data is sent and received as JSON-LD (JavaScript Object Notation for Linked Data), unless stated otherwise.

All timestamps are sent and received in the RFC3339 format¹:

```
1985-04-12T23:20:50.52Z
```

¹ https://tools.ietf.org/html/rfc3339
3.2.1.3 Client errors

When an operation is attempted for which the user has insufficient rights, the API will return a 403 Forbidden response.

3.2.2 API Resources

3.2.2.1 InteropTestbed

An InteropTestbed contains a collection of resources which are present on one or more of the federated testbeds. It can also contain a stand-alone F-Interop Message Bus (ie. AMQP-server) and testing tools.

An example:

```
{
  "@type": "InteropTestbed",
  "id": 1548,
  "@id": "https://tbaas.ilabt.iminds.be/api/interopTestbed/1548",
  "experiment": {
    "@type": "Experiment",
    "id": 958,
    "project": "finterop",
    "sliceName": "slice1",
    "sliceUrn": "urn:publicid:IDN+f-interop.eu:finterop+slice+slice1",
    "created": "2016-10-01T16:00:00Z",
    "expire": "2016-11-30T16:00:00Z",
    "owner": "urn:publicid:IDN+f-interop.eu+user+user1",
    "sharedWith": [
      "urn:publicid:IDN+f-interop.eu+user+user2"
    ],
    "am": "http://fls.fed4fire.eu/api2/server/320",
    "status": "READY"
  },
  "environment": {
    "@type": "InteropEnvironment",
    "version": "v2.0",
    "amqpUrl": "amqp://user1:password1@amqp.example.com/session1",
    "amqpServerPort": 1234,
    "amqpAuthentication": {
      "@type": "AMQPAuthentication",
      "rootCertificate": "-----BEGIN CERTIFICATE-----\n...
------END CERTIFICATE-----\n",
      "clientCertificate": "-----BEGIN CERTIFICATE-----\n...
------END CERTIFICATE-----\n",
      "clientPrivateKey": "-----BEGIN PRIVATE KEY-----\n...

    }
  },
  "resources": [
    {
      "id": "urn:publicid:IDN+wall2.ilabt.iminds.be+node+xxxxx",
      "image": "urn:publicid:IDN+wall2.ilabt.iminds.be+image+coap2test"
    },
    {
      "id": "urn:publicid:IDN+wall2.ilabt.iminds.be+sensor+n095b:1",
      "image": "https://www.example.com/chaser.hex"
    }
  ],
  "status": "STARTING"
}
```
All requests and responses to this endpoint will contain an InteropTestbed-object. This object has the following fields:

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Integer</td>
<td>Unique ID of the InteropTestbed instance. Generated by the TBaaS.</td>
</tr>
<tr>
<td>experiment</td>
<td>Experiment</td>
<td>The underlying slice in which the testbed resources are contained.</td>
</tr>
<tr>
<td>environment</td>
<td>InteropEnvironment</td>
<td>This contains all the information about the F-Interop environment in which the resources must operate.</td>
</tr>
<tr>
<td>resources</td>
<td>InteropTestbedResource[]</td>
<td>List of resources which are requested for this InteropTestbed-instance.</td>
</tr>
<tr>
<td>status</td>
<td>String</td>
<td>The current status of the InteropTestbed instance. The possible values are STARTING, READY, FAILED or NONEXISTING</td>
</tr>
</tbody>
</table>

### 3.2.2.1.1 Experiment

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Long</td>
<td>Unique ID within the TBaaS for this slice</td>
</tr>
<tr>
<td>Project</td>
<td>String</td>
<td>Project in which the slice was created</td>
</tr>
<tr>
<td>sliceName</td>
<td>String</td>
<td>Name of the slice</td>
</tr>
<tr>
<td>sliceUrn</td>
<td>String</td>
<td>URN of the slice</td>
</tr>
<tr>
<td>created</td>
<td>Timestamp</td>
<td>Time when the slice was created</td>
</tr>
<tr>
<td>expire</td>
<td>Timestamp</td>
<td>Time when the slice will expire</td>
</tr>
<tr>
<td>owner</td>
<td>String</td>
<td>URN of the user who owns the slice</td>
</tr>
<tr>
<td>sharedWith</td>
<td>String[]</td>
<td>List of URN’s of users with who the slice is shared</td>
</tr>
<tr>
<td>am</td>
<td>String</td>
<td>URL referring to the Aggregate Manager on which the slice is located</td>
</tr>
<tr>
<td>status</td>
<td>String</td>
<td>Internal status of the slice. Possible values are STARTING, READY, FAILED and NONEXISTING. The status of the slice doesn't necessarily match that of the InteropTestbed, as the latter also takes the installation and configuration of software and firmware on the resources into account.</td>
</tr>
</tbody>
</table>

