Remote Vehicle Interaction

Architecture High Level Description

RVI Release v0.2

Magnus Feuer
mfeuer@jaguarlandrover.com

Revisions

Revision	Date	Author	Notes
Draft 1	2014-06-11	Magnus Feuer	First Draft.
Draft 2	2014-06-11	Magnus Feuer	Review feedback from Arthur Taylor integrated.
Draft 3	2014-06-12	Magnus Feuer	Additional feedback from Arthur Taylor and Paul Hanchett integrated.
Draft 4	2014-06-14	Magnus Feuer	Formatting and minor corrections
Draft 5	2014-06-26	Magnus Feuer	Renamed document, removing the Tizen reference. Updates from security review: 1. Change document license to Creative Commons. 2. Remove all security between Service Edge and services Securing the service - Service Edge link security is now scoped out to implementation / deployment / ops. 3. Add certificate timestamp field. Allows chronological sorting of certificates 4. Add node detection technique examples 5. Add per-user service access to the Issue list 6. Add suggestion on how to avoid node spoofing to issue list
A	2014-09-22	Magnus Feuer	Updated to reflect implementation of RVI v0.2. 1. Renamed Store and Forward component to Schedule. 2. Topic tree renamed service name 3. Use case in chapter 8 rearranged to reflect v0.2

Table of Contents

Contents

1. References

[1]	Google Protocol Buffers	https://code.google.com/p/protobuf/
[2]	Requirement Specification	TBD

2. Acronyms and definitions

RVI	Remote Vehicle Interface
SOTA	Software Over The Air
TSP	Telematics Service Provider
RVAM	Remote Vehicle Access Manager
Node	A vehicle, mobile device, or backend server running RVI-integrated services
Component	An internal service inside the RVI system
Manager	A higher level component
Service	An external application connected to the RVI system
Service Name	A global registered name used to identify and locate a specific service.
Network Address	An URL, IP address, MSISDN, or similar that a node can be reached at

3. Introduction and Purpose

This document introduces the Remote Vehicle Interaction (RVI) architecture on a high level, outlining its design, the components, and the core use cases. It is intended for the reader who wants a comprehensive overview of the RVI and its inner workings. Once this document has been read, the reader can continue with the DDS (see below) for specific components to understand their implementation requirements.

The purpose of the RVI architecture is to enable vehicles, mobile devices, and backend servers to exchange today's and tomorrow's connected car services in a robust, secure, and versatile manner. The RVI will place a minimum number of requirements and restrictions of the services that use it, aiming at giving applications a maximum degree of freedom in their implementation. Examples of applications using RVI are remote door unlock, software upgrades (through the dealer, OTA and USB sticks), climate control from mobile device, syncing of media files between the cloud and the vehicle, geo fencing, etc.

The RVI architecture describes a set of components and services that form a distributed, sparsely connected, secure P2P network where vehicles, mobile devices, and backend servers can access each other.

The complete set of RVI architecture documents will be used as the basis for a reference open source implementation of the RVI that will be released to the community.

The philosophy behind the RVI effort is that the reference implementation drives the specification, and that any design issues resolved in the implementation flows down to the specification. Thus, if the implementation and the specification are in conflict with each other, the implementation takes priority. The feedback from the implementation will be used to refine the RVI architecture and specification itself.
Currently, all component APIs are using JSON-RPC. Future revisions may extend this to support additional protocols.

3.1. Document Structure and Reader Assumptions

The RVI is described in the following hierarchy of documents:

1. **RVI AGL PowerPoint presentation 2014-06-18 (rev 2)**
   Gives a summary of the design, what it does, and how it operates

2. **High Level Description (HLD) (this document)**
   Goes through the overall schematics, its included components, and core use cases.

3. **Per Component Detailed Design Specification (DDS)**
   Each component described by the HLD is described in detail in their own DDS document. Descriptions, Schematics, API specifications are included for each component DDS.

It is recommended that the documents are read in the order given above.

The reader of this HLD is assumed to be familiar with the following areas:

1. **Mobile terminology**
   Terms such as 3G, handset, pppd, and mobile data links will be used throughout the HLD.

2. **Automotive applications**
   Media Players, door lock management, and other applications will be used as examples.

3. **Networking**
   Distributed system terminology, IP-terms, and web technologies such as JSON-RPC and HTTP will be used.

3.2. Issues

This specification is not complete. Below is a list with currently identified issues and shortcomings that should be addressed.

