HORIZON 2020

Information and Communication Technologies

Integrating experiments and facilities in FIRE+

Deliverable D1.3 Initial architecture design

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:	**31/10/2016**
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.

Co-funded by the European Union

Co-funded by the Swiss Confederation

Document properties

Responsible partner	imec
Author(s)/editor(s)	Brecht Vermeulen (imec), Thijs Walcarius (imec), Wim Van de Meerssche (imec), Federico Sismondi (INRIA), Remy Leone (INRIA)
Version	1
Keywords	Initial Architecture, SDK, APIs

Abstract

This deliverable is the first of two deliverables in the task 1.3 Architecture and FIRE+ integration design (D1.4 with final architecture is planned for the end of the project). The requirements studied in task 1.1 and 1.2 were used as input. We describe first the different location models (location of devices and users) that we will support in F-interop, followed by the current high level architecture of components in F-Interop and integration with FIRE+ testbeds. After that, we go into more details on the specific interaction between the components. This interaction is based on the AMQP message protocol and we have defined the message formats, functioning as an API when contributors want to add functionality to F-Interop.

The deliverable is also meant as part of the software development kit (SDK) towards (open-call) contributors.

Table of Contents

List of Figures

List of Acronyms

AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
CA	Consortium Agreement
CoAP	Constrained Application Protocol
ComSoc	Communications Society
DESCA	Development of a Simplified Consortium Agreement
DHCP	Dynamic Host Configuration Protocol
DHT	Distributed Hash Tables
DNS	Domain Name System
DNSSec	Domain Name System Security Extensions
DPA	Data Protection Authorities
DPO	Data Protection Officer
DUT	Device Under Test
EC	European Commission
ENISA	European Union Agency for Network and Information Security
ETSI	European Telecommunications Standards Institute
EU	European Union
FP7	Seventh Framework Programme
GA	Grand Agreement
GA	General Assembly
GPS	Global Positioning System
HTTPS	Hypertext Transfer Protocol Secure
ICT	Information and Communication Technologies
ID	Identifier
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IERC	European Research Cluster on the Internet of Things
IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
IPC	Intellectual Property Committee
IPM	IPR Monitoring and Exploitation Manager
IPR	Intellectual Property Rights
IPSEC	Internet Protocol Security
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
ISO	International Standards Organization
ISP	Internet Service Provider
IT	Information Technology
ITU	International Telecommunication Union
IUT	Implementation Under Test
KPI	Key Performance Indicator
LSPI	Legal, Security and Privacy Issues

MAC	Media Access Control
MSc	Master of Science
M2M	Machine to Machine
OASIS	Organization for the Advancement of Structured Information Standards
OECD	Organization for Economic Cooperation and Development
OS	Operating System
OSN	Online Social Network
PC	Project Coordinator
PCAP	Packet CapturePCP Partner Contact Person
PDPO	Personal Data Protection Officer
PERT	Program Evaluation Review Technique
PhD	Doctor of Philosophy
PM	Person Month
PMB	Project Management Board
PPR	Periodic Progress Report
PRAAT	Privacy Risk Area Assessment Tool
P&T	Post & Telecom
QoS	Quality of Service
RAND	Reasonable and Non Discriminatory
RFC	Request For Comments
RSpec	Resource Specification
R&D	Research & Development
SDK	Software Development Kit
SME	Small Medium Enterprise
SMS	Short Message Service
SOTA (or SoA)	State Of the Art
SSL	Secure Sockets Layer
TBaaS	TestBed as a Service
TC	Technical Coordinator
TCP	Transmission Control Protocol
TL	Task Leader
TLS	Transport Layer Security
Tor	The Onion Router
TRL	Technology Readiness Level
UK	United Kingdoms
UN	United Nations
UNCTAD	United Nations Conference on Trade and Development
UPRAAT	Universal Privacy Risk Area Assessment Tool
URL	Uniform Resource Locator
US	United States
VoIP	Voice over Internet Protocol
WES	Women’s Engineering Society
WiTEC	Women in science, Engineering and Technology
WoT	Web of Trust
WP	Work Package
WPL	Work Package Leader

W3C	World Wide Web Consortium
XML	Extensible Markup Language
YAML	YAML Ain't Markup Language, human friendly data serialization

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

WP1 has the following goals:

- Analyze and specify the online testing tools requirements
- Analyze and specify personal data protection and security requirements
- Design and specify the F-Interop architecture

1.2.2 Task Objectives

Task 1.3 in WP1 has the specific goal of defining the architecture and the integration with FIRE+ testbeds. For this, a milestone MS2 about the initial architecture agreement was reached at month 9 of the project, this deliverable D1.3 is delivered at M12 and there is an upcoming milestone MS3 at M24 for an architecture update based on the progress in the project and a final MS4 and D1.4 with the final architecture at M33 of the project.

1.2.3 Deliverable Objectives and Methodology

The requirements studied in task 1.1 and 1.2 were used as input. First, we will highlight the current high level architecture of components in F-Interop and integration with FIRE+ testbeds. After that, we go into more details on the specific interaction between the components. This interaction is based on the AMQP message protocol and we have defined the message formats, functioning as an API when contributors want to add functionality to F-Interop.

The deliverable is also meant as part of the software development kit (SDK) towards (open-call) contributors.

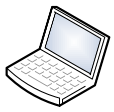
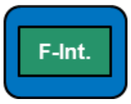
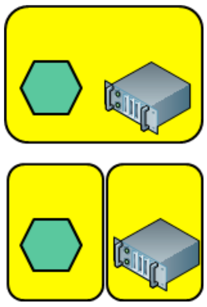
2 Location models

2.1 Introduction

To start, we identified all location models (which device and tool are located where) that F-Interop should support for its goal of developing a new experimental platform for online interoperability tests and validation tools, remote compliance and conformance tests, scalability tests, Quality of Service (QoS) tests, SDN/NFV interoperability tools, online privacy test tools, energy efficiency tools.

