317959

Mobile Opportunistic Traffic Offloading

D4.2 – Protocols for infrastructure offloading control and coordination

SEVENTH FRAMEWORK PROGRAMME

Grant Agreement No.	317959
Project acronym	*MOTO*
Project title	Mobile Opportunistic Traffic Offloading
Advantage	
Deliverable number	D4.2
Deliverable name	Protocols for infrastructure offloading control and coordination
Version	V 1.0
Work package	WP4 – Offloading protocols and algorithms
Lead beneficiary	FON
Authors	Alaitz García, David Valerdi (FON), Filippo Rebecchi, Farid Benbadis (TCS), Raffaele Bruno, Andrea Passarella, Lorenzo Valerio (CNR) Patricia Ortiz, Daniel Alcaraz and Iván Prada (INNO).
Nature	R – Report
Dissemination level	PU – Public
Delivery date	31/10/2014 (M24)

Table of Contents

List of Tables

List of Figures

Acronyms

Table 1: Acronyms

Acronym	Meaning
API	Application Programming Interface
AA	Application API
IA	Infrastructure API
IO	Infrastructure Operator
LTE	Long Term Evolution
MA	MOTO Application
MP	MOTO Platform
MQTT	MQ Telemetry Transport
OMA DM	Open Mobile Alliance Device Management
QoE	Quality of Experience
QoS	Quality of Service
Rest	Representational State Transfer
SOAP	Simple Object Access Protocol
WSDL	Web Service Definition Language

Executive Summary

This document corresponds to the deliverable D4.2 “Protocols for infrastructure offloading control and coordination”, in FP7-MOTO project. The main purpose is to define the protocols to be used for the communications between the MOTO Platform and the external entities (terminals, infrastructure operators and content providers).

The document identifies different pieces of information required to be exchanged among entities by basing on the MOTO architecture and the involved offloading algorithms, which are also described in the deliverable. Once interfaces and information exchange are defined, different protocols are assessed for each of the interfaces. This exercise results in the selection of RESTful for the Infrastructure and the Application API’s and ATOM for the communications with terminals. Security aspects regarding the communications between the MOTO platform and external entities is also covered, as an advancement of task 4.3. This document will serve as a basis for the implementation of the prototype in work package 5. In that sense, an Appendix, which shows the kind of messages that are to be implemented, is included.

1. Introduction

Figure 1 shows the main communication interfaces between Core MOTO Services and other entities within MOTO ecosystem (following the guidelines of the MOTO general architecture [1]): infrastructure operators, applications and terminals (i.e., Terminal MOTO Services). These interfaces are the Infrastructure API, the Application API, and the Control Channel, respectively. This document will cover the specification of these interfaces for terminal-to-terminal communications.

The following figure shows (in a simplified way) MOTO entities, the different communications interfaces between them, and where they are defined across different MOTO tasks. The first two interfaces are conceived as API's that implement a protocol. Thus, both methods and protocol should be defined, while the control channel is a pure protocol approach.

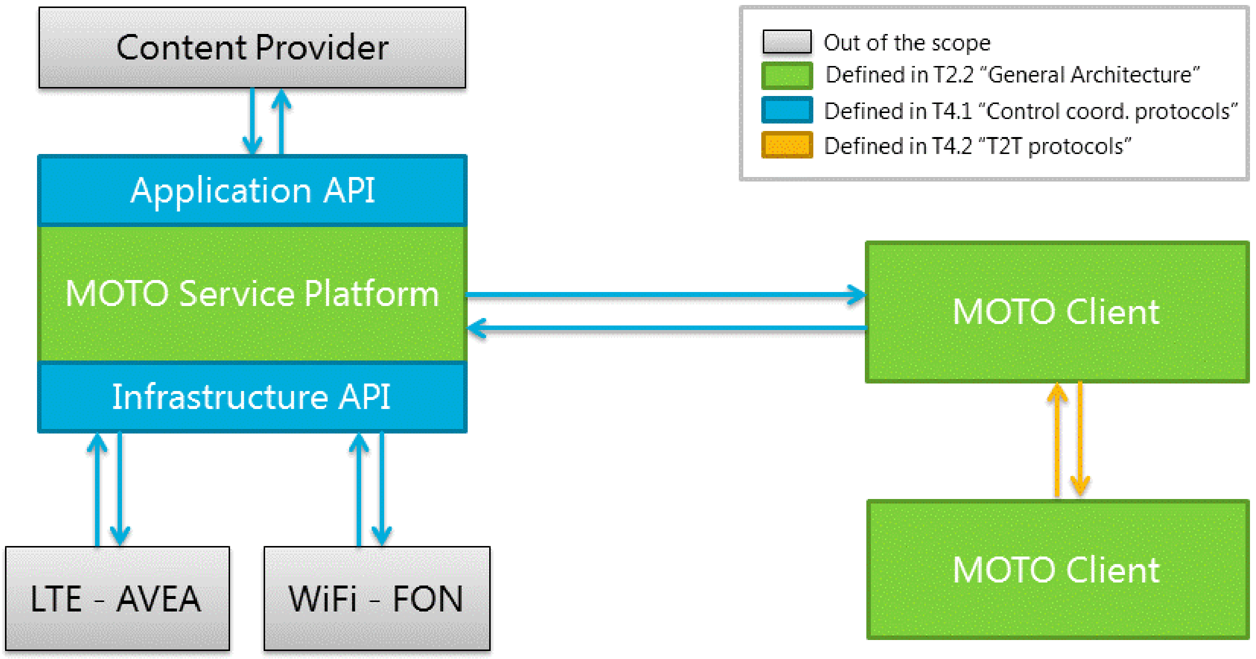


Figure 1. - MOTO specification overview.

The *Infrastructure API* enables agnostic integration of Core MOTO Services with different access networks (e.g., Wi-Fi, LTE). It is mainly used to exchange information regarding user localization and authentication, access network topology, authentication between access networks platforms and Core MOTO services, access network conditions / status, to cite a few. This information is consumed by different blocks within the Core MOTO Services. For instance, LOC component, which deals with information of the localization of MOTO clients, requires user localization and network topology information provided by access networks through the Infrastructure API. Similarly, the content diffusion manager sets dissemination strategy by basing on user localization, network topology, and access network (e.g., load / remaining capacity) status.

The *Application API* allows entities like content service providers to integrate their application and services into a MOTO ecosystem. Any application can request MOTO to deliver content by using the *Application API*, while offloading the network(s) used by application subscribers. Through this API, a content provider delivers the following inputs to Core MOTO Services: content to disseminate, list of users interested or subscribed to content updates, and service constraints to be met (SLA specifying, for example, the delay tolerance).

The *Control Channel* enables control information exchange between the Core MOTO Services and the Terminal MOTO Services. Through this channel, terminals receive dissemination strategy/instructions provided by the Content Diffusion Manager and provide, among others, content tracking feedback, localization information, and exchange authentication messages.

For the sake of completeness, we reproduce the MOTO architecture in Figure 2 [1]. Note that along the entire document we will constantly refer to the components of this architecture.

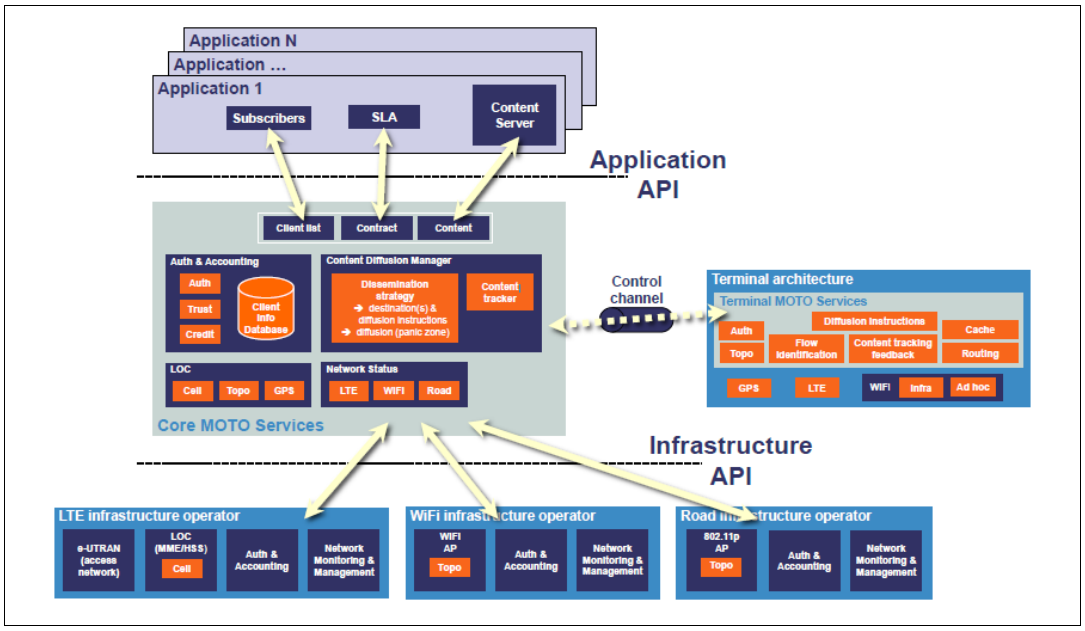


Figure 2. - MOTO Architecture [1]

2. Offloading algorithms: Definition and principles

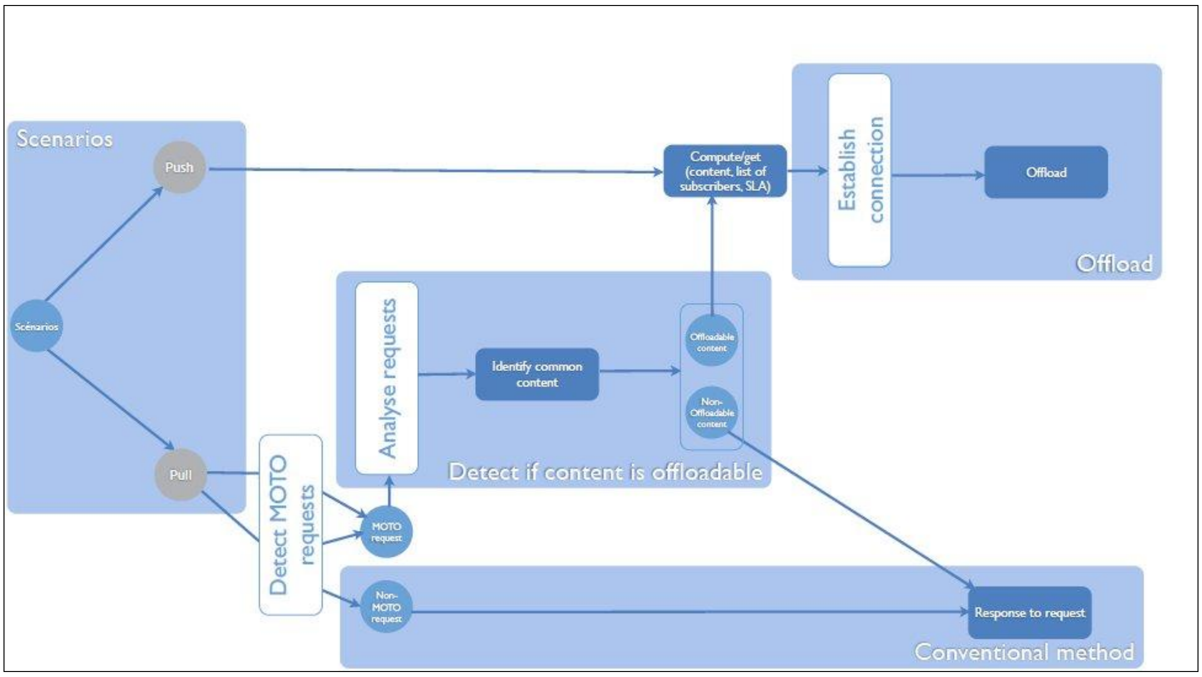


Figure 3: Involved mechanisms depending on push or pull scenario.