3.2.1. Federation

The current specification leaves it up to each individual node to trust the certificates sent out by a provisioning server. There is no way for a node to validate the legitimacy of a provisioning server.

3.2.2. Data Link node discovery

There is no specification for how two nodes should find each other prior to running the authorize/announce use case. While this is technically in the domain of each Data Link implementation, best practices such as UDP broadcasts, connection attempt to well-known addresses, etc, should be documented.

3.2.3. Certificate revocation

The revocation model provided in the current revision of the specification is a primitive blacklist distribution that may not scale depending on the communication channels available.
This model can be improved upon by having P2P propagation. There may also be a better general solution for revocation than blacklisting.

3.2.4. Request transaction id

The current specification does not indicate the necessity for a request-sending service to provide a monotonically increasing transaction id with each request. This opens up the Service – Service Edge up for replay attacks. A transaction id should be provided by the sending service, which is then translated by Service Edge to an internal transaction id that will follow the transaction through the network. The reason for the transaction id switch is to avoid duplicate IDs being provided by two different services; the Service Edge-generated ID is guaranteed to be RVI-wide unique.

3.2.5. Certificate format

There is an IETF draft for JSON-based web encryption. The certificate format in that draft is a candidate to replace the format used by the current version of this HLD.

3.2.6. User-level Node authorization

Currently the authentication and authorization system of the platform that a Node runs on is ignored. Future versions of RVI may want to integrate user-level access control to determine which users can access which services on a Node.

3.2.7. Node spoofing prevention

There is currently no way to detect that a node's private key and certificates have been stolen and are used to impersonate that node. A partial solution is to have a local node instruct its remote counterpart to provide a specific token in the authorization process next time it is connected to the local node. If the remote node is impersonated, and there are currently two nodes with a single identity, one of the two nodes will fail to provide the correct token during its subsequent authorization.

There is, however, currently no way to validate that an authorizing node is executing on a specific hardware unit.

4. Requirements and Objectives

There are several objectives for the RVI design and its reference implementation.

The core mission is to provide a specification and implementation that is easy to adopt for vendors and customers, who are either are starting from a blank page or want to integrate their existing solution into the RVI framework.

Please see [2] for a formal functional and non-functional requirement specification.

The following chapters outline the high level requirements and objectives being pursued.

4.1. Internet Connectivity Optional

If two nodes see each other over a communication link, their services must still be able to exchange traffic even if neither node has an Internet connection. The typical use case is an owner who wants to unlock and start a car in a garage using his/her mobile phone, and wants to do this even if there is no carrier signal. The access app on the mobile phone must be able to talk to the access service in the vehicle without having to authenticate toward a (non-reachable) central server.

4.2. Zero Service Configuration

Given the Internet requirement above, a new service, be it a mobile phone app, an in-vehicle HVAC controller, or a cloud-based traffic information service, must be able to register themselves without updating a central repository. A new service should only have to present correctly signed requests to the RVI system in order to register and access other, remote services.

Please note that nodes (hosting services) still need to be provisioned with addresses, protocols, and access rights, all expressed through certificates.

4.3. Decentralized Service Discovery

Nodes need to discover services on other nodes without involving a central repository so that two nodes without an Internet connection can explore each other's available services. This is also true for when two nodes communicate with each other over non-IP based links such as Bluetooth, IR, or even USB sticks (software updates or content transfer).

4.4. Self-Carried Authorization

When two nodes have discovered each other's services, they need to authenticate their right to access them. With the optional internet connectivity requirement, these authorizations have to be made in a pure peer-to-peer fashion without third party involvement.

4.5. Interoperability

An organization shall be able to implement individual components and drop them into an existing set of RVI components without compatibility issues, given that they use the API protocol (JSON-RPC, etc).

4.6. Connectivity Awareness

Both Services and individual RVI components need to know if a remote node and its services can currently be reached or not. This spans from a mobile device knowing if a peer-to-peer connection for a specific vehicle is available or not, to a cloud-based media server wanting to know the available bandwidth to a media player, to a vehicle wanting to know its currently available data channels.

4.7. Sparse Connectivity Support

Since data links come and go, often with short notice, and services on two nodes may not see each other for weeks at the time, the RVI shall handle resend attempts, delivery validation, timeouts, and store & forward for transactions in order to support sparse connectivity.

4.8. Hybrid Deployment Support

RVI nodes will have to integrate with servers and vehicles using other designs and products. An RVI deployment must, with custom protocol implementations, be able to connect and interact with other telematics and M2M systems.