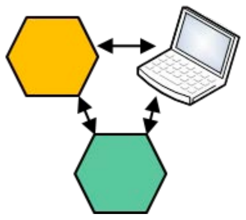
2.2 Symbols used

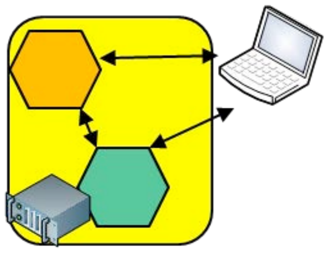
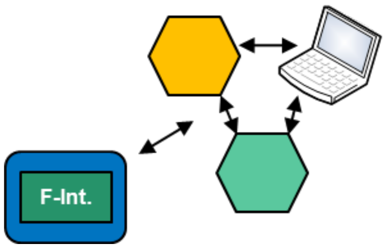
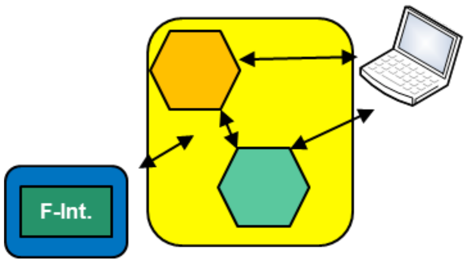
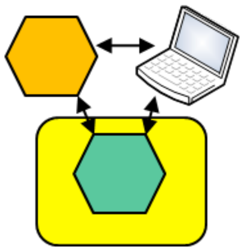
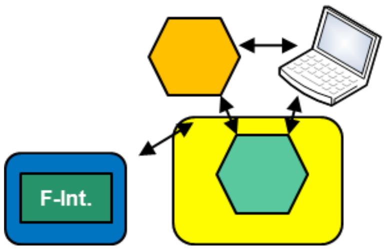
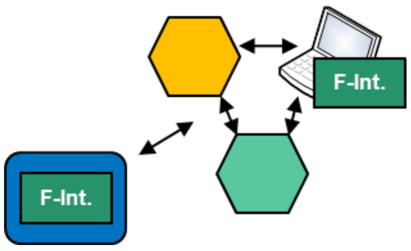
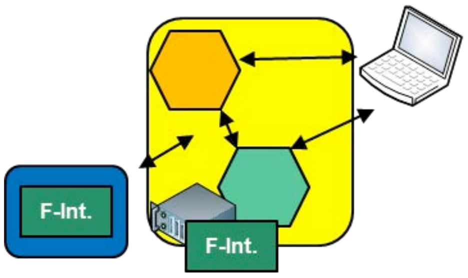
The following symbols will be used:

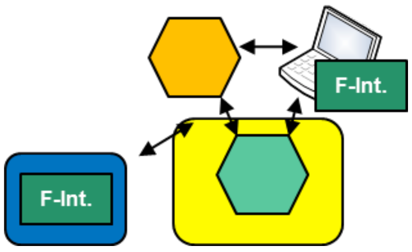
	User laptop
⬢	Device(s) under test
⬢	Other device(s) needed for test
	F-Interop central services
	Testbed with devices (e.g. IoT devices, servers, virtual machines, network devices, …). The second picture shows that also multiple testbeds can be involved, e.g. a Cloud testbed runs the test suite, while an IoT testbed is used for the IoT devices. The testbeds can be interesting when they offer devices that the user has not available on his desk or when more devices are needed for a test (e.g. scalability) then locally available or when you want to remotely collaborate with other users or when you want to do multiple tests or developments in parallel.

2.3 Location models with a single user

We will start off illustrating the different location models we foresee to be supported by the F-Interop architecture and tools, in case of a single end-user, e.g. in case of conformance testing or in case of interoperability testing with known other devices.

	A. User downloads all necessary tools and software from F-interop website and executes everything locally and processes the interaction locally (e.g. for development of new tests)

	B. User downloads all necessary tools and software from F-interop website and executes everything remotely in a testbed and processes the interaction locally or in the testbed (e.g. for development of new tests)
	C. User downloads all necessary tools and software from F-interop website and executes all locally and uploads the interaction capture (e.g. pcap file) to F-interop (e.g. User wants to orchestrate his own interactions)
	D. User downloads all necessary tools and software from F-interop website and executes all remotely in a testbed and uploads the interaction capture (e.g. pcap file) to F-interop (e.g. User wants to orchestrate his own interactions, but on devices he has not at his desk)
	E. User downloads all necessary tools and software and tests local DUT (Device Under Test) against device in testbed (or in central platform) and processes the interaction capture locally
	F. User downloads all necessary tools and software and tests local DUT against device in testbed (or in central platform) and uploads the interaction to F-interop for analysis
	G. Similar to C, but now a F-interop proxy(s) executes all tests and uploads the interaction to F-interop (e.g. Automatic test with minimum interaction of user)
	H. Similar to D, but now F-interop proxy(s) runs in remote testbed and executes all tests and uploads interaction to F-interop (e.g. Automatic remote test)

	I. Similar to F, but now F-interop proxy(s) executes all tests and uploads pcap to F-interop

Of course, in the F-Interop project different technologies are addressed. Not all those technologies are able to be tested in all proposed ways. E.g. CoAP is an application layer protocol that runs over IP and can thus be transported over the internet for e.g. scenario E and F above. On the other hand, 6TISCH is closer to link layer and so the devices should be in the same room for most of the tests.

2.4 Location models with 2 or more users

While the previous section talked about a single user testing against a golden implementation or device (for conformance testing) or another implementation or devices at his desk or on a testbed (for interop or performance testing, etc), it is also possible that 2 or more users test their own implementation against each other. In this phase of the project we consider only 2 users for this kind of interop tests.

The setups are very similar to the ones listed in the previous section, but now e.g. the orange and green devices are configured and controlled by 2 different users. Especially on the scenario and interaction side, this has some implications, but F-Interop will develop its tools immediately with these cases in mind, meaning that the single user case is in fact a special case of the 2 user case. A single user has then to do all handling instead of 2 users, or in case of full automation (the F-Interop goal), has to run a proxy per device.

3 High level architecture

3.1 High level architecture without testbeds

This section will describe the high level architecture for F-Interop tests without any testbeds involved.