2.1 Push/pull Scenario

In Figure 3, we propose a diagram describing the different steps required for the establishment of an offloading scenario. As we can see in Figure 3, we identify two possible offloading schemes: **pull** and **push**. On the one hand, the **push** schema involves a scenario where the request for a given transaction is set up by the publisher (content provider, or MOTO Application Server). The server providing content knows in advance the list of target users, similarly to a Publish/Subscribe mechanism. Thus, the Application Content Server sends the content, the list of recipient users, and their respective SLA to the MOTO service. On the other hand, the **pull** scenario requires a few extra steps in order to identify which content is apt to be offloaded, and which users should be targeted. The request for the transmission of information is initiated by the receiver (MOTO client). The MOTO Content Diffusion Manager implements the chosen offloading mechanism, based on these inputs.

In the case of a pull scenario, the MOTO platform does not know a priori the list of users who have requested the content, which can evolve in time. Thus, the MOTO architecture requires an additional element to allow the identification of similar requests by co-located users. The simplest method is to let the MOTO system be in contact with the external content provider, which is in charge of updating the list of users that requested the content using the Application APIs [1]. The information passed is then confronted with location information, to find offloadable content. Once this step is performed, the same offloading mechanism used in the push scenario is established, with the difference that the content and the list of clients are provided by the request analyzer block. If the request analyzer block does not identify similar content, then this latter is distributed according to the conventional method currently used in cellular networks (e.g., unicast communications). In the remaining part of this section, we provide a high level description of the Content Diffusion Manager represented in Figure 3.

2.2 Offloading algorithm at a glance

In this section, we give a brief overview of the algorithms governing the MOTO offloading schema. The offloading algorithm is implemented into the Content Diffusion Manager (CDM) block represented in Figure 2. - MOTO Architecture [1], and already introduced in the deliverable 3.1 “Initial results on offloading foundations and enablers”[2].

Figure 4 outlines the basic offloading operations, following the Push-and-Track and DROiD approaches described in[3][4] and [5]. Initially, only a small subset of MOTO clients receives the content through the LTE data channel. They start propagating content opportunistically through the ad-hoc interface to other encountered MOTO clients. When a MOTO client receives the content of interest opportunistically from a neighbor, it acknowledges the reception to the MOTO Content Diffusion Manager through a data LTE channel used as control. Control messages always transit through the LTE channel, so to allow the MOTO CDM to monitor the real-time evolution of the content dissemination process, in order also to account for data usage. The MOTO CDM continually estimates the temporal evolution of the content, and may decide to re-inject additional copies in order to boost the diffusion.

Effectiveness of opportunistic communications hinges upon the mobility pattern of MOTO users. In fact, opportunistic networks can only provide probabilistic guarantees of successful content delivery and reception times. To solve this issue, and to give a minimal QoE guarantee to MOTO clients interested in the content, MOTO CDM implements panic zone re-injections. When the elapsed approaches the maximum delivery delay for content, the MOTO CDM enters a panic zone and pushes the content to all uninfected nodes through the LTE data channel, guaranteeing full dissemination and a minimal QoE to all MOTO clients.

Note that panic zone re-injections guarantee a fallback method to overcome various issues that may appear in the opportunistic network, such as node failures or mobile users behaving selfishly – the occurrence of these events could heavily impact the opportunistic diffusion.

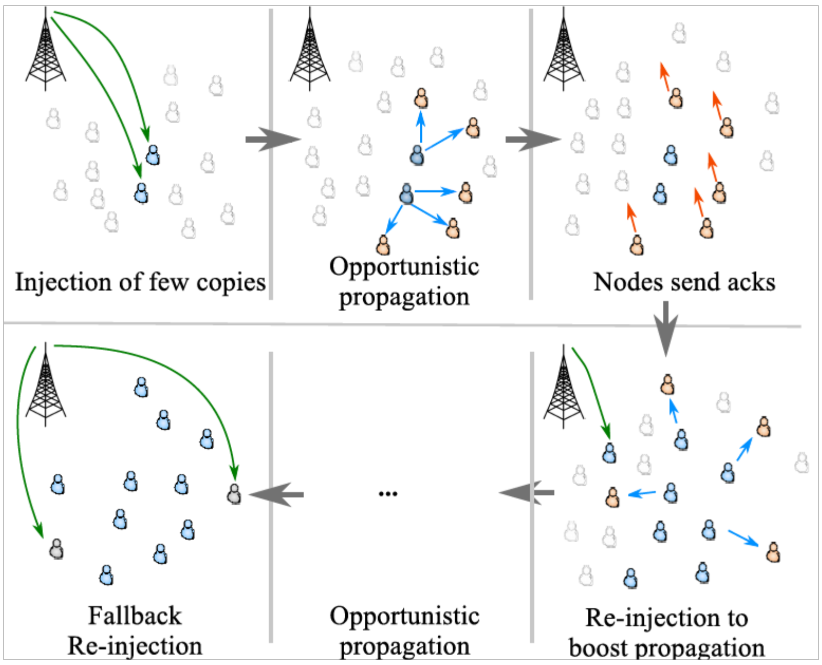

Injection of few copies
Opportunistic propagation
Nodes send acks
Fallback Re-injection
...
Opportunistic propagation
Re-injection to boost propagation

Figure 4: MOTO Content Delivery Manager functioning: Initially a small amount of copies is injected through the LTE data channel to kick start the dissemination process. Content is diffused through opportunistic communications. Upon content reception, users acknowledge the offloading agent using the control channel. MOTO Content Delivery Manager may re-inject additional copies to control the diffusion. 100% delivery ratio is reached through fallback re-injections.

It is important to highlight that effective propagation in such a scheme is strongly linked to a good selection of the first nodes that will start the offloading phase of the communications. Therefore, the offloading algorithm will base the selection of the first offloading seeds nodes on different aspects relevant to their suitability: location, communications history (trust), resources, etc.

2.3 Information required by the MOTO Service from external entities and terminals

As stated above, MOTO service needs to retrieve contextual information related to its clients to drive the offloading process. We will start by briefly explaining the basic information needed at MOTO CDM block to perform offloading. Following sections will deal more on the protocols used to get this information.

First, for any content to be offloaded, the MOTO CDM needs to access the application data at the *Application Content Server*. For each type of content, MOTO CDM needs also the list of MOTO client IDs that requested the data and their SLA. The list of recipients may be nearly static (e.g., a subscription service in the push scenario) or dynamic (e.g., pull scenario). The SLA should include information such as the maximum permitted reception delay (delay-tolerance) in order to guarantee a minimal QoS level.

For any content not yet expired, the MOTO CDM has a *content tracker* that keeps the identifier of clients that have received the content. The *content tracker* is updated each time a MOTO client receives the content; it also serves to monitor in real-time the temporal evolution of content dissemination in the network and represents one of the inputs of the offloading algorithm. It is then straightforward for the CDM block to extract the list of clients that have not yet received content.

Along with information related to the reception of the content, information about the user's behaviour (Trust) is received by the AUTH & ACCOUNTING module. This information will serve the MOTO CDM to fine tune in real-time the selection of the seed nodes in the case of re-injection. In this way, MOTO communications will avoid the selection of erroneous seed nodes that could, through a selfish or malicious behaviour, severely affect the dissemination effectiveness and overload the primary channel with the transmissions of content.

The MOTO service should be able to receive *control messages* from user terminals on the cellular channel. *Control messages* may contain a variety of contextual information regarding the status of MOTO clients. The main pieces of information carried by *control messages* are the content reception status of clients and the trust feedbacks. In this case, *control messages* act like *Ack* messages upon opportunistic content reception. *Ack* messages inform the MOTO CDM that a MOTO client received the content through the alternative ad hoc technology. In addition, they update the information gathered by the AUTH & ACCOUNTING. *Ack* messages **must** contain the identifier of the target MOTO client, the identifier of the received content, and they **should** also enclose the identifier of the MOTO client that delivered content (eventually to allow the operator to apply billing/discounts strategies) as well as the trust values feedback.

Each time an *Ack* message is received, MOTO CDM updates the associated content tracker. The AUTH & ACCOUNT module processes and updates (if necessary) the reputation information of users. Information carried by *Ack* messages is required to allow MOTO CDM block to track the content dissemination, to permit the AUTH & ACCOUNTING module update user's reputation, and to provide recommendations of user's selection to the MOTO CDM. This information is needed by the MOTO CDM to apply the dynamic re-injection strategies proposed by MOTO algorithms presented in [1].

Depending on the selected strategy, a number of complementary information may be necessary at the CDM block in order to implement the dissemination. CDM may retrieve additional information through *control messages* or by the platform APIs; among others, the most important ones are:

- **Subscription time:** The subscription time of MOTO clients could be seen as an indicator of their satisfaction. The MOTO service may need to know what clients have the most recent or oldest subscription time so to prioritize the transmissions. This information is provided by the Content Provider using the Application APIs. In addition, whenever MOTO provides a location-based service, the subscription time could be seen as the entry time in the interest area. Thus, this information can be related to the client's position inside the area of interest.
- **Position:** The MOTO service may decide to target MOTO clients near specific zones with a lot of clients requesting the same data. Pushing data toward clients in such dense zones increases the likelihood of opportunistic encounters between neighbour clients, improving content

dissemination. The LOC component inside the MOTO architecture is in charge of gathering the positioning information through explicit control messages sent by MOTO clients (e.g., using the GPS information available), or querying through the Infrastructure APIs the operator infrastructures. As location information has serious implications regarding personal data, a strong effort shall be made to avoid disclosure of this type of information to third parties. This control messages should therefore be encrypted with a scheme that guarantees no listener (wireless communications are transmitted through an open channel) can take this information.