5. Architecture Overview

Below is a layout of a typical RVI deployment, where the blue and green components form the core of the architecture.

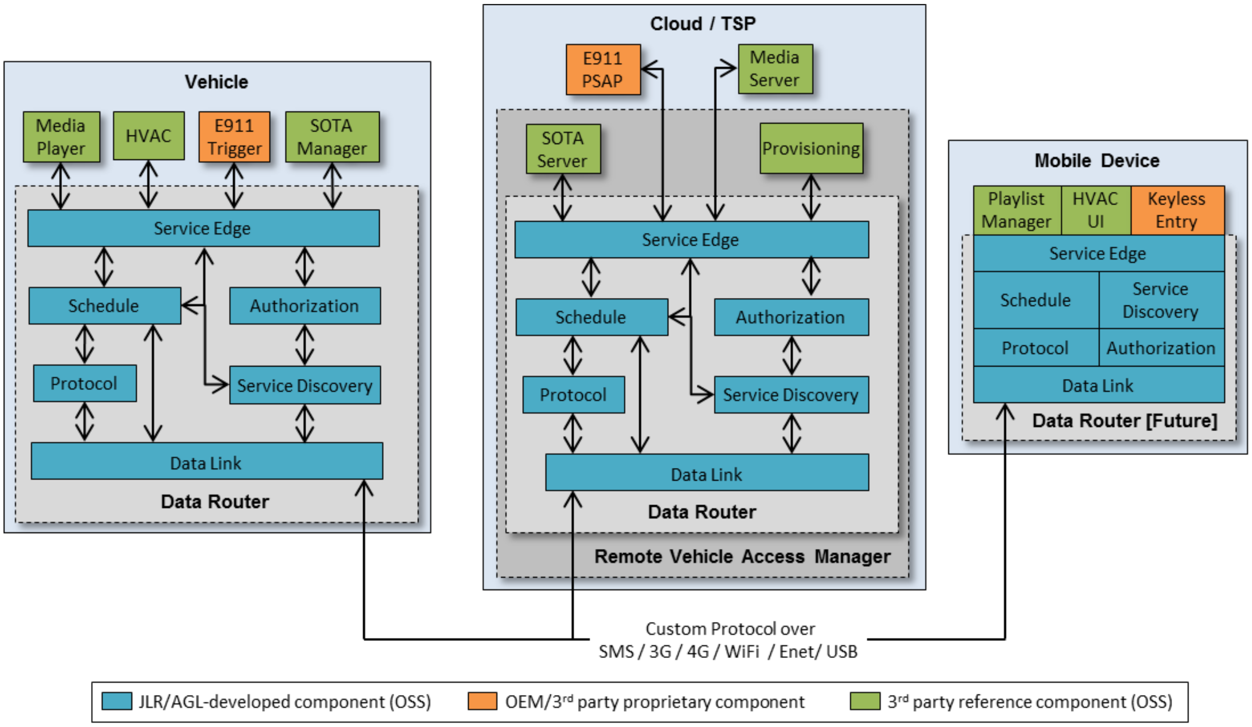


Figure 1 - Architecture Overview.

The API of the individual components are designed as standalone as possible, allowing the components to be used outside the original RVI environment. Services can communicate peer-to-peer with each other, allowing, for example, a mobile device to interact directly with a vehicle, even if neither have an Internet connection to a Remote Access Vehicle Manager.

The components are described in the following chapters.

5.1. Data Router

A high-level aggregate of lower-level components that delivers transactions between services robustly and securely over sparsely connected networks. Different compositions of the data router executes inside the vehicle, on the backend server, and in mobile devices.

5.2. Remote Vehicle Access Manager (RVAM)

The server-side RVAM aggregates the Data Router, Billing & Charging, Software Over The Air (SOTA), and Provisioning into a single server system that extends the service transaction routing with external device and software management, as well as billing, charging, and reconciliation.

5.3. Service Edge

Service Edge, present inside the Data Router acts as a service-facing API coordinating all traffic sent to and from local services. It is responsible for authorizing and routing requests to their targeted end points.

5.4. Schedule

Schedule receives requests from Service Edge, and, in case the addressed service cannot be reached, stores the request until a data link to the service becomes available. It is also responsible for management communication channels to other nodes (and their services) through Data Link.