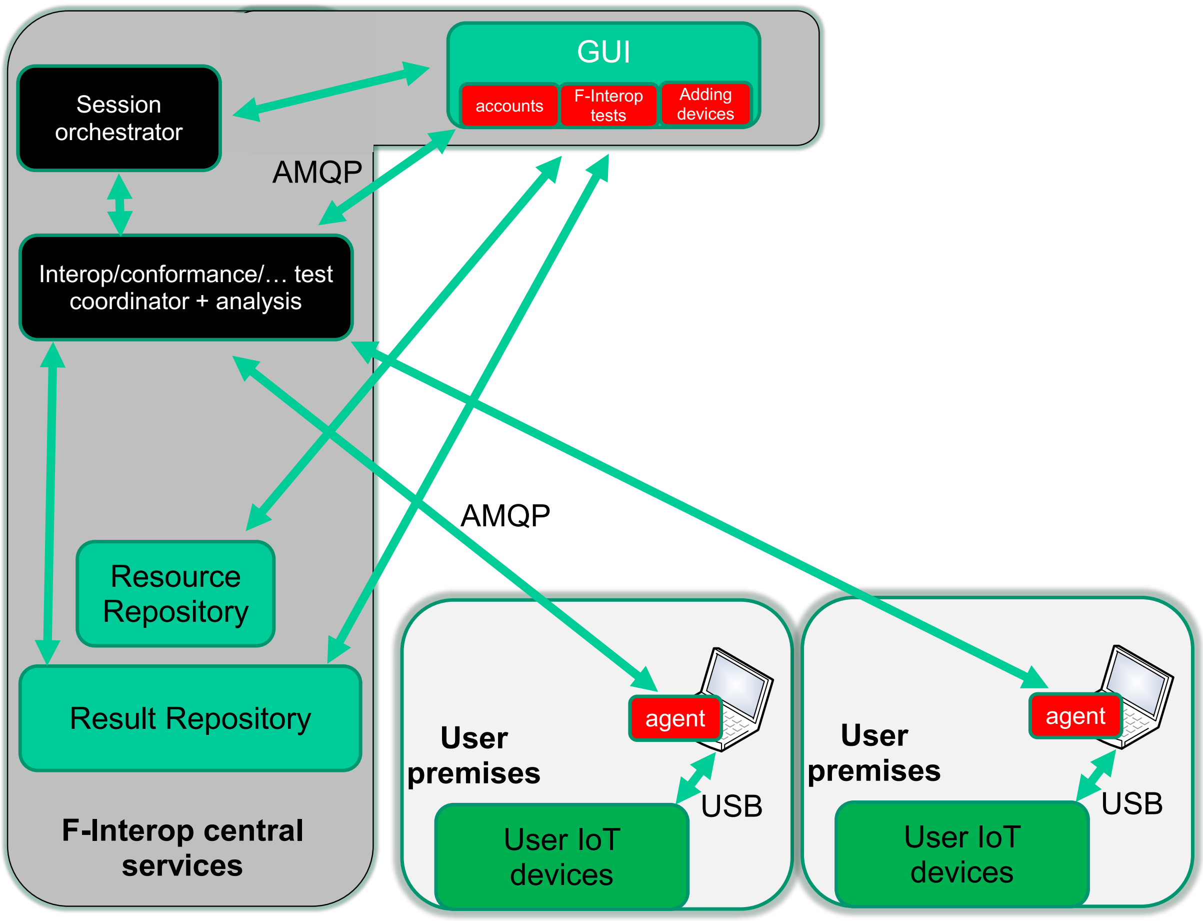


Figure 1: High level architecture without testbeds

Figure 1 shows the components that are involved at the central F-Interop services side and the user premises side (this figure shows 2 remote users doing interop tests, but the other location models are easy to derive). We identify the following components:

- The **graphical user interface (GUI)** is the first contact point for an end-user and makes it possible to register for an account, list all the F-interop tests available and register your own devices (if any). The registration of your own device can be for a single test or if you want to contribute this device so other people can also test against it, this is also possible
- The **resource repository** contains a list of all devices and their properties that can be used for F-interop tests. The resource repository communicates with the GUI.
- The **session orchestrator** plays an administrative role: it monitors the users that are connected, activates the rooms currently in use and starts/stops the test sessions.
- **The test coordinator and analysis component** is the base of an individual (interop, conformance, performance, etc.) test and does first the orchestration (what should be done next according the test script, e.g. reset device) and afterwards the analysis (e.g. by analysing a pcap file which was sniffed during the interaction). Per type of test, this orchestration and analysis components have a different implementation.

- The **agent** is a software component developed by F-Interop as well. It runs near devices (e.g. at the user premises) to communicate with the orchestration and analysis components. It can do different tasks depending on the test: e.g. sniff all traffic and send a pcap to the analysis step, or e.g. full automatic orchestration of the device during the test).
- When the test is done, the results are stored in the central **result repository** for future referral. The test results are only accessible by the people taking part in the test.

For the communication between the components, AMQP (Advanced Message Queuing Protocol, www.amqp.org) has been selected as a message bus protocol. It will be used for both control message interaction as for the transmission of wrapped data packets between devices.

3.2 High level architecture with testbeds

Figure 2 starts from the basic architecture from the previous section, but adds testbed devices in the picture. The figure illustrates the setup when a contributor wants to extend an existing test, add a new test or debug in detail what goes on during testing, while using one or more devices at his desk as well during the test (development).

An extra component **TBaaS** (Testbed as a Service) is 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.

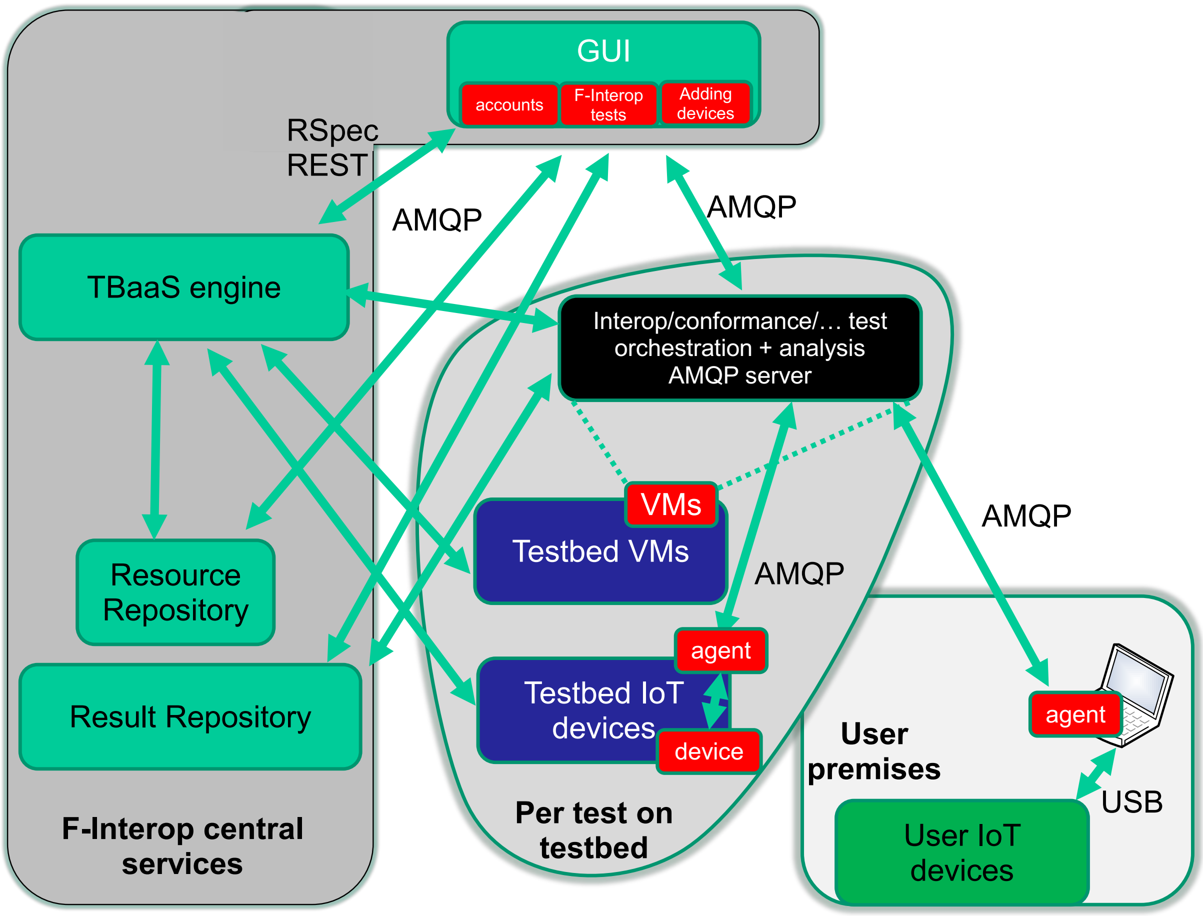


Figure 2: High level architecture with testbeds

The TbaaS component has two REST APIs on different levels:

- The low-level API is F-Interop independent and a testbed specific RSpec (Resource Specification) has to be specified (which then contains references to the software to be installed). This API is detailed in Annex 2: TBaaS low level RSpec API.