### 3.2.2.1.2 InteropEnvironment

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>String</td>
<td>Unique name which allows the TBaaS to identify the version of all F-Interop tools.</td>
</tr>
<tr>
<td>amqpUrl</td>
<td>String</td>
<td>The URL of the exchange on which the F-Interop session is hosted. This URL can contain authentication details such as an username and/or password</td>
</tr>
<tr>
<td>amqpAuthentication</td>
<td>AMQPAuthentication</td>
<td>Authentication information that can be used by the F-Interop agents to connect to the AMQP F-Interop Bus (optional)</td>
</tr>
</tbody>
</table>
### 3.2.2.3 AMQPAuthentication

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rootCertificate</td>
<td>String</td>
<td>PEM-encoded X.509 (self-signed) root certificate. This certificate will be used to create certificates for all parties in the F-Interop Testbed</td>
</tr>
<tr>
<td>clientCertificate</td>
<td>String</td>
<td>PEM-encoded X.509 client certificate, signed by the root authority. This certificate can be used to setup the connection to the AMQP F-Interop Bus</td>
</tr>
<tr>
<td>clientPrivateKey</td>
<td>String</td>
<td>PEM-encoded private key that must be used together with the client certificate to setup the connection to the AMQP F-Interop Bus</td>
</tr>
</tbody>
</table>

### 3.2.2.4 InteropTestbedResource

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>String</td>
<td>Unique ID of the resource</td>
</tr>
<tr>
<td>image</td>
<td>String</td>
<td>The software that must be deployed on the resource. In case of a VM, this must reference to a disk image. In case of an IoT device, this must reference to a hex-file that can be flashed on the IoT device.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The image may be referenced by an URN or an URL. A filename is also accepted when the file is uploaded together with the InteropTestbedResource as part of a Multipart request.</td>
</tr>
</tbody>
</table>

### 3.2.2.2 Creating an InteropTestbed

A new InterTestbed can be requested by posting a InteropTestbed-object:

```
POST /interop-testbed
```

The minimal InteropTestbed-object that needs to be posted is an empty one, in which case all the default options will apply. The default options are as follows:

**Experiment:** a new slice, owned by the user performing the request, with a duration of 2 hours will be created. This slice will have a random name, and will be created in a project that is available to the user.

**Environment:** a stand-alone F-Interop environment of the latest stable version will be setup. For authentication in this environment, a new self-signed root certificate will be created. This root certificate will then be used to issue server and client certificates.

**Resources:** no IoT devices or VMs will be allocated.

```
{  
    "@type": "InteropTestbed"  
}
```

The following fields can be used to customize the request:
- InteropTestbed:
  - experiment
  - environment
  - resources
- Experiment
  - project
  - sliceName
  - expire
  - sharedWith
- Environment
  - version
  - amqpServerHostname
  - amqpServerPort
  - amqpAuthentication
- AMQPAuthentication
  - rootCertificate
  - clientCertificate
  - clientPrivateKey
- InteropTestbedResource
  - id
  - image

When specifying the environment, the version must always be provided. If you want to setup a stand-alone F-Interop test environment, no amqp-fields should be defined. However, if you want to use an existing F-Interop test environment, all amqp-fields must be defined. The AMQPAuthentication-object in the amqpAuthentication-field must also be fully defined.

All other fields can be customized without restriction.

### 3.2.2.2.1 Response if InteropTestbed was successfully created

Status: 200 OK

```
{
  "@type": "InteropTestbed",
  "id": 15,
  <FULL INTEROP TESTBED OBJECT>...
}
```

### 3.2.2.2 Response if request contains an error

Status: 400 Bad Request

```
{
  "code": 400,
  "message": "Unknown environment version"
}
```

### 3.2.2.3 Response if an error occurred during the InteropTestbed creation

Status: 500 Internal Server Error

```
{
  "code": 500,
  "error": "Error occurred during InteropTestbed creation"
}
```
"message": "The AM could not be reached to create a new slice"
}

3.2.2.3 Fetching current status of InteropTestbed

The current status of an interop testbed can be requested by performing a GET-request:

```
GET /interop-testbed/:id
```

This request will return a full InteropTestbed-object.