- **Ad-hoc Topology:** The MOTO service may decide to send data to clients with many neighbors (e.g., clients with high centrality metrics). Thus, the MOTO service may be interested to learn the current topology of the ad hoc overlay connecting the MOTO clients. Similarly to the previous case, the network topology is retrieved through periodic explicit control messages by MOTO clients. It could also be retrieved, with a degree of approximation, from the operator infrastructures through the Infrastructure APIs. Privacy and data integrity concerns apply as well here.
- **WAN access conditions:** The algorithm may want to target specific nodes experiencing good WAN access conditions. In this way, it may help in minimizing the cost of the service for mobile operators. This information may be retrieved from the *network status* block inside MOTO service.
- **User's trust:** The MOTO service will need to filter those nodes that are more likely to provide inefficient offloading rates. Therefore, the recommendations received by the AUTH & ACCOUNTING module related to the suitability of user's to act as seeds in the offloading scheme (especially in the initial phases) will be crucial. The use of this information is the key to make good re-injection decisions and push the correct number of copies. In fact, the MOTO Service should be able to decide to whom inject the content through e.g. the LTE data channel. Choosing high potential clients with good historic trust reputation may help in maximizing the effectiveness of the overall offloading strategy, minimizing at the same time the LTE data use.

Table 2: Required MOTO Service information.

INFORMATION	RETRIEVED FROM	DESCRIPTION
Subscription Time	Application APIs	Date at which the UE requested the content
UE position	Control messages / Infrastructure APIs	Geographic position of UEs (GPS / Cell)
Topology	Control messages	Neighborhood of users
Channel Conditions	Infrastructure APIs	Channel conditions of UE
Trust	Control messages / Infrastructure APIs	Recommendation for UE's trust and reputation

2.4 Dissemination Strategies

For each piece of content to be delivered, the MOTO Content Diffusion Manager needs to track the remaining time to the delivery delay and the instantaneous dissemination level. The delivery delay information can be extracted from the SLA. On the other hand, the instantaneous dissemination level and other useful information can be inferred through the received acknowledgements. The employed offloading algorithm should know which dissemination strategy to employ. Examples of dissemination strategies from the literature are:

- Fixed objective functions [4]: The current propagation ratio is compared to an objective function, which indicates for each instant what the current propagation ratio should be. If, at any time, the measured propagation ratio is below the target propagation ratio, the strategy returns the minimum number of additional copies that need to be re-injected to meet the target.
- Derivative based [3]: The trend (derivative) of the propagation ratio is evaluated and compared with a dynamic threshold. If the propagation ratio does not evolve quickly enough (the propagation ratio falls into a plateau), copies of the content are re-injected.
- Learning method [5]: The system is able to automatically learn the best subset of nodes to which delegate the dissemination. The strategy concurs to reduce the traffic on the cellular network by sensing the infection level and the dissemination potential of each node.

3. Communications with external infrastructure operators

This section covers the communications between the MOTO Platform and the external infrastructure operators such as LTE or Wi-Fi operators. First of all, the interfaces for which the protocols must be defined are listed. Later, the different protocol alternatives are analysed to finally select the most appropriate one for each interface.

3.1 Interface definition

Before analysing the possible protocols to be implemented for the communications between the MOTO platform and the external infrastructure operators, the communication interfaces must be defined.

In order to define the methods for the Infrastructure APIs as well as the protocols that are going to be used, the modules that should receive/send information from/to the external infrastructure operators and the information that must be exchanged between them needs to be defined first.

To accomplish this, the use case flows described in[3]can be taken as a reference. In those flows, the communications between the different modules of the MOTO Platform and the external infrastructure operators were described.

The communication needs and the information exchange between the external infrastructure operators and the MOTO Platform are:

- **Authentication of the Infrastructure Operator in the MOTO Platform**

  Prior to executing any dissemination strategy, the network that is to offer the MOTO service needs to be authenticated against the MOTO platform. That way, the MOTO platform ensures that the network it communicates with is a legitimate one and attacks could be avoided from untrusted parties trying to avoid the delivery of MOTO services, sending to the MOTO platform false information about the network status of operators.

  The system in charge of this function is located within the infrastructure operator core and it performs the authentication when the rest of the infrastructure operator core services are enabled for MOTO operations.

- **Authentication of the MOTO platform in front of the Infrastructures operator**

  In the same way that authentication is required by the MOTO platform in order to trust the network status information from the infrastructure operator, it is necessary to assure the confidence from the operator’s side as well. Therefore, the MOTO platform needs to authenticate against the infrastructures that are connected with. This way, operators will assure that the access to their user’s personal information (e.g. location), their network status, etc. is granted only for the MOTO platform.

  This authentication process will be performed by the AUTH & ACCOUNTING module of the MOTO platform, when the MOTO platform makes use of a certain operator’s infrastructure or requires update of information regarding users (e.g. location).

- **Informing the MOTO Platform on the topological information of the infrastructure operator network**

  After the infrastructure operator and the MOTO platform have authenticated one another, the infrastructure operator provides MOTO with the topological information of the network for the areas where the MOTO service is going to be enabled, such as malls, stadiums, museums, etc.

This topological information can be provided to the MOTO Platform once authentication of both of them has taken place, or it could be the MOTO Platform the responsible for asking to the Infrastructure Operator Core Network for clients' location each time they connect to a different AP/Cell.

- **Updating the network's status in the MOTO Platform**

  The infrastructure operator needs to update the status of the network overall and other performance related parameters to the MOTO core in order to provide the latter with the needed information to perform the expected dissemination.

  For example, the infrastructure operator can provide MOTO Platform with the information of the Aps/Base Stations that are out of order, so that MOTO knows about the issue and takes it into account in the dissemination process.

Depending on the final location of the MOTO platform, internal or external to the operator's infrastructure, information delivered between them might travel to a potentially insecure channel. In this sense, and taking into account the sensitiveness of the information exchanged, strong security must be in place.

After analysing the communication needs between the two entities, the messages that must be exchanged between them are set out in the following table:

Table 3: Communication messages between Infrastructure Operator and Infrastructure API.

DIRECTION	MSG TYPE	MSG NAME	MSG FIELDS	DESCRIPTION
IO → IA	CONNECT	CONNECTION_REQUEST		IO SEND MOTO PLATFORM A CONNECTION REQUEST
IA → IO	AUTH	AUTHENTICATION_REQUEST	UID	MOTO PLATFORM SEND AN AUTHENTICATION REQUEST TO IO
IO →IA	AUTH	AUTHENTICATION_RESPONS E	UID	IO SENDS CREDENTIALS TO IA FOR AUTHENTICATION
IA →IO	AUTH	AUTHENTICATION_ID_RESP ONSE	UID, ALLOW/DE NY	IA SENDS A CONFIRMATION TO IO GRANTING OR NOT THE ACCESS TO CONFIGURE THE MOTO SERVICES RELATED TO OPERATOR PARAMETERS
IO→ IA	AUTH	AUTHENTICATION_REQUEST	UID	AFTER BEING ACCEPTED BY THE MOTO PLATFORM, IO SENDS AN AUTHENTICATION REQUEST TO IA TO CONFIRM ITS IDENTITY
IA → IO	AUTH	AUTHENTICATION RESPONSE	UID	MOTO PLATFORM SENDS CERTIFICATE TO IO
IO →IA	CTRL	UPDATE_TOPO_INFO	UID, POSITION	IO SENDS TOPOLOGICAL INFORMATION TO IA WITHOUT BEING PREVIOUSLY REQUESTED BY THE IA
IA→IO	CTRL	REQUEST_TOPO_INFO	UID,	IA REQUESTS TOPOLOGICAL

			POSITION	INFORMATION TO IO
IO→IA	CTRL	UPDATE_NETWORK_STATUS	UID, APID, CELLID STATUS	IO SENDS NETWORK STATUS TO IA WITHOUT BEING PREVIOUSLY REQUESTED BY THE IA. IT CAN BE THE STATUS OF A CERTAIN AP OR THE ENTIRE NETWORK.
IA→IO	CTRL	REQUEST_NETWORK_STATU S	UID, APID, CELLID STATUS	IA REQUESTS NETWORK STATUS TO IO OR A CERTAIN AP STATUS.

Legend:

IO = Infrastructure Operator	AUTH = Authorization	UID = User ID
IA = Infrastructure APIs	CTRL = Control	APID = Access Point ID
Cell ID = BaseStation ID		

3.2 Protocol analysis

After analysing the messages that must be exchanged between the MOTO Platform and the Infrastructure Operator, the Infrastructure API has to be defined. To accomplish this, first, the different alternatives for the protocols that would be valid for that API have to be analysed. Thus, this section covers the analysis on what type of protocols would be valid for the defined message exchange.

There are two main options for the protocol to use for the Infrastructure API, to either define a proprietary protocol from the scratch or to employ the ones that are used to create web based APIs. In the following sections, each of the options is covered and the benefits and drawbacks presented as well.

3.2.1 Proprietary protocol

By defining a proprietary protocol it is possible to achieve a complete customisation level that would allow tailoring the protocol to MOTO's needs. That way, it could be possible to optimise the protocol for the type of communications made with the MOTO platform. However it involves a major effort on defining and developing it.

3.2.2 Web API technologies

Web APIs are programmatic interfaces to define request-response message systems that are made available in a network through a web related protocol. These APIs allow interacting with a service that is available in a server, and since is a web related service it is known as web service.

Web services, in general, allow different type of applications to consume their resources. For example, the clients can be web pages, stand alone applications, mobile applications etc.

Each web service describes the API that it serves through a file, in a machine processable fashion. The language used to describe the API is Web Services Description Language (WSDL), and the file is usually known as WDSL file. This language is defined by the W3C and the current specification is the 2.0 one[3].

The most commonly used approaches for web APIs are SOAP, and more recently, RESTful APIs. Both define a way to interact with a server side API and they can be secured by using an HTTPS scheme.

In the following sections, both the SOAP and the RESTful approaches are covered separately and then a side by side comparison is made.

3.2.2.1 SOAP

SOAP, which stands for Short for Simple Object Access Protocol, is an XML-based messaging protocol used to encode the information in Web service request and response messages before sending them over a network. The protocol is defined and standardised by the W3C.

SOAP messages are independent of any particular operating system or programming language so theoretically the clients and servers in these dialogues can be running on any platform and written in any language as long as they can formulate and understand SOAP messages. Therefore, the SOAP messages can be transported using a variety of Internet protocols, including SMTP, MIME, and HTTP. Nevertheless, since in this section the focus is made on web APIs only the latter one is going to be considered.