- 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. This API is detailed in Annex 1: TBaaS high level API.

Both APIs can be used to deploy a full F-Interop system, e.g. for development, or for only reserving and allocating a couple of devices on a testbed for use during a test.

The way in which the F-Interop tools and tests are packaged is with Ansible (https://www.ansible.com). In Ansible you can specify Ansible Playbooks described in YAML for to automate tasks. In this way we define Playbooks for installing AMQP, F-Interop orchestrators and analysis tools, and it becomes very easy to deploy the F-Interop framework automatically.

This is also the way to extend the F-Interop test portfolio: create and develop new tools, but package them in Ansible for easy installation and replication. Thus, it is also easy to track and use different versions of a tool.

As shown on the Figure 2, the TBaaS engine feeds also the Resource Repository with all devices available on the testbeds. This is performed 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 the Advertisement RSpec to the right JSON format and finally send this data as an AMQP message towards the Resource Repository.

The other components are similar to the previous section.

Figure 3 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.

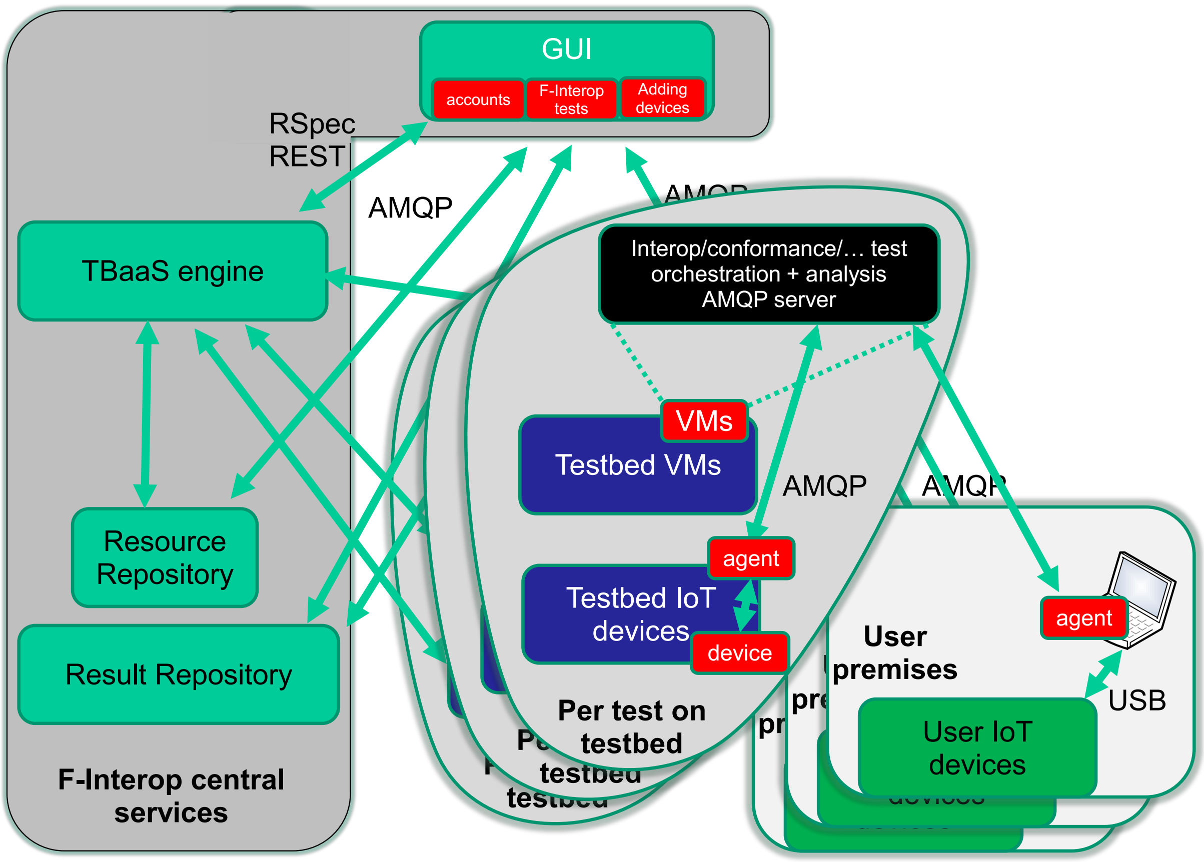


Figure 3: High level architecture with multiple tests on testbeds running in parallel

3.3 Hybrid high level architecture

If we combine now both previous architectures, we get Figure 4. This gives us full flexibility on the choice of resources and deployment of those resources.

The Session orchestrator spawns now central test coordinators and can allocate and provision testbed resources through the TBaaS engine depending on the needs. The GUI or Session orchestrator needs to map the demands of a user for a specific F-Interop test to the resources. Resources can be found in 3 places (the Resource Repository will have this information):

- Can be run as central services (e.g. a software COAP server on a virtual machine or docker instance)
- Can be found on testbeds
- Can be found at user premises by users who contribute their devices

NOTE: as can be seen in Figure 4, the test orchestrator/analysis can be run as a central service for a specific F-Interop test (think a user just running a test), or can be deployed on a testbed virtual machine (think a user wanting to adapt a test or develop a new test).

This architecture is the one we will target for a first running version of F-Interop.