3.2.2.4 Updating an InteropTestbed

An InteropTestbed can be updated by performing a PUT-operation:

```
PUT /interop-testbed/:id
```

Only the fields ‘expire’ and ‘sharedWith’ of Experiment may be modified. In all other cases a 
400 Bad Request will be returned.

3.2.2.5 Destroying an InteropTestbed

An InteropTestbed can be destroyed by performing a DELETE-operation:

```
DELETE /interop-testbed/:id
```

This operation is permanent, and will release all allocated resources. When a stand-alone F-Interop Bus 
and testing tools were requested, then these will be shut down and destroyed immediately.

3.2.3 SpeaksFor

In order for the TBaaS to be able to allocate and provision resources on the federated testbed for a user, 
the TBaaS must have a valid SpeaksFor-certificate of that user. This certificate authorizes the TBaaS to 
speak on the behalf of the user to the testbeds.

Figure 3 shows how the speaks-for authentication and authorisation mechanism works when users 
allocate testbed resources through a 3rd party service (such as the F-Interop platform).

In step 1, the user needs to have a public signed cert/private key pair from the F-Interop authority (for usability, 
this can be created at the F-Interop web-based GUI and hidden away from the end-user). In step 2, The user creates then a speaks-for certificate which says that the user allows F-Interop (the TBaaS component in particular) to talk to non-F-Interop testbeds in the name of the user. (this speaks-for certificate is a signed XML).

In step 3, the TBaaS uses then its own certificate/keypair to talk to a testbed, but it also transmits the 
speaks-for certificate as an argument. In the end (step 4), the testbed has thus knowledge of the real 
end user (from 1) and knows that it has been set up by the TBaaS engine of F-Interop.

This means, that the testbeds can do the following:

- In case of abuse of resources, the testbed can immediately contact the end-user (and F-Interop as intermediate)
- F-Interop can state that the end-user is responsible for what he does on the testbeds
- If the testbeds have quota per user, it can be finetuned, and not F-Interop as a whole needs to 
have a specific quota for usage e.g.
3.2.3.1 Registering a SpeaksFor credential

A SpeaksFor credential can be registered by performing a POST-operation, with the body containing the SpeaksFor-credential.

```
POST /speaksfor
```

The subject of this SpeaksFor credential must be the subject of the client certificate that is used to authenticate the SSL-session to the TBaaS API. When these users don’t match, a 403 Forbidden reply will be returned.

This credential is XML-based, so all requests and responses containing the credential will have Content-Type: application/xml

### 3.3 The Low-Level API

This low-level API is already functional and available at [http://jfedapi.ilabt.iminds.be/#/demo](http://jfedapi.ilabt.iminds.be/#/demo) (for a web-based functional demo), and at [https://jfedapi.ilabt.iminds.be/api/experiment/](https://jfedapi.ilabt.iminds.be/api/experiment/) if you want to use the API natively.

It is REST based and consists out of four functionalities:

- CREATE a new experiment
- STATUS of an experiment
- RENEW an experiment to extend the expiration time
- DELETE an experiment
3.3.1 Authentication

This is similar as with the high-level API. All API access is over HTTPS. Users authenticate with the API by using their client certificate to setup the SSL-connection. You get a client certificate (a PEM file containing an X.509 private key and a certificate = public key signed by the authority) when you register for an account on the F-Interop authority.

```bash
curl -X POST -H "Content-Type: application/json" --data @create-experiment-data.json \
--cert-type pem --cert <YOUR AUTHORITY PEM> \
https://jfedapi.ilabt.iminds.be/api/experiment
```

When the provided credentials are incorrect, the SSL-connection setup will fail.

For using the fully functional web-based demo, you can load your client certificate as follows in your favourite browser:

- Login to the iLab.t authority (https://authority.ilabt.iminds.be/) and click “Download PKCS12 version of your certificate”.
- Now import the certificate into your OS/Browser:
  - For Windows users using Chrome or Internet Explorer: double click the p12 file and install it on your OS.
  - For MAC users using Safari: double click the p12 file and install it on your OS.
  - For Firefox users (Windows, Linux, MAC): in Firefox, go to Options > Advanced > Certificates and click on View Certificates. Now click Import and browse to the location of your p12 certificate and import it.

Also for this API, Speaks-for is in use (see Annex 3). The web-based demo shows how a speaks-for certificate can be created.