Basically, the protocol defines a set of XML envelopes that contain different fields to be processed in the receiver node. In the case of web APIs, the fields reference a Remote Procedure Call (RPCs) to be executed by the server, and the parameters to be used in each case. The server's response is also SOAP formatted and it indicates the result of the operation.

Additionally, SOAP uses the same error and status codes as those used in HTTP so that HTTP responses can be directly intercepted by a SOAP module.

3.2.2.2 RESTful

REST stands for Representational State Transfer. It relies on a stateless, client-server, cacheable communications. The HTTP protocol is an example of a REST solution where the state of the server side application is modified by issuing different requests. That is why, when referring to REST solutions, in almost all the cases the HTTP protocol is considered[7].

RESTful web APIs employ mainly four of HTTP's request methods (namely GET, POST, PUT, and DELETE) to define different API functions on top. Simplifying the behaviour, the following can be expected from each of the methods:

POST: Create objects.

GET: Retrieve objects.

PUT: Replace objects.

DELETE: Delete objects.

When defining a RESTful web API, a URI is defined for each of the API functions and the URI provided in the HTTP method is used to distinguish the API function that is to be called. Parameters for each of the functions are also sent along.

Unlike SOAP, RESTful web services do not have necessarily need to use XML to provide the response and other formatting options are available, such as: Command Separated Value (CSV), JavaScript Object Notation (JSON) and Really Simple Syndication (RSS).

3.3 Protocol selection

After analysing the different available protocols to implement the infrastructure API one of them must be selected.

The proprietary protocol alternative requires a huge effort on protocol definition and development, despite being a good approach since it allows to totally adapting the protocol to MOTO needs. However, the web based API alternative is much simpler because the protocols are already specified and only their implementation needs to be defined within the MOTO context.

As it has been analysed on the previous section, two different approaches have been taken into account for the web API. In this section, both of them are going to be compared in order to select the most appropriate one for the Infrastructure API.

Table 4 Comparison between SOAP and RESTful.

SOAP	RESTful
Language, platform, and transport independent (REST requires use of HTTP)	No expensive tools require to interact with the Web service
Works well in distributed enterprise environments (REST assumes direct point-to-point communication)	Smaller learning curve
Standardised	Efficient (SOAP uses XML for all messages, REST can use smaller message formats)
Provides significant pre-build extensibility in the form of the web service standards	Faster (no extensive processing required)
Built-in error handling	Closer to other Web technologies in design philosophy
Automation when used with certain language products	

To sum up, SOAP is definitely the heavyweight choice for Web service access, while the Restful approach is easier to use for the most part and is more flexible.

Due to the amount of data that the Infrastructure API will manage, among others, the best option would be developing the Web Service by using **RESTful** approach.

4. Communications with terminals

This section covers the communications between the MOTO Platform and the end users' terminals. First of all, the interfaces for which the protocols must be defined are listed. Later, the different protocol alternatives are analysed to finally select the most appropriate one for each interface.

4.1 Interface definition

Table 5 Communication messages between Infrastructure Operator and terminals.

DIRECTION	MSG TYPE	MSG NAME	MSG FIELDS	DESCRIPTION
MA→MP	CONNECT	CONNECTION_REQUEST		MA SEND A CONNECTION REQUEST TO MP IN ORDER TO ACCESS TO MOTO SERVICES
MP → MA	AUTH	AUTHENTICATION_REQUEST	UID	MP SEND AN AUTHENTICATION REQUEST TO MA
MA →MP	AUTH	AUTHENTICATION_RESPONSE	UID	MA SENDS CREDENTIALS TO MP FOR AUTHENTICATION
MP →MA	AUTH	AUTHENTICATION_ID_RESPONSE	UID, ALLOW/DENY	MP SENDS A CONFIRMATION TO MA GRANTING OR NOT THE ACCESS TO MOTO SERVICES
MP→MA	AUTH	CERTIFICATE_INFO_REQUEST	UID1, UID2	MA1/MA2 REQUESTS TO MP INFORMATION ABOUT THE MOTO CERTIFICATE OF MA1/MA2
MP →MA	AUTH	CERTIFICATE_INFO_RESPONSE	UID1, UID2, YES/NO	MP COMMUNICATES TO MA1 IF MA2 CERTIFICATE IS VALID AMONG OTHER INFO
MA →MP	CONT	CONTENT_REQUEST	UID, CONTENT QUERY	MA MAKES A REQUEST FOR A CONTENT TO MP
MA→ MP	CONT	CONTENT_RECEIVED	UID, CONTENT_ID, OTHER USEFUL INFO	MA COMMUNICATES TO MP THAT THE CONTENT REQUESTED HAS BEEN RECEIVED. THE MESSAGE CAN CONTAIN OTHER ADDITIONAL INFORMATION ABOUT THE OPPORTUNISTIC CONTENT DELIVERY PROCESS
MP →MA	CONT	CONTENT_SEND	UID, CONTENT_ID, PAYLOAD	MP SENDS A CONTENT TO MA

MP→MA	CONT	NOTIFY_NEW_CONTENT	UID, CONTENT_INFO	MP NOTIFIES TO MAS THE EXISTENCE OF A NEW INTERESTING CONTENT
MP→MA	CTRL	ASSIGN_ROLE	UID, SEEDCL/OPPCL, WIFI/LTE	MP ASSIGNS TO MA A ROLE AND, TO WHICH NETWORK INFRASTRUCTURE MA MUST CONNECT
MA →MP	CTRL	UPDATE_POSITION	UID, POSITION	MA SENDS ITS LOCATION INFORMATION TO MP
MA→MP	CTRL	TERMINAL_STATUS_UPDATE	UID, BATTERY LEVEL;DELAY;AVG ACTIVE USERS	MA SENDS ITS UPDATED STATUS INFORMATION TO MP
MP->MA	CTRL	TERMINAL_PARAM_UPDATE	UID, DUTY CYCLE, ETC.	MP SETS/UPDATE MA'S PARAMETERS.
MP→MA	CTRL	SET_T2T_COMM_PARS	UID1, UID2, COMM_PARS	MP SET MA'S COMMUNICATION PARAMS INCLUDING THE INDICATION OF THE FORWARDING/DISSEMINATION ALGORITHM TO USE.

Legend:

MA = Moto Application	AUTH = Authorization	CONT = Content
MP = Moto Platform	CTRL = Control	

4.2 Protocol analysis

There exist three main possible approaches to follow in order to define the APIs allowing the communication between the MOTO Platform and MOTO Terminals. Namely, possible options are a proprietary solution, an http-based RESTful protocol and a non-http based protocol.

4.2.1 Proprietary Protocol

The definition of a proprietary protocol permits to design a complete customized solution. A complete customized solution allows a deep optimization of the protocol. However, it involves a great effort related to its definition and development.

4.2.2 HTTP Based

4.2.2.1 OMA-DM 2.0

OMA DM 2.0 [8] is a standard protocol defined by the Open Mobile Alliance. It is designed for the management of mobile devices through a two-way communication between server and client.

It provides the interface between the DM Server and the DM Client to manage the device. OMA DM 2.0 leverages a RESTful architecture for scalability and management performance, and is also designed to work efficiently on less capable devices.

The OMA DM specifications define the protocols and the mechanisms allowing an OMA DM Server to deliver configuration parameters to an OMA DM Client, by using a defined set of “DM Commands” for various management procedures to be executed inside a well-defined and secure environment (the “DM Session”).

The OMA DM Client exposes the device internal data to the OMA DM Server in the form of a hierarchic tree known as the “DM Tree”: it is made up of different building blocks (or sub-trees) called Management Objects providing specific functionality in the management of devices. In other words, the management of a device feature consists of the management of the DM Tree, which virtualizes the device features and functionalities.

Through OMA Device Management, a Management Authority (in our case a Cellular Operator) can remotely set parameters, conduct troubleshooting servicing of terminals, install or upgrade software among others.

By using the interfaces specified by the OMA DM Client Framework API, applications running on the device can access the DM Tree and interact with Management Objects and the DM Server, in order to obtain configurations or report data.

As for all the RESTful architecture, each node is addressed by a unique full device URI.

It uses an XML-based package format similar to SyncML Synchronization Protocol and SyncML Representation Protocol. Recent improvements allow other data format as JSON, XML, etc.

Summarizing, with OMA DM, any centralized management entity can do the following operation on mobile devices:

- Configure connectivity
- Update firmware
- Diagnose problems
- Monitor performance
- Install and update software
- Lock and wipe personal data
- Manage capabilities
- Schedule and automate device management tasks

OMA- DM Features:

- Network independence—OMA DM works on GSM, GPRS, WCDMA, 3G Mobile and All-IP networks.
- Built on the efficient full-duplex provisioning method, OMA DM uses TCP-IP which easily extends from mobile networks to other networking environments.
- OMA DM secures all transactions using HTTP with TLS 1.0 / SSL 3.0.
- OMA DM does not assume that a device is attached to a cellular network. The only technical requirement is that a client must communicate with a server—via HTTP, OBEX or WSP.
- Notification can be done via SMS, UDP Push, SIP Push, and HTTP Push. Other methods of communication can be created as well.

4.2.2.2 Atom publishing protocols

The Atom Publishing Protocol (RFC 5023) [9] is an application-level protocol for publishing and editing web resources. The protocol is based on HTTP transfer of Atom-formatted representations. The Atom format is documented in the Atom Syndication Format (RFC 4287)[16]. The Atom Syndication Format is a core Internet standard and uses XML to represent online content in the form of a "feed" and "entries."

To publish or edit content, a representation of the content that is to be manipulated is transmitted to an Atom Server over HTTP. The Atom Publishing Protocol makes use of several features supported by the underlying HTTP protocol, some of which are rarely used by web browsers.

It uses Atom-formatted representations to describe the state and metadata of those Resources. It defines how Collections of Resources can be organized, and it specifies formats to support their discovery, grouping and categorization.

The Atom Publishing Protocol uses HTTP methods to author Member Resources as follows:

- GET is used to retrieve a representation of a known Resource.
- POST is used to create a new, dynamically named, Resource. When the client submits non-Atom-Entry representations to a Collection for creation, two Resources are always created -- a Media Entry for the requested Resource, and a Media Link Entry for metadata about the Resource that will appear in the Collection.
- PUT is used to edit a known Resource. It is not used for Resource creation.
- DELETE is used to remove a known Resource.

The Atom Protocol only covers the creation, edition, and deletion of Entry and Media Resources. Other Resources could be created, edited and deleted as the result of manipulating a Collection

The Atom Protocol uses the response status codes defined in HTTP to indicate the success or failure of an operation.

As for OMA DM, security of transaction can be provided using HTTP with TLS 1.0 / SSL 3.0.