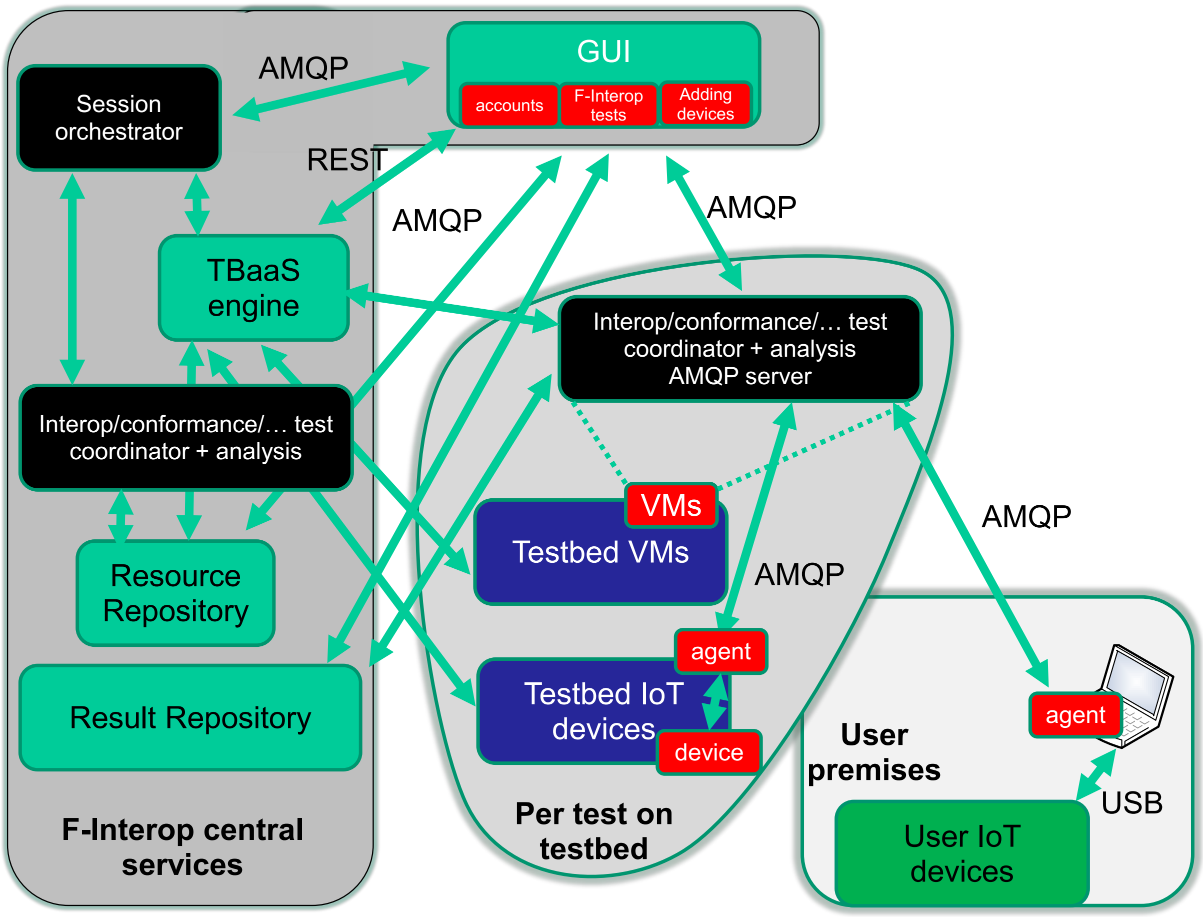


Figure 4: High level hybrid architecture

4 Detailed architecture of a single test implementation

In the previous sections, the high level architecture was discussed. In this section we will have a closer look at the components and APIs involved in a single test. Figure 5 shows the involved components:

- Event Bus to exchange messages (AMQP based, we use RabbitMQ, https://www.rabbitmq.com as an implementation)
- Web interface for the GUI
- Orchestrator which starts and stops test sessions
- Test specific test and analysis tools
- Devices on testbeds or at the desk of the user, which communicate through an Agent with the event bus and the other components

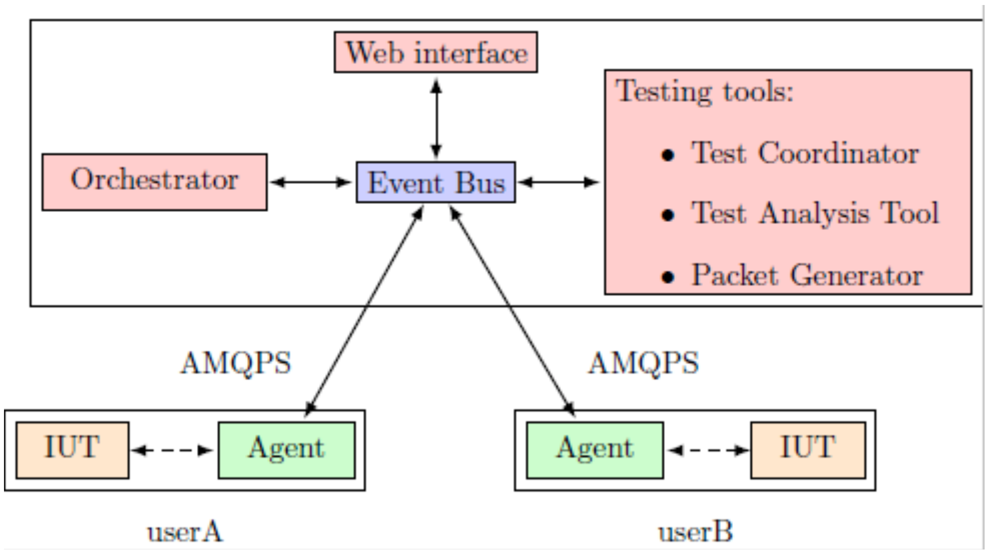


Figure 5: Detailed architecture of a single test

4.1 AMQP message bus

The message bus is used for two types of messages:

- Control plane messages relate to the management of an ongoing test session: e.g. start sniffer, signal the start/end of a test case, etc.
- Data plane messages contains the raw data exchanged between the implementations under test (IUT). The reason why it is possible to send data plane messages over the message bus, is that we send even link layer messages between two remote devices, or that we can use IPv6 even if the networks in between are not fully IPv6 enabled, and also that the messages are secured. It also allows to attach a sniffer during the test which can then send the data to an analyser.

JSON is the data format used for the application messages exchanged over the bus.

4.2 Test Coordinator

The **test coordinator** is the component that glues all parts together for a single test. Its purpose is to coordinate the test cases execution during a test session. It's a generic executor of the test case, although at the bootstrapping phase it retrieves a test session context from the orchestrator.

In the case of interoperability testing, it triggers actions in the following components:

- SNIFFER: This is the component handling the sniffing of the traffic exchanged between the participants of the test session
- GUI: Messages related to the execution (e.g. execute test case 2, step 5)
- AGENT: configure interface, start interface, launch automated step
- Test Analysis Tool (TAT): analyse test, e.g. captured traffic

This component is protocol agnostic, once instantiated it gets the specific context from the orchestrator and feeds from the Test Extended Descriptions (TED yaml files), see D2.1 for details. TEDs play a very important role in the architecture, they describe in detail each test case execution. TEDs are designed to be easy to write using YAML language, and they can also be written by the contributors.

4.3 Test Analysis Tool (TAT)

The TAT provides F-Interop with protocol analysis services. The TAT is the component that performs the verification of traces during a test session. F-Interop provides TATs for different protocols under test.

The TAT issues three types of verdicts:

- PASS when test purpose of the test case is verified
- FAIL when there is at least one fault
- INCONCLUSIVE when the behavior of the IUTs does not apply to the one described in the test purpose.

The architecture supports TATs which perform:

- step-by-step analysis
- analysis post mortem

TATs are created both by the F-Interop core team and by external contributors. The F-Interop API specification defines the format of the messages that a TAT will receive from the AMQP Event Bus, and the format of the messages it will produce.

The TAT may implement its own dissector, and in that case it has to implement all the services described for the Dissector component as well.

4.4 Protocol Dissector

This component analyses the packets captured by the sniffer and dissects the packets into protocol specific packets (e.g. dropping non-relevant packets) and feeds them to the TAT.

4.5 Agent

The agent is a program a user downloads from the F-Interop website, and which allows him/her to connect an IUT to the F-Interop event bus. Communication between the agent and the event bus is authenticated and secured. Through the agent, the F-Interop platform can (remotely) interact with the IUT, for example by changing configuration or injecting packets. Similarly, the agent reports events to the event bus, such as sniffed packets.