3.3.2 Create

For the Create call, the following call is needed:

```bash
curl -X POST -H "Content-Type: application/json" --data @create-experiment-data.json \
--cert-type pem --cert <YOUR AUTHORITY PEM> \
https://jfedapi.ilabt.iminds.be/api/experiment
```

with the following create-experiment-data.json example:

```json
{
  "requestRspec": "<rspec xmlns="http://www....",
  "speaksForCredential": "<xml version="1.0" encoding="UTF-8"... "
  "project": "f-interop",
  "sliceName": "demol",
  "expire": "2016-10-22T15:06:34.000Z"
}
```


F-Interop specific RSpecs will be made available as part of the SDK.

A successful reply will look like this:

```json
{
  "id": 75,
  "expire": "2016-10-22T15:06:34Z",
  "created": "2016-10-22T13:06Z",
  "speakingFor": "urn:publicid:IDN+wall2.ilabt.iminds.be+user+bvermeul",
  "project": "bvermeul",
}
```
### 3.3.3 Status

An update of the status of the experiment (it takes some time to launch it, so you want to know when it’s ready), can be asked through the status call:

```
curl -X GET --cert-type pem --cert <YOUR AUTHORITY PEM> \   
https://jfedapi.ilabt.iminds.be/api/experiment/75
```

Of course, the ID should be the one returned from the status call.

A successful reply will give you the below JSON

```
{
   "id": 75,
   "expire": "2016-10-22T15:06:34Z",
   "created": "2016-10-22T13:13:06Z",
   "manifestRspec": "<?xml version='1.0'?>
<rspec type="manifest" generated_by="jFed RSpec Editor" generated="2016-10-22T15:14:28.327+02:00">
<emulab:enabled>true</emulab:enabled>
<delay:command>
</delay:command>
<client:enabled>true</client:enabled>
<sharedvlan:enabled>true</sharedvlan:enabled>
</RSPEC>
```

Where the manifest RSpec lists the actual resources reserved and the way how to reach them.
3.3.4 Renew

The renew call makes it possible to extend the reservation of resources:

curl -X PUT -H "Content-Type: application/json" --data @renew-experiment-data.json \  --cert-type pem --cert <YOUR AUTHORITY PEM> \  https://jfedapi.ilabt.iminds.be/api/experiment/75

File 'renew-experiment-data.json'

```json
{
  "id": 75,
  "expire": "2016-10-22T15:06:34Z"
}
```

3.3.5 Delete

The delete call makes it possible to stop an experiment and the reservation of resources for it:

curl -X DELETE --cert-type pem --cert <YOUR AUTHORITY PEM> \  https://jfedapi.ilabt.iminds.be/api/experiment/75

3.3.6 Graphical testbed user interface

We can of course combine this TBaaS engine with the use of graphical user interface tools from Fed4FIRE. E.g. when an experiment is created through the API, it can be recovered by the jFed GUI (http://jfed.iminds.be) as can be seen below. The user can access the machine through ssh by simply clicking the green node in the GUI. In this way, it will be possible to give easy ssh access to an instance of F-Interop for development of new tests.
Figure 4: jFed GUI for easy interaction with testbed resources
4 Software development kit (SDK)

4.1 Overview of the SDK

The SDK provides a powerful platform to contributors by deploying a private instance of all F-Interop core components. This gives them the possibility to develop their contribution without interference, and gives them root-access to all components for easy debugging.

The components that are deployed are:
- A RabbitMQ AMQP-server;
- A Session orchestrator;
- A GUI Instance.

Additionally, the SDK also provides a few supporting scripts that can be used to:
- Create a user
- Create a session
- Start a session
- Log the AMQP messages of a session
- Start a testsuite
- Start a testcase

The full instructions on how to deploy an SDK instance can be found in 5 Deploying the software development kit. In 6 Developing a third-party testing tool using the SDK, we provide an overview of the components of a testing tool by exploring the structure of the coap_testing_tool. In this annex we provide references to the most interesting and important parts of the source code and documentation, as a guide to third-party developers. This documentation is also part of the SDK.

4.2 Contents of the software development kit

The SDK contains an Experiment Specification (ESpec) which can be used to provision servers on the imec testbed infrastructure, on which it subsequently installs all the f-interop core components.

This ESpec will request three servers, called ‘server’, ‘gui’ and ‘ansible’. The various F-Interop core components have been distributed among these two, while the third server is used as a controller to install all the software.

This controller uses Ansible, which is a popular “configuration management software” which allows the easy deployment of software on servers. Each f-interop core component which has been developed provides the necessary Ansible-files to do the deployment. These files are used for the deployment of the testing and production servers. The SDK gathers all the core components from the different repositories in which they are developed, and subsequently uses the same Ansible deployment scripts. The SDK also makes sure that the deployment is coordinated and populated with the correct deployment configuration files and variables (like service endpoint locations, account details, file locations, etc). As such, the SDK is a perfect replica of the production environment, but with the added benefit for the developer of having a separate instance where he is not interfered by other users and has full administrator permissions on all services. This helps with debugging problems, investigating unexpected behavior, etc.