5.5. Protocol

Protocol components (there can be many) receive requests from Service Edge and format them as payloads to be transmitted by Data Link to their counterparts on a remote node. Conversely, incoming data payload from Data Link is decoded by the Protocol before it is handed over in a standardized format to Service Edge for validation and forwarding to its destination service. A protocol can also, optionally, bypass Data Link and communicate directly with its counterpart. (Not shown in Figure 1.)

5.6. Data Link

Responsible for bringing up and tearing down data channels to remote nodes. Typical data links are WiFi, SMS, 3G, 4G (over PPP), Ethernet, and USB sticks containing requests. Data Link can be ordered by Store and Forward to setup or disconnect a data link, and will report to subscribing parties when link to a remote node becomes available or unavailable.

5.7. Service Discovery

Service Discovery tracks service availability local and remote nodes. The Service Edge queries Service Discovery in order to retrieve a target node address for a specific request. Service Discovery will opportunistically use available data links to push Service Lists to other Service Discoveries on other nodes as they become available for communication.

5.8. Authorization Manager

The authorization manager adds necessary signatures and certificates to outgoing transactions to remote nodes, allowing requests to be validated prior to execution by the targeted services. Authorization will also validate incoming requests from remote services before Service Edge sends them on to their local service destinations.

5.9. Provisioning Server

The Provisioning server, a part of the RVAM, supports Service Discovery with information about nodes (mobile devices, vehicles, and servers) available in a RVI network, and how they can be reached. Provisioning also supports the Authorization Service with certificates used to sign outgoing requests to prove their authenticity. In many instances, Provisioning is a gateway to one or more external provisioning services hosted by TSPs, Enterprise IT, and other organizations.

5.10. Software Over The Air (SOTA) [Future – v0.3]

Software Over The Air is, from a strict RVI perspective, a generic service with no special access to the internal RVAM / Data Router components. Since SOTA (and Firmware Over The Air) is so central to the RVI, it has been included in the architecture as a specification and reference implementation. The SOTA server, however, can be replaced without modifying any other components apart from its SOTA Manager counterpart on the vehicle.

5.11. Mobile Device Data Router [Future]

Mobile devices, with their often constrained execution environment, can sometimes not be suitable for having a loosely coupled set of components forming the Data Router. Instead, these targets may need a monolithic service or library that presents a similar interface

6. RVI-Wide Concepts

6.1. RPC, Message, and Metric Request Type

A Service can send three different types of requests to a remote counterpart, depending on the type of payload that is to be delivered. The request types are:

1. **RPC [Future - v0.3]**
   Remote Procedure Call. A request sent from one Service to another, where the originating Service expects a reply. If the request cannot be delivered, the originating Service will be notified. A Service can issue a "call" command to invoke an rpc.

2. **Message**
   An asynchronous request sent from one Service to another, where no reply is expected by the originating Service. If the message cannot be delivered, the originating Service will not be notified.
   A Service can issue a "send" command to send a message.

3. **Metrics [Future]**
   A set of requests to subscribe to metrics, published by another Service, that are to be delivered to the originating Service when updated.
   A Service can issue a "subscribe" or "unsubscribe" command to setup or cancel a subscription.
   A Service can issue a "publish" command to send out an updated service value to all its subscribers.

6.2. Service Names

A service, connected to an RVI node, is identified by a service name that is globally unique across all RVI nodes in a service name tree, meaning that a specific service name resolves globally to the same physical service instance for all nodes.

The syntax of a service name is:

```
[type]:[organization]/[path]
```

Figure 2 - Service name syntax.

The components of the entry are as follows:

1. **`[type]` [Future v0.3]**
   Specifies the type of the service, determining how it is invoked. Initial types are `rpc`, `metric`, and `message`, although more can be added in future revisions of the RVI. See "RPC, Message, and Metric Request Type" for details.

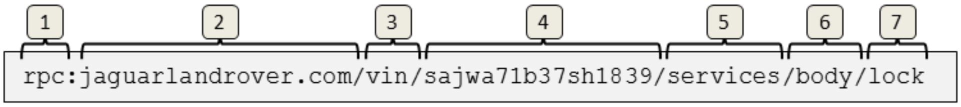
2. **`[organization]`**
   A domain name describing the organization that hosts a sub-section of the root. All service paths under the domain name are managed by the given organization.

3. **`[path]`**
   A path describing the given service hosted by the organization.

If the `[type]:` section is omitted, it will default to `message`.

Please note that several organizations can co-exist in a single service name tree. An organization is simply a top-level divider between different sub-sections of the tree.

Below is an example of a service name, with numbered components.



```
rpc:jaguarlandrover.com/vin/sajwa71b37sh1839/services/body/lock
```

Figure 3 - Typical Service Entry.