4.6 Packet generator

In some conformance tests, a packet generator component can be used to generate and send packets towards the IUT. The component can for example implement the behaviour of a CoAP client when the

IUT implements a CoAP server. A F-Interop client can purposely generate wrongly formatted messages to verify the correct behaviour of the IUT in such situations.

4.7 APIs in a single test

The communication (AMQP messages) between the components in this section is detailed at http://doc.f-interop.eu. This is called the "F-Interop API" in a single test tool.

5 Envisioned interaction for an end-user

An end-user who wants to do an F-Interop test (interop, conformance, performance, etc.) has to go through the following F-Interop session steps (also detailed in D1.1). All interaction is done through the F-Interop GUI. The end-user doesn't need to know the internal details of the F-Interop APIs.

- User registration and login
- Registration of devices at the user's desk if any
- User starts by discovering the available test suites and by selecting the one he/she wants to execute.
- User specifies/selects resources in the F-Interop-Platform that are needed for his/her F-Interop session including the location models (see before), testing tools, libraries, etc. During this phase F-Interop-Platform may request information from the user or provide information to user for a coherent selection of the required resources.
- The resources selected in the previous step are actually reserved.
- The instantiation of the F-Interop-Platform resources that fit best with the user needs is done.
- In case of local resources, the user has to download and execute the Agent locally which needs to talk to the local device.
- The online F-Interop test campaign is launched and the selected test suites are executed against the IUTs.
- Test execution information is analysed. The test results and verdicts are provided together with explanations in case of FAIL or INCONCLUSIVE verdicts or something wrong happened. A report can be provided upon request in case for example the user wants to apply for a certification/labelling program.
- Storage of the F-Interop session information (Session-id, User-id, FI-User's IUT-id, IUTs' version, test description, test version, testing tool, test log and results, etc.) has to remain accessible beyond the F-Interop session for the involved parties.

This user does not have to know the details on all components involved, all interaction is done through the F-Interop GUI website and the local Agent in case the user has local devices on his desk.

NOTE: for this initial version of the architecture (and accompanying implementation), we will focus on instant allocation and provisioning of the resources. Future reservation of resources will be for the next architecture.

6 Envisioned interaction for a contributor and SDK

In D1.1, an F-Interop-Contributor (FI-Contributor) was defined as any entity that provides testing tools or improves existing testing tools in the F-Interop-Platform as well as testbeds and devices that are added to extend the existing available resources.

Adding devices is as simple as described in the previous section where the interaction can be done through the web-based GUI and where a local Agent is needed to run near the devices.

However, for adding or improving tests (e.g. through the F-Interop open calls), the following interaction and workflow is needed:

- User registration and login
- User starts by discovering the available test suites and by selecting the one he/she wants to extend or use as template for a new test.
- The F-Interop platform deploys a full instance of the platform on a testbed (automatization of setup and configuration is done through Ansible Playbooks)
- User adapts/finetunes/debugs the test or defines a new test and can contribute back by giving the (adapted) code and Ansible Playbooks
- the F-Interop operators can verify the automatic deployment and test itself and make it available to others as well

All the information needed so that a contributor can make this kind of contributions to the test suite portfolio of F-Interop, is called the SDK (software development kit), and consists out of the following at this moment:

- this deliverable D1.3
- the documentation of the AMQP messages and single test framework at http://doc.f-interop.eu
- the full automatic deployment of the F-Interop software as described in this section. This will be released end of April 2017 for milestone MS18 (Testbed as a service integration and SDK demonstrated) at month 18.

7 Conclusion

This deliverable is the first of two deliverables in the task 1.3 Architecture and FIRE+ integration design (D1.4 with final architecture is planned for the end of the project). The requirements studied in task 1.1 and 1.2 were used as input. We describe first the different location models (location of devices and users) that we will support in F-interop, followed by the current high level architecture of components in F-Interop and integration with FIRE+ testbeds. After that, we go into more details on the specific interaction between the components. This interaction is based on the AMQP message protocol and we have defined the message formats, functioning as an API when contributors want to add functionality to F-Interop.

We list also the envisioned interaction for an end-user and for a contributor. As such, the deliverable is also meant as part of the software development kit (SDK) towards (open-call) contributors.

8 Annex 1: TBaaS high level API

This section describes the TestBed as a Service API version v1.

8.1 General overview

8.1.1 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.

8.1.2 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 format[1]:

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

8.1.3 Client errors

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

8.2 FInteropTestbedResources object

8.2.1 Main object

The FInteropTestbedResources object 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 of a complete FInteropTestResouces object. Note that the fields are explained below and that not all fields are mandatory in all requests/replies:

```
{
  "@type": "FInteropTestbedResources",
  "id": 1548,
  "@id": "https://tbaas.ilabt.iminds.be/api/FInteropTestbedResources/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",
```

1 https://tools.ietf.org/html/rfc3339

```
    "expire": "2016-11-30T16:00:00Z",
    "owner": "urn:publicid:IDN+f-interop.eu+user+user1",
    "sharedWith": [
      "urn:publicid:IDN+f-interop.eu+user+user2"
    ],
    "status": "READY"
  },
  "environment": {
    "@type": "FInteropEnvironment",
    "version": "v1.0",
    "amqpServerHostname": "amqp.example.com",
    "amqpServerPort": 1234,
    "amqpAuthentication": {
      "@type": "AMQPAuthentication",
      "rootCertificate": "-----BEGIN CERTIFICATE-----\n...",
      "clientCertificate": "-----BEGIN CERTIFICATE-----\n...",
      "clientPrivateKey": "-----BEGIN PRIVATE KEY-----\n..."
    }
  },
  "resources": [
    {
      "id": "urn:publicid:IDN+wall2.ilabt.iminds.be+node+xxxx",
      "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 will contain an FInteropTestbedResources-object (not necessarily all fields will be filled in). This object has the following fields:

Name	**Type**	**Description**
Id	Integer	Unique ID of the FInteropTestbedResources instance. Generated by the TBaaS engine.
Experiment	Experiment	Info on the underlying slice in which the testbed resources are contained
environment	FInteropEnvironment	This contains all the information about the F-Interop environment in which the resources must operate.
resources	FInteropTestbedResource[]	List of resources which are requested for this FInteropTestbedResources-instance
status	String	The current status of the FInteropTestbedResources instance. The possible values are STARTING, READY, FAILED or NONEXISTING

8.2.2 Experiment

Name	**Type**	**Description**
id	Long	Unique ID within the TBaaS engine for this slice
Project	String	Project in which the slice was created

sliceName	String	Name of the slice
sliceUrn	String	URN of the slice
created	Timestamp	Time when the slice was created
expire	Timestamp	Time when the slice will expire
owner	String	URN of the user who owns the slice
sharedWith	String[]	List of URN's of users with who the slice is shared
status	String	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 FInteropTestbedResources, as the latter also takes the installation and configuration of software and firmware on the resources into account.

8.2.3 FInteropEnvironment

Name	**Type**	**Description**
version	String	Unique name which allows the TBaaS engine to identify the version of all F-Interop tools.
amqpServerHostname	String	The hostname of the AMQP-server which hosts the F-Interop Bus
amqpServerPort	String	The port on which the AMQP F-Interop Bus can be accessed
amqpAuthentication	AMQPAuthentication	Authentication information that can be used by the F-Interop agents to connect to the AMQP F-Interop Bus. This is different from the certificate needed to access testbed resources.

8.2.4 AMQPAuthentication

Name	**Type**	**Description**
rootCertificate	String	PEM-encoded X.509 (self-signed) root certificate. This certificate will be used to create certificates for all parties in the F-Interop Testbed
clientCertificate	String	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
clientPrivateKey	String	PEM-encoded private key that must be used together with the client certificate to setup the connection to the AMQP F-Interop Bus

8.2.5 FInteropTestbedResource

Name	Type	Description
id	String	Unique ID of the resource
image	String	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. The image may be referenced by an URN or an URL. A filename is also accepted when the file is uploaded together with the FInteropTestbedResource as part of a Multipart request.

8.3 API calls

8.3.1 Creating FInteropTestbedResources

8.3.1.1 Request

New FInteropTestbedResources can be requested by posting a FInteropTestbedResources-object:

```
POST /interop-testbed
```

The minimal FInteropTestbedResources-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": "FInteropTestbedResources"
}
```

The following fields can be used to customize the request (according the info above):

- FInteropTestbedResources:
  - 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 (the AMQP server will be deployed with the stand-alone setup). 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.

8.3.1.2 Response if FInteropTestbedResources was successfully created

```
Status: 200 OK
```

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

8.3.1.3 Response if request contains an error

```
Status: 400 Bad Request
```

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

8.3.1.4 Response if an error occurred during the FInteropTestbedResources creation

```
Status: 500 Internal Server Error
```

```
{
  "code": 500,
  "message": "The AM could not be reached to create a new slice"
}
```

8.3.2 Fetching current status of FInteropTestbedResources

The current status of FInteropTestbedResources can be requested by performing a GET-request:

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

This request will return a full FInteropTestbedResources-object.

8.3.3 Updating FInteropTestbedResources

FInteropTestbedResources 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.

8.3.4 Destroying FInteropTestbedResources

An InteorpTestbed 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.

8.3.5 SpeaksFor

In order for the TBaaS engine to be able to allocate and provision resources on a federated testbed for a user, the TBaaS engine must have a valid SpeaksFor-certificate of that user. This certificate authorizes the TBaaS engine to speak on the behalf of the user to the testbeds. See also in Annex 3.

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

8.3.5.1 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.

9 Annex 2: TBaaS low level RSpec API

This low-level API is already functional and available at http://jfedapi.ilabt.iminds.be/#/demo (for a web-based functional demo), and at 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

9.1 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.

```
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 favorite browser:

- Login to the iLab.t authority 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.

9.2 Create

For the Create call, the following call is needed:

```
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:
{
  "requestRspec": "<rspec xmlns=\"http://www....",
  "speaksForCredential": "<?xml version=\"1.0\" encoding=\"UTF-8\"… ",
  "project": "f-interop",
  "sliceName": "demo1",
  "expire": "2016-10-22T15:06:34.000Z"
}
```

An RSpec (Resource Specification) lists all resources and their configuration and is defined here https://fed4fire-testbeds.ilabt.iminds.be/asciidoc/rspec.html and some examples can be found here: http://doc.ilabt.iminds.be/ilabt-documentation/urnsrspecs.html#request-rspecs-virtual-wall-physical-hosts .

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

A successful reply will look like this:

```
{
  "id": 75,
  "expire": "2016-10-22T15:06:34Z",
  "created": "2016-10-22T13:13:06Z",
  "speakingFor": "urn:publicid:IDN+wall2.ilabt.iminds.be+user+bvermeul",
  "project": "bvermeul",
  "sliceName": "demo1",
  "owner": "urn:publicid:IDN+wall2.ilabt.iminds.be+user+bvermeul",
  "sliceUrn": "urn:publicid:IDN+wall2.ilabt.iminds.be:bvermeul+slice+demo1",
  "status": "STARTING",
  "@id": "https://jfedapi.ilabt.iminds.be/api/experiment/75",
  "@type": "Experiment"
}
```

An error reply will look like this:

```
{
  "code": 500,
  "message": "There was an error processing your request. It has been logged (ID 
1a252603fa8752eb)."
}
```

9.3 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'?>\n<rspec …",
  "speakingFor": "urn:publicid:IDN+wall2.ilabt.iminds.be+user+bvermeul",
  "project": "bvermeul",
  "sliceName": "demo1",
  "owner": "urn:publicid:IDN+wall2.ilabt.iminds.be+user+bvermeul",
  "sliceUrn": "urn:publicid:IDN+wall2.ilabt.iminds.be:bvermeul+slice+demo1",
  "status": "STARTING",
  "statusDetails": {
    "globalSliverStatus": "UNINITIALISED",
    "jFedExperimentState": "PROVISIONING"
  },
  "@id": "https://jfedapi.ilabt.iminds.be/api/experiment/75",
  "@type": "Experiment"
}
```

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

```
<?xml version='1.0'?>
<rspec xmlns="http://www.geni.net/resources/rspec/3" type="manifest" generated_by="jFed
RSpec Editor" generated="2016-10-22T15:14:28.327+02:00"
xmlns:emulab="http://www.protogeni.net/resources/rspec/ext/emulab/1"
xmlns:jfedBonfire="http://jfed.iminds.be/rspec/ext/jfed-bonfire/1"
xmlns:delay="http://www.protogeni.net/resources/rspec/ext/delay/1" xmlns:jfed-
command="http://jfed.iminds.be/rspec/ext/jfed-command/1"
xmlns:client="http://www.protogeni.net/resources/rspec/ext/client/1" xmlns:jfed-ssh-
keys="http://jfed.iminds.be/rspec/ext/jfed-ssh-keys/1"
xmlns:jfed="http://jfed.iminds.be/rspec/ext/jfed/1"
xmlns:sharedvlan="http://www.protogeni.net/resources/rspec/ext/shared-vlan/1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.geni.net/resources/rspec/3
http://www.geni.net/resources/rspec/3/request.xsd ">
  <node client_id="node0" exclusive="true"
component_manager_id="urn:publicid:IDN+wall2.ilabt.iminds.be+authority+cm"
component_id="urn:publicid:IDN+wall2.ilabt.iminds.be+node+n061-18a"
sliver_id="urn:publicid:IDN+wall2.ilabt.iminds.be+sliver+106075">
    <sliver_type name="raw-pc"/>
    <services>
      <login authentication="ssh-keys" hostname="n061-18a.wall2.ilabt.iminds.be" port="22"
username="bvermeul"/>
    </services>
    <emulab:vnode name="n061-18a" hardware_type="pcgen03-1p"/>
    <host name="node0.demo1.wall2-ilabt-iminds-be.wall2.ilabt.iminds.be"/>
  </node>
</rspec>
```

9.4 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'