In the sections below, we describe on which server each component is installed, and where you can find the most important files; which (web-) interfaces are exposed and which commands can be used to query the component. Note that the part “YOURPROJECT.wall2-ilabt-iminds-be.wall1.ilabt.iminds.be” in the URL’s below is generated by the testbed software on which the SDK is deployed. It will be different for every instance that is deployed.
4.2.1 AMQP-server

The RabbitMQ AMQP-server is installed on server.

For debugging purposes, you can access the RabbitMQ management interface WebGUI on http://server.YOURPROJECT.wall2-ilabt-iminds-be.wall1.ilabt.iminds.be:15672.

There is a default admin user with username and password f-interop that can be used to connect to the AMQP-server and the WebGUI.

4.2.2 Session Orchestrator

The session orchestrator is installed on server in /home/f-interop/finterop/finterop/orchestrator.

The orchestrator components are managed by supervisord. More info on how this supervisord-instance is configured can be found in /etc/supervisor/conf.d/finterop_orchestrator.conf.

The current sessions on the orchestrator are also managed by a second -internal- supervisord-instance. Each active session has a configuration file stored in /home/f-interop/finterop/finterop/orchestrator/conf.d.

The orchestrator uses supervisord to manage the processes it executes. Use the command supervisorctl to check these processes.

4.2.3 GUI

The GUI is installed on gui in /home/myslice. The main configuration details can be found in /etc/myslice/main.cfg.

The following tools are available for use via web-interfaces:

- RethinkDB via: http://gui.YOURPROJECT.wall2-ilabt-iminds-be.wall1.ilabt.iminds.be:8080
- MySlice: http://gui.YOURPROJECT.wall2-ilabt-iminds-be.wall1.ilabt.iminds.be:8111
  o (login: support@myslice.info, password: superpassword)

Via SSH you can do the following:

- Manage MySlice processes with supervisorctl command. (ex. supervisorctl restart all). The configuration of supervisord is stored in /home/supervisor/conf.d/myslice.conf
- Read the log files in /var/log/myslice

4.2.4 Supporting scripts

The supporting scripts can be found on ‘ansible’ in ~/session. These scripts allow to emulate interactions between the GUI and Session Orchestrator, which are documented at http://doc.f-interop.eu/#session-orchestrator-so. This makes it possible to do automated tests without the need to interact with the GUI.

It contains the following scripts:

- setup-user.sh [username]: Creates an user with the given username in the session orchestrator
- setup-session.sh [username] [sessionname]: Creates a session with the given sessionname, and adds the user to this session
- start-session.sh [sessionname]: Sends the "{status:started}" message to the session orchestrator, so that it spawns the testing tools and other requested resources.
- show-logs.sh [amqp-url]: This logs all messages sent to the AMQP-channel to the console
- start-testsuite.sh [sessionname]: Starts the testsuite in the session orchestrator
- start-testcase.sh [sessionname]: this script will open a dialog box requesting the name of the testcase that you want to start. By default testcase ‘TD_COAP_CORE_01_v01’ from the coap_testing_tool is suggested.

The SDK also provides a simple script setup.sh that combines the 4 first scripts into a logical sequence. It takes two arguments: a username and sessionname: setup.sh [username] [sessionname]

```bash
#!/bin/bash -x

USER=${1:-user1}
SESSION=${2:-session1}

./setup-user.sh $USER
./setup-session.sh $USER $SESSION
./show-logs.sh "amqp://f-interop:f-interop@localhost/$SESSION" &
./start-session.sh $SESSION

wait
```
5 Deploying the software development kit

In this section we describe the steps that a third party needs to undertake to start an SDK-instance.

5.1 Registering an Fed4FIRE account

The SDK can be deployed on the imec testbed infrastructure. To access this infrastructure you need to create an Fed4FIRE account on the imec user authority.

Go to [http://authority.ilabt.iminds.be](http://authority.ilabt.iminds.be) and click Sign up.

Fill out the form. When asked for project information, choose Join an existing project and enter finteropsdk.

![Figure 5 Registration form on the imec Authority website](image)

After confirming your e-mail address, your account will be validated manually. Please allow for a few business hours for this to happen.