4.2.3 Non-Http based

4.2.3.1 MQTT Protocol

MQ Telemetry Transport (MQTT) [16] is a lightweight protocol designed to be open, simple, lightweight and easy to implement. MQTT has a client/server model, where every sensor is a client and connects to a server, known as a broker, over TCP. It is message oriented. Every message is a discrete chunk of data, opaque to the broker.

Every message is published to an address, known as a topic. Clients may subscribe to multiple topics. Every client subscribed to a topic receives every message published to the topic.

MQTT represents a valid and lightweight alternative with regard to a RESTFUL HTTP-based approach.

The MQTT protocol has the following features:

- It is based on the publish/subscribe message pattern that provides an easy one-to-many message distribution and, moreover, allows the decoupling of applications.
- It is completely agnostic to the content of the payload. Thus existing solutions like XML, JSON, etc. can be freely adopted.
- It uses TCP/IP to provide basic network connectivity
- It provide three levels of QoS for message delivery :
    - "At most once", where messages are delivered according to the best efforts of the underlying TCP/IP network. Message loss or duplication can occur.
    - "At least once", where messages are assured to arrive but duplicates may occur.

    - "Exactly once", where message are assured to arrive exactly once.
- It has a small fixed-length header (2 bytes). This permits a small transport overhead.
- It supports WebSocket.
- Security is provided through SSL/TSL on top of the TCP stream and, moreover, Pre-shared key encryption is supported.

4.3 Protocol selection

All the considered protocols have characteristics that let them eligible to be used for the communication between the Moto Platform and Moto Terminals.

OMA-DM has the benefits of being network independent and transport protocol independent, i.e. defines standard bindings over several transports, but additional protocols can be also integrated in a proprietary way. At first sight it could be a good choice, but for our purposes it results quite heavy, moreover its implementation is not so easy.

MQTT instead is far more lightweight, less resource consuming and easy to implement than OMA-DM. However, being not based on HTTP, in our case, could result as a disadvantage in the long term. In fact, choosing an HTTP based protocol let us possibly substitute the communication protocol more easily whether the necessity would arise.

Finally, we think the Atom Protocol is flexible, extensible and easy to use enough for our purposes. Thus we consider ATOM as a good candidate to manage the communication between the MOTO Platform and MOTO Terminals.

5. Communications with external content providers

This section covers the communications between the MOTO Platform and the external content providers.

No protocol analysis is done for this section since the communications with external content providers are done through the application API that is analogue to the infrastructure API. Thus, a Restful approach would be used.

Before offloading any kind of data, the MOTO Platform must be provided with the content to be offloaded the list of MOTO client IDs that require/requested the data, and their SLA. The recipients list may be nearly static, (e.g., a subscription service in the push scenario), or dynamic, (e.g., pull scenario). The SLA should include information such as the maximum permitted reception delay -or delay-tolerance- in order to guarantee a minimal QoS level, and drive content re-injection.

After analysing the communication needs between the two entities, the messages that must be exchanged between them are set out in Table 5.

Table 6: Communication messages between Infrastructure Operator and Content Provider.

DIRECTION	MSG TYPE	MSG NAME	MSG FIELDS	DESCRIPTION
CP →AA	AUTH	AUTHENTICATION_REQUEST	UID	CP SENDS CREDENTIALS TO AA FOR AUTHENTICATION
AA →CP	AUTH	AUTHENTICATION_REPLY	UID, ALLOW/DENY	AA SENDS A CONFIRMATION TO CP GRANTING OR NOT THE ACCESS TO CONFIGURE THE MOTO SERVICES RELATED TO APPLICATION CONTENT PARAMETERS
CP →AA	CONT	SEND_CONTENT_TO_OFFLOAD	CONTENT	CP SENDS THE CONTENT TO BE OFFLOADED TO AA
AA→ CP	CONT	CONTENT_RECEIVED	ACK	AA COMMUNICATES TO CP THAT THE CONTENT HAS BEEN RECEIVED.
CP→AA	CTRL	SEND_CLIENTS_LIST	CLIENT_UID_LIST	CP SENDS THE LIST OF THE MOTO CLIENT IDS THAT REQUIRE/REQUEST THE DATA TO THE AA.
CP→AA	CTRL	SEND_SLA	SLA	CP SENDS THE SLA TO THE AA

Legend:

CP = Content Provider	AUTH = Authorization	CONT = Content
AA = Application API	CTRL = Control	

6. Security aspects

In this section, the security considerations for the protocols for infrastructure offloading control and coordination will be exposed.

First of all, the security implications that must be considered in the dissemination strategy are covered. After this, the main security aspects involved in the different communications with external and internal entities of the proposed MOTO environment are analysed.

The objective of this section is to introduce the reader to the underlying security approach of MOTO.

6.1 Offloading algorithms security considerations

After reading section 2 of the document, it is pretty clear that the propagation scheme for the offloading communications proposed by MOTO is based on a multiplying factor. The proposed propagation is strongly dependent on the efficiency of the first selected MOTO peers that will act as seeds in the Ad Hoc network.

In Figure 5, a hypothetical exposure of the effects of selecting an erroneous node as one of the firsts offloading seeds is showed.

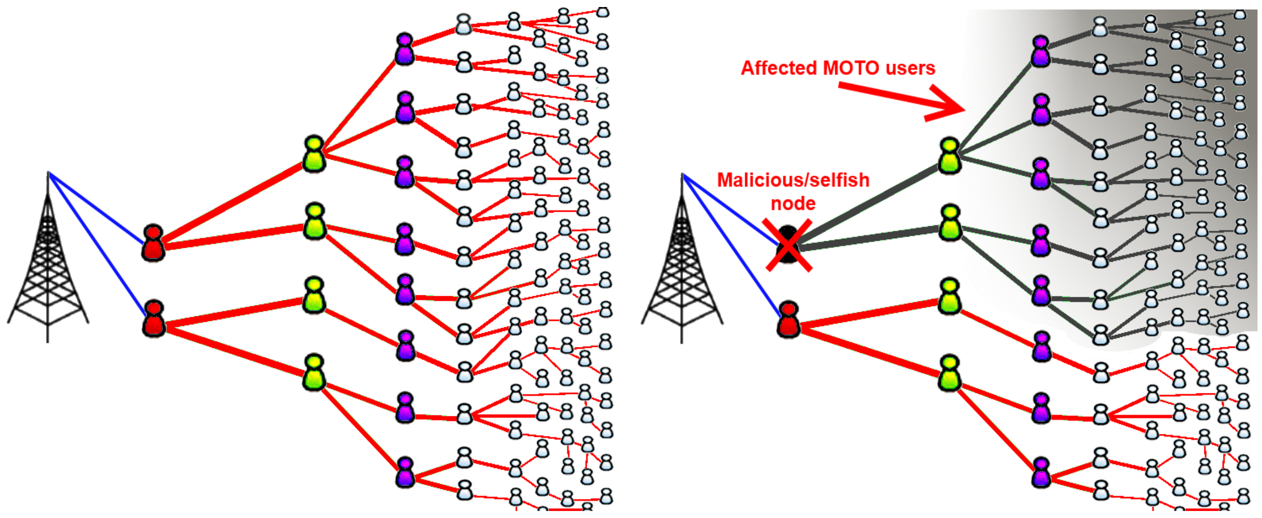


Figure 5. Effect of a bad seed selection node in the initial phases of propagation

The grey shadow indicates the portion of the Ad Hoc nodes that could be affected as a result of the presence of a selfish or malicious node in the initial propagation phase. In this assumption, the two red colored MOTO users receive the content directly through the primary channel and are the responsible to re-transmit it to the following nodes. If one of these initial nodes denies the re-transmission of the content, the effectiveness of the propagation strategy is severely affected.

One of the possible solutions to this eventuality is the reduction of the frequency in which the propagation is assessed by the MOTO platform. In this sense, if the analysis of the propagation is very frequent or even continuous, the identification of failure paths within the Ad Hoc network permits the re-injection of the content to other nodes and the consecution of the delivery attending to the parameters stated.

However, this solution although being effective, as all nodes receive the content within the defined delay (QoS), is not efficient. In an extreme case, where the majority of the selected seeds are selfish or malicious, it does not only overload the operator's network as when using exclusively the operator's infrastructure, but also require high resource consumption by both, the MOTO platform and from the user devices.

Therefore, it can be assumed that in that hypothetical extreme case, this solution could be worse than exclusively delivering the content through the operator's infrastructure.

The most efficient solution to reduce the effect of both selfish and malicious nodes in the propagation is to discard them as possible seeds, especially in the initial phases, condemning them to marginal reception of content in the offloading scheme if they are still acceptable to use MOTO services.

To do so, the MOTO CDM module queries the AUTH module for seed selection recommendations. A possible option for performing this is that the CDM delivers the client list to the AUTH module and that the AUTH module sends this list back either:

- rearranged in relation with their suitability to become a seed (1) - Figure 6
- simply adds information about their suitability in the same list received (2) - Figure 7
- discards users from this list to become seeds based on their trust profile (3) - Figure 8
- A combination of the previous ones (4) Figure 9.