The components are as follows

1. **Service Type**
   Specifies that the named service is invoked as a remote procedure call.

2. **Organization**
   Specifies that this service is hosted by Jaguar Land Rover.

3. **VIN sub-tree**
   A sub-section of the service name space under which all registered VINs and their services are hosted.

4. **VIN**
   The VIN of the vehicle running the service

5. **Services sub-tree**
   A sub-section of the service name hosting all services running on the given vehicle.

6. **Service name**
   The name of the service running on the given vehicle

7. **Command name**
   The name of the command to execute on the given service and vehicle.

Please note that only components 1 and 2 are standardized. All other components in a service name can be specified by the organization hosting the tree.

Since a service name is system-wide unique, i.e. a service name resolves to the same specific service for all nodes in a network, the identity of a node that hosts the service is embedded into the service name. In the examples above, the node is identified by the vehicle VIN running the body service in a car.

When a request is to be sent to the service using a service name, the service name will be resolved to a network address, which will then receive the request. See chapter “Node addressing” for a description of how a service names is translated to a network address.

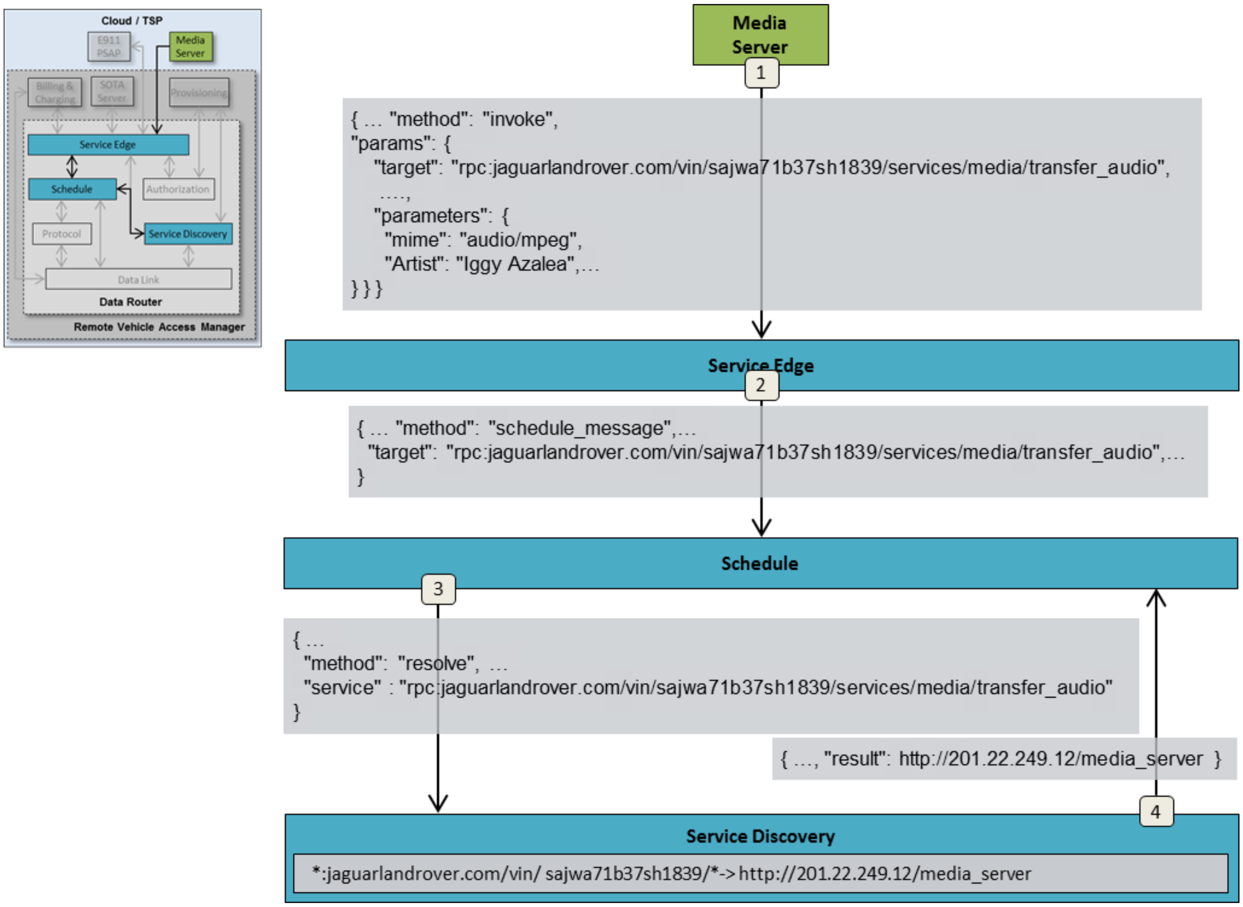
6.3. Node addressing

Nodes, setup in the Provisioning Server, are servers, vehicles, devices running the RVI and a number of services. Each node can communicate with one or more other nodes over various data links.