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

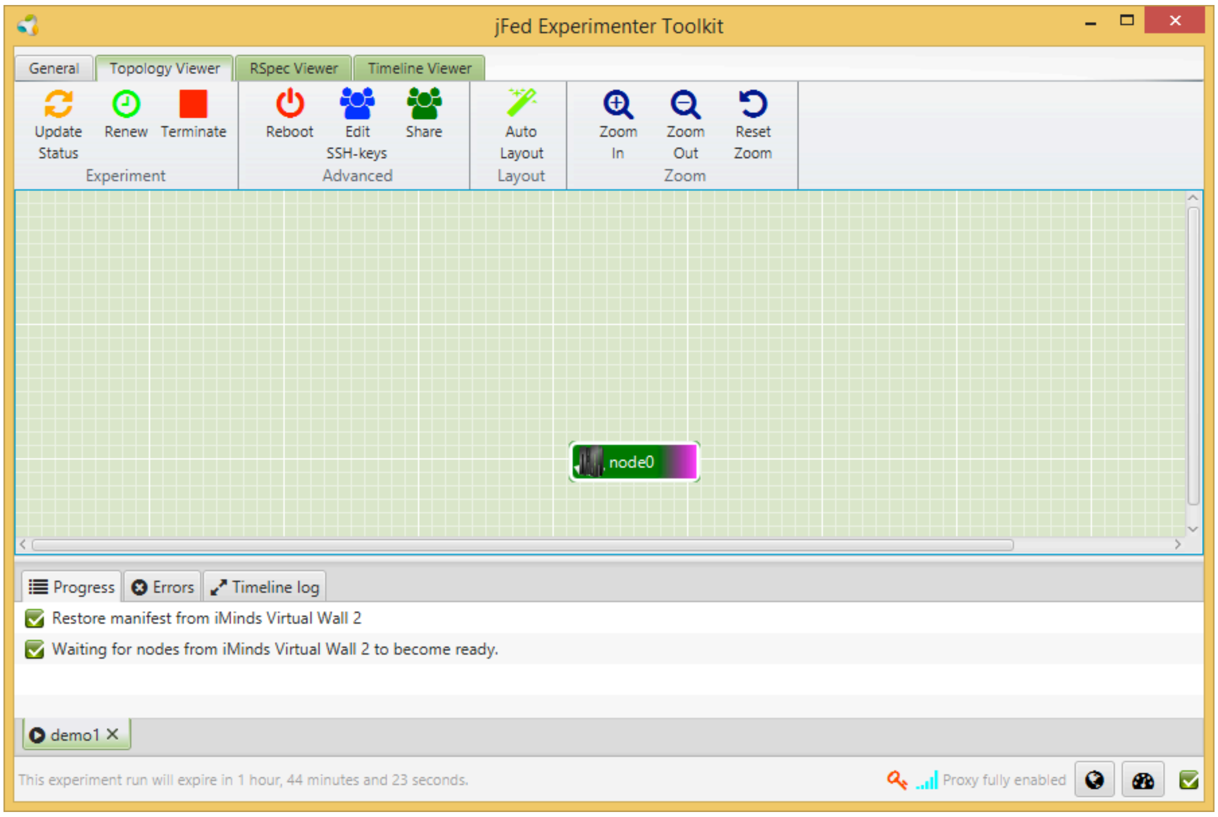
9.5 Delete

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
```

9.6 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.



Restore manifest from iMinds Virtual Wall 2
Waiting for nodes from iMinds Virtual Wall 2 to become ready.

This experiment run will expire in 1 hour, 44 minutes and 23 seconds.

Figure 6: jFed GUI for easy interaction with testbed resources

10 Annex 3: Speaks-for mechanism on testbeds

Figure 7 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.

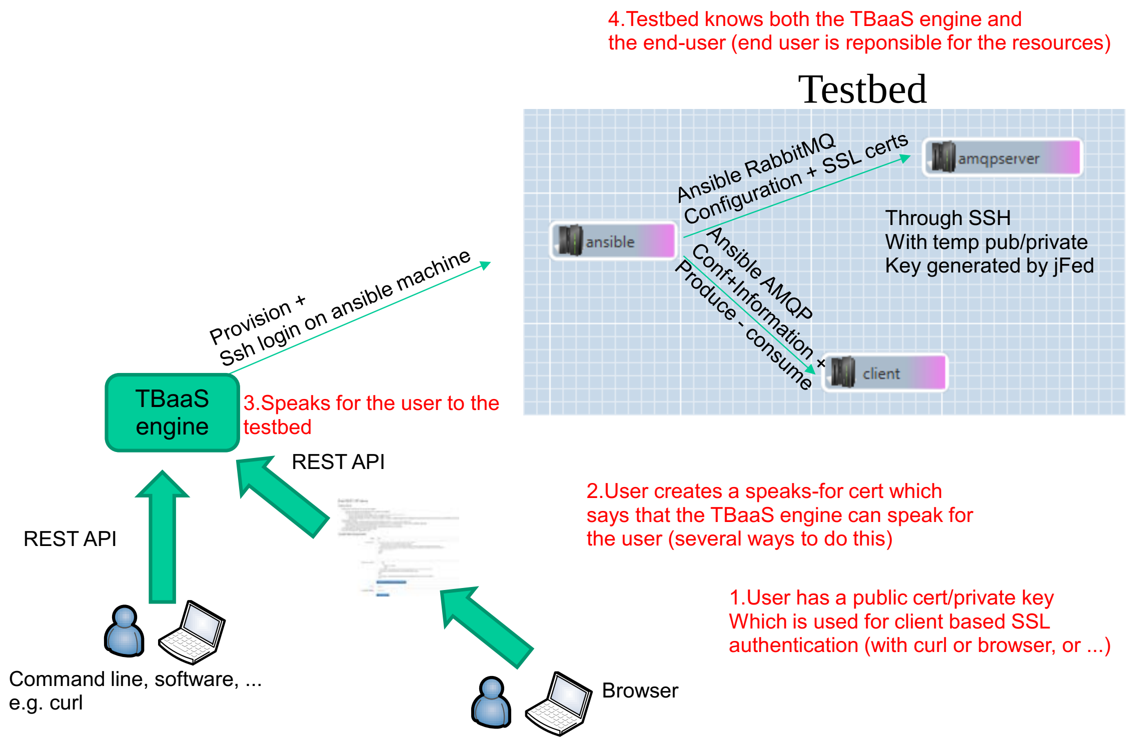

4.Testbed knows both the TBaaS engine and
the end-user (end user is reponsible for the resources)
Testbed
Ansible RabbitMQ Configuration + SSL certs
amqpserver
ansible
Through SSH
With temp pub/private
Key generated by jFed
Ansible AMQP Conf+Information + Produce - consume
client
Provision + Ssh login on ansible machine
TBaaS engine
3.Speaks for the user to the testbed
REST API
REST API
2.User creates a speaks-for cert which
says that the TBaaS engine can speak for
the user (several ways to do this)
1.User has a public cert/private key
Which is used for client based SSL
authentication (with curl or browser, or ...)
Command line, software, ...
e.g. curl
Browser

Figure 7: Speaks-for mechanism