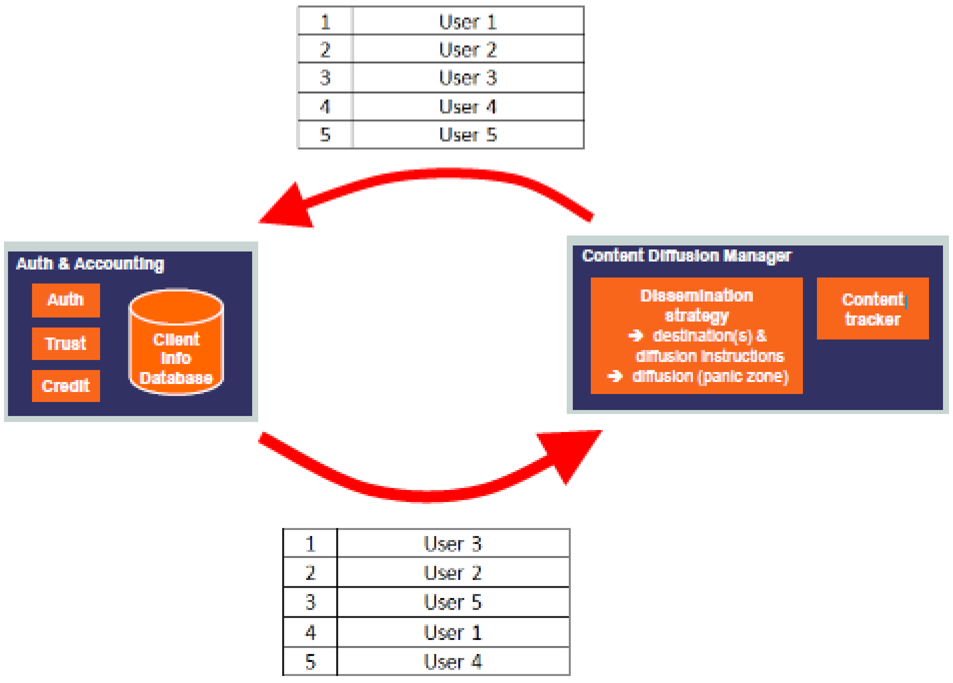
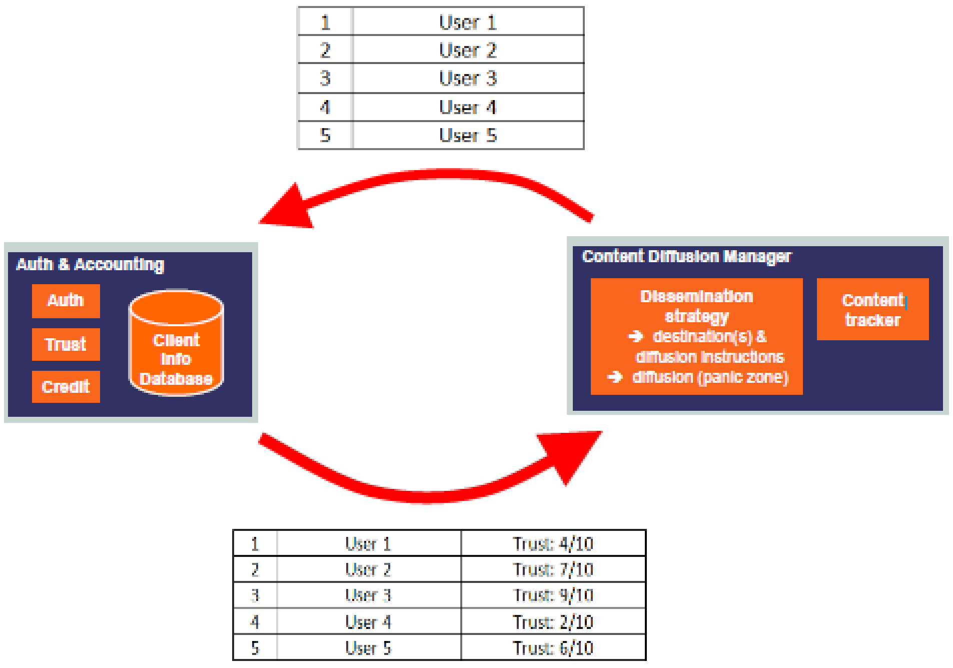
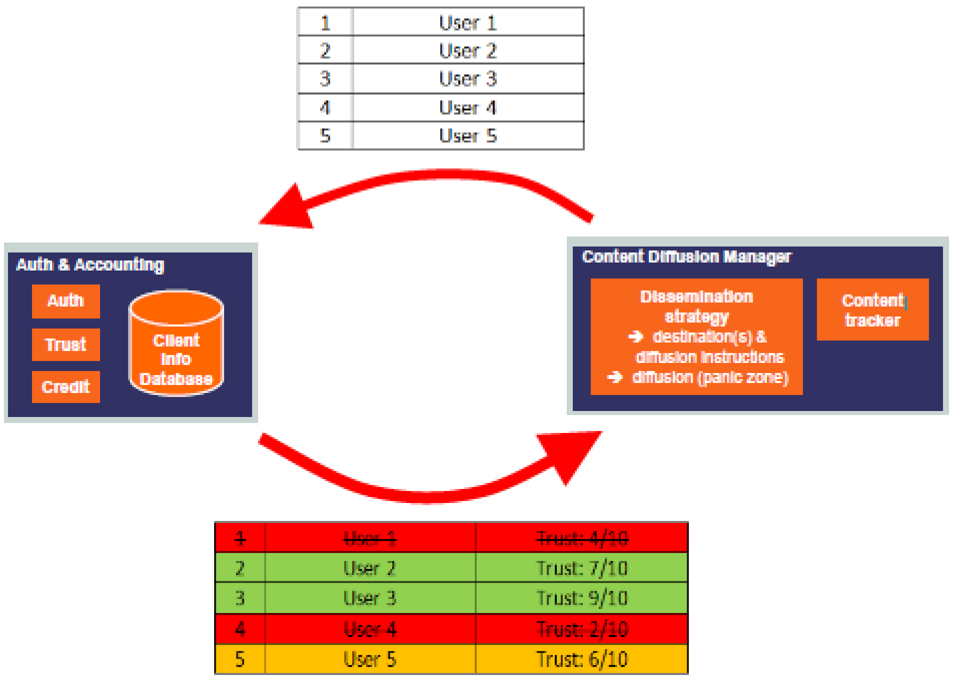


Figure 6. Rearrangement of user based on trust (1)



1	User 1	Trust: 4/10
2	User 2	Trust: 7/10
3	User 3	Trust: 9/10
4	User 4	Trust: 2/10
5	User 5	Trust: 6/10

Figure 7. Delivery of trust related information about potential seed users (2)



~~1~~	~~User 1~~	~~Trust: 4/10~~
2	User 2	Trust: 7/10
3	User 3	Trust: 9/10
4	User 4	Trust: 2/10
5	User 5	Trust: 6/10

Figure 8. Discard of user to be seeds based on trust profile (3)

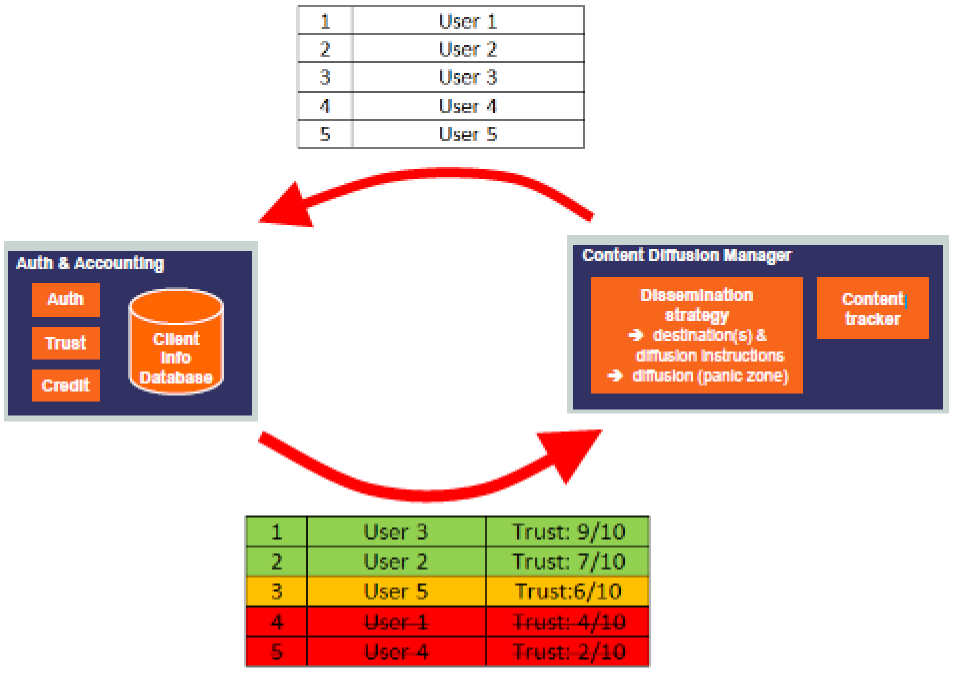


Figure 9. Combination of the different methods for seed recommendation delivery (4)

NOTE: The representation of the trust values is just an example and it does not have necessarily to match with the final trust values used in MOTO.

All these solutions enable to improve the efficiency of the propagation in the MOTO offloading scheme, reducing the emergence of paths cut.

6.2 Security considerations in the communications with external infrastructure operators

This section covers the security considerations in the communications between the MOTO Platform and the external infrastructure operators such as LTE or Wi-Fi operators.

As it has been stated in section 3, regarding web service communications, there are two main types of web services:

- JAX-RPC [10] based web services, in which a Service Provider publishes the service definition using a Web Services Definition Language (WSDL) and the Service Consumer sends a serialized XML message wrapped in a SOAP envelop across the wire.
- JAX-RS based web services [11], in which a Service Provider publishes the Resource name that can be used to consume the service, and the Service Consumer uses stateless operations from the HTTP protocol and sends requests and receives response messages. Because of the stateless nature of the operations, web services are called Representational State Transfer (REST) services. The message payload can either be in XML or JSON format.

REST is based on HTTP, and so the architectural style described by RESTful is simple, more open and scalable, and more consistent with other Internet protocols. It has a lot of advantages, and that's why some of the best known cloud services such as APIs from Amazon or Google are REST-based. Unfortunately, as it occurs in any interface, the security of REST-based APIs suffers from some vulnerabilities. The most common fails on REST based APIs are due to the fact that REST does not have a predefined security method. That means that some external method must be defined. Some of the more common vulnerabilities are:

- System Vulnerabilities
  - Poor coding practices
  - Configuration issues (weak encryption)
  - Lack of proper test and evaluation
- Web Application Vulnerabilities:
  - Session hijacking
  - Cross-Site Scripting (XSS)
  - SQL Injection
  - Format String Vulnerabilities
  - Inadequate authentication/authorization methods
  - Cross Site Request Forgery (CSRF)
  - Access control policies

In order to prevent vulnerabilities that can jeopardise the privacy of the data or the resources, a set of security measures must be taken into account. RESTful web services differentiate themselves from SOAP-based web services mainly in the simplicity of their design and implementation. However, they can become vulnerable when they are unsecured, especially when serving controlled data.

When it comes to communication between two entities one of the most important issues is authentication. Below some of the options for setting up authentication are presented:

- Basic authentication: This is one of the simplest ways to implement security as it does not require overhead of additional APIs, apart from the APIs used in the implementation framework itself. It should always be used with SSL encryption to prevent credentials being decoded.
- OAuth 1.0a: Oauth1 [12] is a signature-based protocol. It is a widely-used, well-tested, and very secure protocol. The biggest advantage of OAuth 1 is that it does not require passing the token secret across the wire directly. Because of this, it completely eliminates the possibility of tapping the password from over the communication channel. This can be safely used without SSL. However, this higher level of security offered in this protocol comes with an overhead of complex signature generating process.
- OAuth 2: This protocol [13] completely eliminates signatures; hence, it is much simpler. It requires Transport Layer Security, which handles encryption. Oauth2 is less widely used and less secured as compared to OAuth 1.0a. Therefore, it can be used for less sensitive data.

After having analysed the options above, it can be assumed that it is safer and simpler to choose Basic Auth with SSL for most REST services. However, high overhead based OAuth 1.0a protocol can be used for extremely sensitive data such as location data in less common situations.