A service name is resolved to a network address by Service Edge, which will keep track in real time of how to reach all services known to it.

Since it is up to Service Discovery to convert a service name to a network address, the exact procedure of converting a service name to a network address is implementation dependent.

The figure below shows the high-level call flow for address resolving a service, identified by its name, to a network address.



```
{ … "method": "invoke",
"params": {
   "target": "rpc:jaguarlandrover.com/vin/sajwa71b37sh1839/services/media/transfer_audio",
   ….,
   "parameters": {
     "mime": "audio/mpeg",
     "Artist": "Iggy Azalea",…
}}}
```

```
{ … "method": "schedule_message",…
  "target": "rpc:jaguarlandrover.com/vin/sajwa71b37sh1839/services/media/transfer_audio",…
}
```

```
{ …
  "method": "resolve", …
  "service" : "rpc:jaguarlandrover.com/vin/sajwa71b37sh1839/services/media/transfer_audio"
}
```

```
{ …, "result": http://201.22.249.12/media_server }
```

```
*:jaguarlandrover.com/vin/ sajwa71b37sh1839/*->http://201.22.249.12/media_server
```

Figure 4 - Node to network address resolution.

The steps above are as follows.

1. **Send Request service edge**
   The Media Server sends a request, to be forwarded to the vehicle with vin sajwa71b37sh1839, to Service Edge

2. **Forward to Scheduler**
   Service Edge forwards the request to Scheduler, which will manage data link setup and data transmission.

3. **Lookup node address**
   Scheduler sends a request to Service Discovery, providing the full service name, as received from Media Server, in order to retrieve the network address of the node (vehicle) that hosts the targeted service.

4. **Return network address**
   Service Discovery, having matched the service name against its internal database, returns the network address, which in this case is a URL, to Scheduler.

Please note that this is a simplified scenario since no data link types (3G, WiFi, SMS, etc), specifying the recommended delivery methods to the destination node, is returned by Service Discovery.

6.4. RVI Security Scope

Security, in the context of this architecture, is the process of authenticating services to access other services in an RVI system. A service should only be able to discover and access other services it has received authorization to from a provisioning system trusted by both the sending and receiving RVI nodes

The constraints imposed by the sparse connectivity requirement means that each node, and its services, must be able to authenticate themselves toward other nodes without having to rely on a connection to a central provisioning server.

The security management of the RVI architecture covers the following areas:

1. **Certificate Generation**
   The provisioning server has support for generating certificates granting access for a given service to one or more other services. See chapter "Certificates" and "Authorization" for details on service's public key, can authenticate traffic sent from the service.

2. **Certificate Distribution**
   Once a certificate has been generated by Provision Service, it has to be sent out to the node that the certificate was generated for. This transmission can be implemented through an RVI built-in certificate service.

3. **Certificate Revocation**
   All certificates are specified with a time interval within which they are active. Sometimes, however, certificates need to be revoked prior to their expiration date. Such revocation can also be implemented through the certificate service.

The security management of the RVI does **not** cover the following areas.

1. **Communication Channel security**
   The actual transmission protocol between two Protocol (or Data Link) instances should be secured using SSL/TLS or similar mechanisms. The exact security mechanisms employed for this is to be determined by the Protocol implementations, and are outside the scope for the RVI itself.

2. **Node – Certificate linking**
   Since the RVI will operate on many devices, platforms, and network protocols, there is no uniform way of linking a certificate to a specific device using a mac address, CPU ID, etc. As a consequence, there is no way for a node receiving a certificate to validate that the sending device is actually the owner of that certificate. If a device manages to steal a certificate and the private key from a node, the device will then be able to impersonate the node and hijack its traffic.

3. **Node protection of certificates**
   Another consequence of the unknown devices and platforms that will run RVI, the storage and protection of certificates and key pairs on a node is outside the scope of the RVI design and has to be addressed by the implementation of individual Authorization components.