5.2 Install jFed

Provisioning of resources on the imec testbed infrastructure is done via the jFed Experimenter GUI.


5.3 Start the SDK in jFed

Download the latest version of the SDK from the SDK repository, which can be found at [https://gitlab.f-interop.eu/f-interop-contributors/sdk/](https://gitlab.f-interop.eu/f-interop-contributors/sdk/).

Start jFed, and login via the Login with Fed4FIRE-credentials-button.
This will open the imec user authority-website on which you must login. JFed will then retrieve your login certificate (an X.509 certificate and associated private key, which will be used for communicating with the imec testbeds).

The downloaded login certificate is encrypted, and must be unlocked with your password. JFed will open a second login window, in which you must enter your password a second time.
Now the main jFed Experimenter GUI is shown. Select Open ESpec in the upper button bar. Enter the folder location of the SDK in the dialog (e.g. the folder containing experiment-specification.yml).

![Local login screen in jFed](image)

**Figure 8 Local login screen in jFed**

![Opening an ESpec in jFed](image)

**Figure 9 Opening an ESpec in jFed**
In the following screen, enter a globally unique name for your SDK-deployment (tip: use your initials in the name). Select an appropriate duration for your deployment. imec enforces a maximum duration of 90 days, but you can extend the duration of the deployment indefinitely by 'renewing' the experiment when it nears its expiration date (but never more than 90 days in the future).

![Figure 10 Configuring an experiment for the F-interop SDK in jFed](image)

jFed will now start with allocate the necessary hardware, and provision them with the f-interop software. This can take up to an hour to complete. Wait until jFed indicates that everything has finished before you continue.
5.4 Access the SDK resources

The F-Interop components can now be accessed on the reserved servers. These servers can be accessed through SSH by double-clicking on the icons in the jFed GUI. You will be authenticated via the private key in your login certificate. You have full sudo rights. To obtain root access execute `sudo su`.

![Figure 11 An SDK being deployed by jFed](image)

The resources of the imec Virtual Wall testbed infrastructure do not have public IPv4 addresses, but they do have public IPv6 connectivity. However, some F-Interop services only bind to IPv4 interfaces, making them unreachable via IPv6.

The easiest way to access IPv4-resources the imec testbed resources is by setting up an SSH-tunnel with dynamic port forwarding on Linux or Windows, and configuring your browser and tools to use this SOCKS proxy. Instructions for Linux can be found at [https://help.ubuntu.com/community/SSH/OpenSSH/PortForwarding#Dynamic_Port_Forwarding](https://help.ubuntu.com/community/SSH/OpenSSH/PortForwarding#Dynamic_Port_Forwarding), instructions for Windows at [http://dimitar.me/dynamic-port-forwarding-with-socks-over-ssh/](http://dimitar.me/dynamic-port-forwarding-with-socks-over-ssh/).
6 Developing a third-party testing tool using the SDK

An f-interop testing tool consists out of the following components:

- an index.json-file describing the testing tool
- a supervisord-configuration file to start the testing tool docker containers
- one or more docker containers that connect to an AMQP event bus to send/receive control and data-messages

The coap_testing_tool (cfr. https://gitlab.f-interop.eu/f-interop-contributors/coap_testing_tool) can be used as a reference implementation. We will explore this repository in this annex, and refer to the most important files, documentation and source code where appropriate.

6.1 index.json

The index.json-file describes the testing tool and it's configuration options. It's mainly used by the GUI to populate the testing tool list, and allows you to expose a few configuration options via ui_information2.

Currently, these index-files are manually added to the orchestrator, and can be found in the templates-directory3. Later on, this index file will be automatically imported from the root of the testing tool repository.

An example index.json4 looks as follows:

```
{
  "_type": "testsuite.manifest",
  "testing_tool": "CoAP Testing Tool",
  "version": "0.0.8",
  "test_type": "conformance",
  "protocols_under_test": [
    "CoAP",
    "CoAP_CORE",
    "CoAP_OBS"
  ],
  "protocols_info": [
    {
      "protocol": "CoAP_CORE",
      "specification_id": "RFC7252",
      "test_description_id": "TD_COAP_CORE",
      "test_description_ref": "http://doc.f-interop.eu/coap_testing_tool/extended_test_descriptions/TD_COAP_CORE.yaml"
    },
    {
      "protocol": "CoAP_OBS",
      "specification_id": "RFC7641",
    }
  ]
}
```