6.3 Security considerations in the communications with terminals

This section covers the security considerations in the communications between the MOTO Platform and the end users' terminals.

As resulting from the protocol assessment performed in section 4, the ATOM protocol is the candidate selected to support the communications between the MOTO platform and the User terminals. In the next paragraphs, the security considerations for this protocol are exposed, along with recommendations for its implementation.

ATOM security

As the communication between MOTO platform and MOTO users is achieved through a potentially hostile channel and presumably insecure (wireless = open), the security design should be a paramount consideration. For surpassing the possible challenges for security that may appear in these communications, the security review of the ATOM protocol made in the IETF standard document [8], the Atom Syndication Format - RFC 4287 [15] and by the NIST Special Publication 800-95 [13] are taken into account.

The main threats that the communications between MOTO platform and the terminals are exposed to are the following:

- MOTO users devices could be compromised, or even be malicious or selfish.
- Communications that are performed in this scheme could be eavesdropped, intercepted or disclosed.
- Protocol behaviour could be analysed in order to attempt attacks.

The aim of the security implementation is to guarantee at least the following elements of security:

- Identification and authentication
- Authorization
- Integrity
- Privacy
- Confidentiality

As ATOM protocol uses XML to represent online content in the form of a "feed" and "entries," that are transmitted to an ATOM server over HTTP, and in order to provide the required security for the MOTO communications, there are two approaches that can be implemented:

1) **Securing the XML files**: The XML files transmitted can be secured in order to provide the security elements defined before, while existing XML security mechanisms can be used to secure its content [14]. In this sense, security standards developed by W3C allow XML content to be signed and encrypted [13]. The information to be transmitted in XML can be encrypted and signed, to avoid disclosure of personal data (such as location information), to prevent it from being manipulated (integrity), to identify the source of the information (identification). To do so, [14] states some requirements for the ATOM processors and documents:

    a. Atom Processors MUST NOT reject an Atom Document containing a XML signature because they are not capable of verifying it; they MUST continue processing and MAY inform the user of their failure to validate the signature.

    b. Atom Processors that verify signed Atom Documents MUST be able to canonicalize with the exclusive XML canonicalization method identified by the URI.

c. Atom Processors that verify signed Atom Documents MUST be able to verify RSA signatures, but do not need to be able to verify DSA signatures.

d. Atom Documents SHOULD NOT use MACs for signatures

e. The root of an Atom Document MAY be encrypted, using the mechanisms described by XML Encryption Syntax and Processing (http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/).

f. Atom Processors that decrypt Atom Documents MUST be able to decrypt with AES-128 in Cipher Block Chaining (CBC) mode.

g. Atom Processors that decrypt Atom Documents SHOULD check the integrity of the decrypted document by verifying the hash in the signature (if any) in the document, or by verifying a hash of the document within the document (if any).

h. If MACs are used for document authentication, the order MUST be that the document is signed and then encrypted, and not the other way around.

The approach is, therefore, based on the transmission of encrypted/signed XML files over HTTP.

2) **Securing the HTTP transmission**: The HTTP transmission can be converted into HTTPS. It is trivial to modify a Web service to support HTTPS [13]. In [8] it is stated that authentication mechanisms to prevent POSTing or editing by unknown or unauthorized clients are RECOMMENDED, in MOTO this is mandatory. In [8] some recommendations are made to avoid a list of known threats:

   a. Denial of Service: Atom Publishing Protocol server implementations need to take adequate precautions to ensure malicious clients cannot consume excessive server resources (CPU, memory, disk, etc.).

   b. Code Injection and Cross Site Scripting: Server implementations are strongly encouraged to verify that client supplied content is safe prior to accepting, processing, or publishing it.

This approach considers the delivery of unencrypted XML files over HTTPS.

3) Combining both approaches: In order to provide an optimal security, a combination of the two approaches is also feasible. XML encrypted and signed files can be sent over HTTPS, which provides a higher level of security. However, this approach could also increase the size of the messages and the resources consumption. To reduce this negative effect, the combination of both approaches can be made without implementing all the features of both approaches. For example, the XML file can be signed but not encrypted, and the encryption could be done only by HTTPS.

The final approach to provide security for the ATOM protocol implementation in MOTO must be further analysed. The requirements of the security scheme require that MOTO provides integrity, confidentiality and authentication within the communications between the MOTO platform and the users, and therefore, a good balance between security level and channel bandwidth and resources consumption must be searched.

Regardless of the exposed efficiency considerations, the following requirements for the implementation of this protocol must be taken into account:

- MOTO implementation of the ATOM protocol must support at least TLS 1.0, and ideally TLS 1.2.
- MOTO messages containing sensible information such as credentials, location, etc. must be encrypted.
- MOTO implementation of the ATOM protocol MUST support the transmission of X.509 v3 certificates.

- MOTO implementation of ATOM MUST allow for authentication where the MOTO user credentials are flowed in an Application Message from the users devices.
- MOTO implementation of ATOM must restrict access to MOTO platform services based on information provided by MOTO users.
- MOTO implementation of ATOM must provide capabilities to support OSCP (Online Certificate Status Protocol) to prevent revoked certificates from being used.

Other desirable functionalities for the MOTO ATOM processor are to identify and maintain a history of:

- Repeated connection attempts.
- Repeated authentication attempts.
- Abnormal termination of connections.

7. Conclusion

The present document has focused on defining the protocols to be used for the communications between the MOTO platform and the external entities (terminals, infrastructure operators and content providers).

For the Infrastructure API, two different alternatives have been analysed. On the one side, proprietary protocols have been considered and on the other side, web API technologies. The proprietary protocol alternative requires a huge effort on protocol definition and development. However, the web based API alternative is much simpler because the protocols are already defined and only their implementation needs to be defined in the MOTO context. Two different approaches have been taken into account for the web API: SOAP and RESTful. The first one is definitely the heavyweight choice, while the second one is easier to use for the most part and is more flexible. Finally, the RESTful approach has been selected for developing the Infrastructure API.

In the communications with terminals section, different alternatives have been assessed. On the one side, proprietary protocols have been considered and on the other side, HTTP (OMA-DM and Atom) and non HTTP protocols (MQTT) have been analysed. The first one, as on the previous case, has been discarded since it requires a huge effort on definition and development. At first sight, OMA-DM could be a good choice, but for MOTO's purposes it results a very demanding protocol, moreover its implementation is not so easy. MQTT instead is far more lightweight, less resource consuming and easy to implement than OMA-DM. However, not being based on HTTP, it could result disadvantageous in the long term. Finally, the Atom protocol has been selected since it is flexible, extensible and easy to use. Moreover, choosing an HTTP based protocol, will possibly let us substitute the communication protocol more easily, whether such necessity would arise.

The last one is the Application API. In this case, no protocol analysis has been done since it is analogue to the infrastructure API. Thus, a RESTful approach would be used.

Finally, the security considerations for the protocols for infrastructure offloading control and coordination have been presented with the objective of introducing the reader to the underlying security approach of MOTO. For the communications with infrastructure operators through the Infrastructure API a basic auth with SSL has been chosen. However, high overhead based OAuth 1.0a protocol can be used for extremely sensitive data such as location data in less common situations. For the communications with terminals the security review of the ATOM protocol made in the IETF standard document [8], The Atom Syndication Format - RFC 4287 [14] and by the NIST Special Publication 800-95 [13] have been taken into account. The final approach providing security for the ATOM protocol implementation in MOTO must be further analysed. The requirements of the security scheme require that MOTO provides integrity, confidentiality and authentication within the communications between the MOTO platform and the users, and therefore, a good balance between security level and channel bandwidth and resources consumption must be searched.

8. Appendix

8.1. Atom format

The MOTO server stores content following an RSS representation of XML as depicted by the figure below. The Atom Syndication Format is an open document format based on XML designed for syndicating periodical content such as blogs or news sites. UEs subscribe to a feed and can download the entries from this feed. Each entry can contain any type of content.

In the Atom architecture, the two main components are the feeds and the entries.

Below, a classic feed is shown. It is composed by the minimum items required for a feed.

a title	`<title type="text">MOTO</title>`
an id	`<id>4</id>`
a date	`<updated>2014-03-31T14:59:16.821Z</updated>`
an author	`<author><name>Thales</name></author>`

a list of entries	`<entry>          ... ...     </entry>`
	`<entry>          ... ...     </entry>!`

```
<?xml version='1.0' encoding='UTF-8'?>
<entry xmlns="http://www.w3.org/2005/Atom" xmlns:georss="http://www.georss.org/georss">
        <updated>2013-10-02T18:37:45.237Z</updated>
        <title type="text">New Entry</title>
        <id>6</id>
        <author><name>Thales</name></author>
        <summary type="html">
                        ... short summary ...
        </summary>
        <content type="html">
                        ... here it is the object itself ...
        </content>

</entry>
```

This is an entry, it is composed by some items, for each entry, a number of items are required for the creation:

required	**a title**	`<title type="text">New Entry</title>`
required	**an id**	`<id>6</id>`
required	**a date**	`<updated>2013-10-02T18:37:45.237Z</updated>`
required	**an author**	`<author><name>Thales</name></author>`

required	**a summary**	`<summary type="html">`
		`... short summary ...`
		`</summary>`

required	**a content**	`<content type="html">`
		`... here it is the object itself ...`
		`</content>`

8.2. RESTFUL Web Service

RESTFUL services are built to work best on the web. Representational State Transfer (REST) is an architectural style consisting of a coordinated set of architectural constraints applied to components, connectors and data elements, within a distributed hypermedia system. The REST architectural style is applied to the development of web services as an alternative to other distributed-computing specifications such as SOAP. The architecture specifies constraints such as a uniform interface that if applied to a web service induces desirable properties, such as performance, scalability and modifiability that enable services to work best on the web.

The REST architectural style constrains client/server architecture and is designed to use stateless communication protocol like HTTP.

RESTful web services expose a set of resources identified by Uniform Resource Identifiers (URIs). These provide a global addressing space for resource and service discovery. All resources manipulated use a fixed set of four operations, the CRUD operations (Create, Read, Update and Delete) representing a POST, GET, PUT and DELETE HTTP Request.

The POST request creates a new resource, which can be then deleted by a DELETE request. GET request retrieves the current state of a resource. PUT can transfer a new state of a resource.

The Table 7below shows the methods needed to implement the communication between the MOTO Platform and the end users' terminals:

Table 7: Methods needed to implement for the communication between the MOTO Platform and the end users' terminals.