4. **Service – Service Edge Authentication and Authorization**
   Services are implicitly trusted by Service Edge and are expected to execute inside the trusted domain of a node. It is up to the implementation, deployment, and operations of an RVI system to ensure that no illicit services gain access to Service Edge.

7. Service Management

Service Management consists of a set of components, use cases and APIs that enables services to:

1. **Register with the RVI**
   A service can connect to the RVI (through Service Edge) and register itself.

2. **Announce themselves**
   A registered service can announce their availability not only to the local node they are executing on, but also to other, remote nodes in the RVI network.

3. **Discover other services**
   A service can receive availability announcements from other services, with all information necessary to connect and send requests to them.

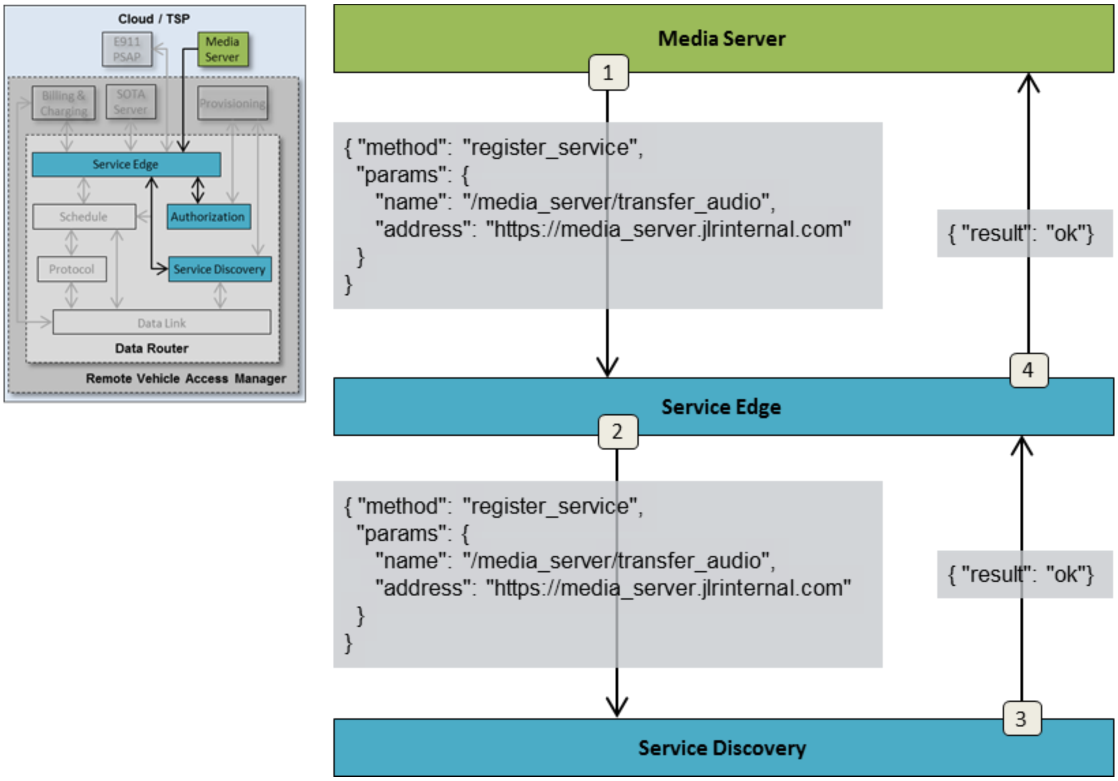
4. **Authorize themselves**
   A service can authorize themselves toward other services, which in their turn can validate the originating service.

5. **Invoke other services**
   Once authorized, a service can send requests to other, remote services, having them carry out tasks and send back the result.

The following chapters describe the service management on a high level.

7.1. Service Registration

A Service can register itself with the RVI through Service Edge to set itself up for sending and receiving requests, as is shown below.



```
{ "method": "register_service",
  "params": {
     "name": "/media_server/transfer_audio",
     "address": "https://media_server.jlrinternal.com"
  }
}
```

```
{ "result": "ok"}
```

```
{ "method": "register_service",
  "params": {
     "name": "/media_server/transfer_audio",
     "address": "https://media_server.jlrinternal.com"
  }
}
```

```
{ "result": "ok"}
```

Figure 5 - Service Authorization.

The steps above are as follows.

7.1.1. Step 1 - Register with Service Edge

The Media Server sends a request to register itself with Service Edge.