---

2 See GUI-implementation in file https://gitlab.f-interop.eu/f-interop/gui.git


4 This file can be found at https://gitlab.f-interop.eu/f-interop/f-interop_ietf.git
"test_description_id": "TD_COAP_OBS",
"test_description_ref": "http://doc.f-interop.eu/coap_testing_tool/extended_test_descriptions/TD_COAP_OBS.yaml"
],
"underlying_supported_protocol": [ "ipv6",
"udp"
],
"agent_names": [ "coap_client_agent",
"coap_server_agent"
],
"iut_roles": [ "coap_client",
"coap_server"
],
"ui_information": [ {
"field_name": "testingtool.description",
"description": "Testing Tool Description",
"type": "text",
"value": [ "Reference-based interoperability (~conformance) Testing Tool for validating conformance to CoAP RFCs."
"The testing tool uses as reference implementation (golden image) Californium implementation (CoAP client and CoAP server)."
"Reference implementation is hosted at F-Interop plaform"
"The implementation of the tests are based on ETSI test description :
] },
{ "field_name": "testingtool.version",
"description": "Testing tool version (optional)",
"type": "selection",
"value": [ "0.0.8",
"0.0.7",
"0.0.6",
"0.0.5",
"0.0.4",
"0.0.3",
"0.0.2",
"0.0.1"
],
"mandatory": false
},
{ "field_name": "testsuite.testcases",
"description": "Test cases selection (optional)",
"type": "multi-selection",
"value": [ "http://doc.f-interop.eu/tests/TD_COAP_CORE_01",
"http://doc.f-interop.eu/tests/TD_COAP_CORE_02",
"http://doc.f-interop.eu/tests/TD_COAP_CORE_03",
"http://doc.f-interop.eu/tests/TD_COAP_CORE_04",
"http://doc.f-interop.eu/tests/TD_COAP_CORE_05"
] }
6.2 supervisord configuration file

The supervisor configuration file is located in the same location as the index.json file, and is called \texttt{supervisor.conf.j2}.

The \texttt{.j2} suffix indicates that the configuration file is template, and will be processed by a jinja2-engine to create the final supervisord-configuration file.

The following variables will be passed to the template:

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>session</td>
<td>The name of the session, typically an UUID</td>
</tr>
<tr>
<td>amqp_url</td>
<td>The URL of the AMQP-event bus to which the testing tool must connect</td>
</tr>
<tr>
<td>amqp_exchange</td>
<td>The name of the exchange on the AMQP-event bus to which the testing tool</td>
</tr>
</tbody>
</table>
Typically, the `amqp_*` variables will be passed down to the docker containers as an environment variable, like in this example:

```
[program:{{ session }}|testing_tool]
stopsignal=TERM
killasgroup=true
autostart=false
stdout_logfile = %(here)s/logs/{{ session }}-testing_tool-stdout.log
stderr_logfile = %(here)s/logs/{{ session }}-testing_tool-stderr.log
command = docker run
    --env AMQP_URL={{ amqp_url }}
    --env AMQP_EXCHANGE={{ amqp_exchange }}
    --rm
    --privileged=true
    --sysctl.net.ipv6.conf.all.disable_ipv6=0
    --name="session_{{ session }}-testing_tool-conformance-coap"
    testing_tool-conformance-coap
```

### 6.3 Docker images

The testing tool docker containers are expected to connect to the AMQP-event bus which has been passed down via the supervisor-configuration file. The number of docker containers is currently not limited, and you can configure them as required via the supervisor configuration file.

These docker containers will be retrieved from the f-interop docker image registry. Currently, new images must pushed manually.

In the reference implementation `coap_testing_tool`, multiple docker containers are built. In the root of the repository, you find a Dockerfile for the container which hosts the test coordinator, an agent, and message capturing and analysis tools. In the subfolders `automated_IUTs`, you will find the Dockerfile's of the containers hosting the reference CoAP servers/clients.

*Note: All docker images use an `supervisord` instance internally to manage the different processes. Environment variables of the supervisor-process are automatically passed down to child processes, but are also available as variables with the prefix `ENV_`. This last form is used to pass the parameters down to the agent in the inner supervisor configuration file:*

```
command = sh -c "sleep 3;python -m agent connect --url %(ENV_AMQP_URL)s --exchange %(ENV_AMQP_EXCHANGE)s --name agent_TT"
```

### 6.4 Testing tool interface

After connecting to the AMQP-event bus, a testing tool must respond to and send when appropriate the minimal message events outlined in the documentation. These include reporting to be ready, accepting configuration options and starting and stopping a test suite.