Description	**HTTP Request**	**URL**	**Attributes**	**Result**
Create feed	GET	/		Feed id
Get feed	GET	/{idFeed}		Feed with contents and acks
Get entries	GET	/{idFeed}/entries		Feed + list of contents
Get entry	GET	/{idFeed}/entries/{idEntry}		Entry
Get acks	GET	/{idFeed}/entries/{idEntry}/Ack/		Feed + List of acks
Add feed	POST	/	Atom Feed	Feed with new id

Description	HTTP Request	URL	Attributes	Result
Add entry	POST	/{idFeed}/entries	Atom Entry	Content with new id
Edit entry	PUT	/{idFeed}/entries	Atom Entry	Content updated
Edit acks	PUT	/{idFeed}/entries/{idEntry}/Ack/	Atom Entry	Ack updated
Remove entry	DELETE	/{idFeed}/entries/{idEntry}		

The APIs provide different types of annotations to be compatible with a REST architecture.

To communicate with the MOTO server, the UE sends a request to a specific URI and depending of the annotation, the MOTO server will answer with an ATOM document. For some URI, the request needs an attribute like an ATOM entry. In the following example, we explain the behavior of the MOTO server depending on the type of transaction.

For content retrieval, the UE is notified through a websocket that it has to retrieve the content marked with id msg_id at the MOTO SERVER using the celular connectivity. Then the UE performs an HTTP GET on the address /feedsync/rest/myfeeds/4/entries/msg_id, the MOTO server will get the entries of the feed with the msg_id = "/5" from the database and returns to the UI an ATOM documents.

```
> GET /feedsync/rest/myfeeds/4/entries/5 HTTP/1.1
> User-Agent: curl/7.35.0
> Host: 192.168.20.239:8083
> Accept: */*
> Content-Type: application/atom+xml
> Content-Length: 511

< HTTP/1.1 200 OK
< Server: Apache-Coyote/1.1
< Content-Type: application/atom+xml
< Transfer-Encoding: chunked
< Date: Wed, 02 Apr 2014 12:33:48 GMT

<?xml version='1.0' encoding='UTF-8'?>
<feed xmlns="http://www.w3.org/2005/Atom">
<title type="text">Moto</title><id>9</id><link href="" /><link href="" rel="self"
/><updated>2014-05-28T06:47:26.400Z</updated><author />
<entry xmlns:georss="http://www.georss.org/georss">
<title type="html">Photo</title>
<content type= "html">http://192.168.1.2:8083/feedsync/media/5.jpg </content>
<author><name>2</name></author>
<summary type="html"></summary><id>5</id><georss:point>0.0
0.0</georss:point><updated>2014-05-20T14:57:52.000Z</updated><link
href="/9/entries/5" rel="edit" type="application/atom+xml" length="0" /><link
rel="replies" href="http://192.168.20.239:8083/feedsync/rest/myfeeds/5/comments-5"
type="application/atom+xml" /></entry></feed>

</feed>
```

Figure 10: HTTP GET Request

For content posting (i.e. by the content provider in the MOTO framework) an HTTP POST is performed on the address */feedsync/rest/myfeeds/4/entries/msg_id/,* the MOTO server will receive and decode the entry, add in his database then returns the updated entry with its id.

For acknowledging the content reception HTTP PUT is performed on the address */feedsync/rest/myfeeds/4/entries/msg_id/Ack*. The acknowledged content id for is implicit in the address.

Upon reception of the PUT message, the server reads and decodes the entry, updates the data in his database, then returns the updated Ack entry. Inside the Author block we may find the MOTO_UE_ID of the UE performing the ack.

```
> POST /feedsync/rest/myfeeds/4/entries HTTP/1.1
> User-Agent: curl/7.35.0
> Host: 192.168.20.239:8083
> Accept: */*
> Content-Type: application/atom+xml
> Content-Length: 511

< HTTP/1.1 201 OK
< Server: Apache-Coyote/1.1
< Content-Type: application/atom+xml
< Transfer-Encoding: chunked
< Date: Wed, 02 Apr 2014 12:33:48 GMT

<?xml version='1.0' encoding='UTF-8'?>
<entry
xmlns="http://www.w3.org/2005/Atom"
xmlns:georss="http://www.georss.org/georss">
    <updated>2013-10-02T18:37:45.237Z</updated>
    <title type="text">New Entry</title>
    <id>6</id>
    <author><name>Thales</name></author>
    <summary type="html">
        ... short summary ...
    </summary>
    <content type="html">
        ... here it is the object itself ...
    </content>
    <georss:point>38.719537 -9.30577</georss:point>
           <link
       href=http://192.168.20.239:8083/feedsync/rest/myfeeds/4/entries/6
       rel="edit" type="application/atom+xml" length="0" />
</entry>
```

Figure 11: POST HTTP Request

The Table 8 below shows the methods needed to implement the communication between the MOTO Platform and the network infrastructures (through the Infrastructure APIs):

Table 8: Methods needed to implement for the communication between the MOTO Platform and the Infrastructure Operators.

Description	**HTTP Request**	**URL**	**Attributes**	**Result**
Get feed for {idOperator}	GET	/{idOperator}/		Feed with available information from the operator's network

Description	HTTP Request	URL	Attributes	Result
Get topology entries	GET	/{idOperator}/topology/		Topology for each physical radio access network of the operator
Get specific topology entry	GET	/{idOperator}/topology/{idDevice}		Topology entry of {idDevice}
Get network status entries	GET	/{idOperator}/netstats/		Network status for each physical radio access network of the operator
Get specific network status entry	GET	/{idOperator}/netstats/{idDevice}		Network status entry of {idDevice}

Similarly to the communications between MOTO platform and user terminals, the APIs expose different types of annotations to be compatible with a REST architecture. However, please note that information on local topology and network status could be retrieved and aggregated by operators using a Pub-Sub mechanism or other methods. Since the information exchanges related to these functionalities take place outside of the MOTO service, we do not describe them in detail. Moreover, the MOTO platform can only retrieve these information (GET operations only).

In the following examples, we give two examples of the behavior of the MOTO service depending on the type of transaction.

For retrieving the topology of the user terminals connected to the AP with id MAC_ADDRESS_1, the MOTO platform performs a HTTP GET through the Infrastructure APIs with the address /FON/topology/MAC_ADDRESS_1. The operator will get the entries of the topology feed with the id "/MAC_ADDRESS_1" from its database and returns to the MOTO platform the ATOM documents:

```
> GET /FON/topology/MAC_ADDRESS_1 HTTP/1.1
> User-Agent: curl/7.35.0
> Host: xxx.xxx.xxx.xxx:port
> Accept: */*
> Content-Type: application/atom+xml
> Content-Length: 511

< HTTP/1.1 200 OK
< Server: Apache-Coyote/1.1
< Content-Type: application/atom+xml
< Transfer-Encoding: chunked
< Date: Wed, 07 Jul 2014 12:35:48 GMT

<?xml version='1.0' encoding='UTF-8'?>
<entry xmlns="http://www.w3.org/2005/Atom"
xmlns:georss="http://www.georss.org/georss">
       <updated>2014-07-07T11:35:45.237Z </updated>
       <title type="text">Topology</title>
       <id>MAC_ADDRESS_1</id>
       <author><name>FON</name></author>
       <summary type="html">
        Topology of AP MAC_ADDRESS_1
       </summary>
       <content type="html">
              ... List of UEs connected to the AP ...
       </content>
      <georss:point>38.719537 -9.30577</georss:point>
       <link href="http:// xxx.xxx.xxx.xxx:port/FON/topology/MAC_ADDRESS_1"
       rel="alternate" type="application/atom+xml"/>
    </entry>
```

Figure 12: HTTP GET request.

Similarly, when the MOTO platform wants to retrieve the information on the network status of the AP with id MAC_ADDRESS_2, it performs a HTTP GET through the Infrastructure APIs with the address /FON/netstats/MAC_ADDRESS_1. The operator will get the entries of the network status feed with the id "/MAC_ADDRESS_2" from its database and returns to the MOTO platform an ATOM documents:

```
> GET /FON/netstats/MAC_ADDRESS_2 HTTP/1.1
> User-Agent: curl/7.35.0
> Host: xxx.xxx.xxx.xxx:port
> Accept: */*
> Content-Type: application/atom+xml
> Content-Length: 511

< HTTP/1.1 200 OK
< Server: Apache-Coyote/1.1
< Content-Type: application/atom+xml
< Transfer-Encoding: chunked
< Date: Wed, 07 Jul 2014 11:33:48 GMT

<?xml version='1.0' encoding='UTF-8'?>
<entry xmlns="http://www.w3.org/2005/Atom"
xmlns:georss="http://www.georss.org/georss">
       <updated>2014-07-07T11:33:47.534Z</updated>
       <title type="text">netstats</title>
       <id>MAC_ADDRESS_2</id>
       <author><name>FON</name></author>
       <summary type="html">
        Statistics of AP MAC_ADDRESS_2
       </summary>
       <content type="html">
              ... AP status (No of UEs connected, remaining capacity, etc.)...
       </content>
      <georss:point>38.5431 -9.0127</georss:point>
       <link href="http:// xxx.xxx.xxx.xxx:port/FON/netstats/MAC_ADDRESS_2"
       rel="alternate" type="application/atom+xml"/>
    </entry>
```

Figure 13: HTTP GET request.

9. References

[1] D2.2.1: General architecture of the Mobile Offloading system (release a), MOTO Consortium, October 2013.

[2] D3.1: Initial results on offloading foundations and enablers, MOTO Consortium, October 2013.

[3] DROiD: Adapting to Individual Mobility Pays Off in Mobile Data Offloading, IFIP Networking, 2014

[4] Push-and-Track: Saving Infrastructure Bandwidth Through Opportunistic Forwarding, Pervasive and Mobile Computing, 2012

[5] Adaptive Data Offloading in Opportunistic Networks through an Actor-Critic Learning Method, ACM CHANTS, 2014

[6] WDSL http://www.w3.org/TR/wsdl

[7] Chapter 5 of Fielding's dissertation is "Representational State Transfer (REST)" (http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm)

[8] OMA DM 2.0 (http://technical.openmobilealliance.org/Technical/technical-information/release-program/current-releases)

[9] Atom Publishing Protocol (http://tools.ietf.org/html/rfc5023)

[10] JAX-RPC (https://java.net/projects/jax-rpc/)

[11] JAX-RS (https://jcp.org/en/jsr/detail?id=311)

[12] Oauth 1.0a (http://tools.ietf.org/html/rfc5849)

[13] Oauth 2.0` (http://tools.ietf.org/html/rfc6749)

[14] NIST Special Publication 800-95 (http://csrc.nist.gov/publications/nistpubs/800-95/SP800-95.pdf)

[15] The Atom Syndication Format RFC 4287 (http://tools.ietf.org/pdf/rfc4287.pdf)

[16] MQTT http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html

DISCLAIMER