In the `coap_testing_tool` reference implementation, the testing tool interface is implemented by the `test_coordinator`. This component is implemented as a state machine of which the states and transitions can be found at the end of the file `states_machine.py`.

---


When starting a test suite, the test coordinator sends a `tun.start` message to the agents of the different components involved in the interoperability test, this sets up the tunnels over which the network traffic between the devices under test and/or the reference devices will be sent. How these tunnel interfaces are configured is currently hardcoded in `notify_tun_interfaces_start` of `amqp_connector.py`.

### 6.5 Sending and receiving network traffic

All network traffic is to be encapsulated and sent over the AMQP event-bus. This encapsulation is done via the `f-interop agent`. The agent currently has two modes: `tun` (default) and `serial`. The `tun`-mode will setup a local layer 2 tunnel that encapsulates all messages and transports them via the AMQP-event bus. Sending network traffic to other parties of the interoperability test is done by sending traffic to the local tunnel address (the address of the local tunnel is determined by the `tun.start` message sent by the test coordinator to the agent).

The link between your testing tool code and the AMQP-event bus is made via an `f-interop agent` in the docker container. You can use the reference `f-interop agent`, or you can implement one yourself by looking into the AMQP-event bus messages documentation.

### 6.6 Executing a testcase

Executing a test case will typically involve sending and/or responding to one or messages between the devices/implementations being tested. In `coap_testing_tool` the reference implementations use the Python class `AutomatedIUT` to generate the initial messages.

For example, the `coap_client_californium` implements the `execute_stimuli`-method in which it starts a Californium client-process performing the requested test case. The `coap_server_californium` is simpler, as it only needs to start the Californium server when starting.

### 6.7 Analyzing the network messages

The messages sent and received by the devices/implementations must then be analyzed and assessed. In `coap_testing_tool`, the messages are captured by the sniffer and are subsequently processed.

---

8 Folder `coap_testing_tool/test_coordinator` in https://gitlab.f-interop.eu/f-interop-contributors/coap_testing_tool.git

9 Implemented with https://github.com/pytransitions/transitions

10 See line 489 of file `coap_testing_tool/test_coordinator/states_machine.py` in https://gitlab.f-interop.eu/f-interop-contributors/coap_testing_tool.git


12 See repository https://gitlab.f-interop.eu/f-interop-contributors/agent

13 See http://doc.f-interop.eu/#agent

14 See file `automated_IUTs/automation.py` in https://gitlab.f-interop.eu/f-interop-contributors/coap_testing_tool.git

15 See file `automated_IUTs/coap_client_californium/automated_iut.py` in https://gitlab.f-interop.eu/f-interop-contributors/coap_testing_tool.git

by the test analysis tool, which then issues a verdict, which is sent as a
`testcoordination.testsuite.report`-message back to the GUI.

The test analysis tool is called `ttproto`, and the implementation of the AMQP-messages can be found in `tat_amqp_interface.py`\(^\text{18}\)

\(^{17}\) See http://doc.f-interop.eu/#testcoordination-testsuite-report

\(^{18}\) See `ttproto/tat_amqp_interface.py` in https://gitlab.f-interop.eu/fsismondi/ttproto.git
7 Using testbed resources

In a similar way that the SDK uses testbed resources to deploy the different components, also devices on the testbeds can be used to be part of the testing.

In the screenshot below you see a similar SDK deployed, but now an extra node has been added, called sensor. This is a Linux box on the imec w-iLab.t testbed with 2 Zolertia Re-motes attached. One of them configured as a border router (doing wireless to USB bridging) and one as the device under test (a CoAP server e.g.). Through the Linux box different firmware can be pushed to the sensors.

With this setup a CoAP test can be launched, where the client side is launched as a docker container on the orchestrator, and the server side is a real sensor device on the testbed.

Of course, the variations on the exact use (6TiSCH, privacy setup, etc) are endless.

The advantage of this setup is that the user can do all this testing remotely with a variety of devices.


Figure 12 An SDK with a Zolertia Re-mote on the imec w-iLab.t testbed
8 Conclusion

This deliverable describes the work that has been done in task 4.2 “Testbed as a service integration and SDK” of WP4 “Testbed as a service”.

In this deliverable we discussed how the “Testbed As A Service” component interacts with the other F-Interop components, as outlined in the F-Interop architecture described in D1.3. Subsequently, we described the two API's (high-level and low-level) that were developed to ease the integration of FIRE+ testbed resources in F-Interop.

We describe the contents of the software development kit (SDK), and how it is deployed. We also describe how the software development kit (SDK) can be used by (open-call) contributors for the development of third party extensions.