The request contains the following elements:

1. **Service Name**
   The service name to be registered with the node.

2. **Service Address**
   Specifies where t to forward requests and replies targeting the service.

7.1.2. Step 2 - Register with Service Discovery

The service is forwarded by Service Edge to Service Discovery, who will announce it to all matching subscribers. After this point, the service will be forwarded requests addressed to it.

7.1.3. Step 3- Confirm Service Discovery registration

Service Discovery confirms that the service has been registered and announced.

7.1.4. Step 4 - Confirm Service Edge registration

Service Edge replies to Media Server service that the registration was successful, and that traffic is to be expected.

7.2. Certificates

Certificates are, in the RVI context, cryptographically signed JSON structures that prove the authenticity and authorization of one node to another.

Certificates are exchanged, peer-to-peer, between two nodes, and references to those certificates are included to requests sent from one node to another.

An example of a certificate is given below (with non JSON-compliant comments added for clarity):

```
{
  "node_certificate": {
    // Service name prefixes that this node is authorized to
    // process requests for.
    "sources": [ "jaguarlandrover.com/cloud/media_server" ],

    // Services that can be accessed by the source service.
    “destinations": [
      "rpc:jaguarlandrover.com/vin/+/services/media_player"
    ],

    // Public key for source.
    // Used to validate signature of requests, etc.
    "public_key": {
      "algorithm": "....", // Of sending node.
      "key": "..."
    },
    // Period during which certificate is valid. UTC
    "validity": {
      "start": 1401918299,
      "stop":  1402000000
    },
    // A system wide unique id for the certificate
    "id": b674546e-76ae-4204-b551-3f850fbffb4b
    // UTC timestamp of when the certificate was created.
    “create_timestamp”: 1403825201
  },
  // Signed by provisioning server.
  // All nodes have provisioning server's public key.
  // Signature covers all data in claims element.
  "signature": {
   "algorithm": "...",
   "signature": "..."
  }
}
```

Figure 6 – Certificate Example.

The following elements are included:

1. Sources
   A list of service name prefixes that the sending node is authorized to process requests for. This list will be installed in Service Edge of certificate-receiving nodes to be matched against requests that are to be sent out. If there is a match, the current network address of the certificate-issuing

node is looked up (see chapter “Step 5 [Vehicle] - Resolve network address”)
Service Edge assigns a node-unique transaction id that ties the request to the originating Media Player. This id will follow the transaction throughout its travel to its targeted service.
Service Edge then forwards the original request to Schedule in order to schedule it for transmission to the destination Cloud node.”), and the request is sent to it.

2. **Destinations**
   The destination services which the certificate authorizes the source service to access. Specified as one or more service name prefixes. During request authorization, the request is pattern and prefix matched against the destination fields.

3. **Public key**
   The public key of the source service, used to validate the signature of incoming authorizations and requests originating from the certificate-owning node.

4. **Validity period**
   The start and stop dates, specified as UTC, within which the certificate is valid.

5. **ID**
   System-wide unique ID for the certificate. Higher value is newer.

6. **Creation timestamp**
   Specifies the time, in UTC, when the certificate was created.

7. **Signature**
   Certificate signature, generated by the provisioning server’s private key. All nodes have the provisioning server’s public key setup as a step in the installation process.

See chapter “Node and Service Provisioning” for details on how certificates are exchanged between nodes.

7.3. Service Discovery

Service discovery is the process in where the availability of services is announced between two or more nodes. In order to fulfill the decentralized service discovery, sparse connectivity, and zero service configuration requirements stated in “

Requirements and Objectives”, a somewhat different strategy was chosen for this process. This strategy breaks down into the following two tenets.

1. **Peer to Peer Service Announcements**
   Since two nodes exchanging service availability information may not have an internet connection to a backend server, the exchange must be purely P2P without relying on a trusted third party for validation.

2. **Pairing-based Service Discovery**
   While traditional service discovery mechanisms are often broadcast information over a local network, the sparse connectivity environment of RVI makes that impossible. Instead two nodes exchange information on an opportunistic basis as soon as they see each other. A node keeps track of all other nodes it has ever seen together with the services that were available on each of these nodes when they were last seen. When two nodes see each other again, they exchange updated service information.

A backend server/cloud node will see most of the other nodes as they connect in to the server and will, over time, form a complete picture of all other nodes (vehicles, mobile phones and other devices) and their available services. Services connected directly to the backend node will, through its Service Edge, have access to all other nodes and their services, as is shown below:

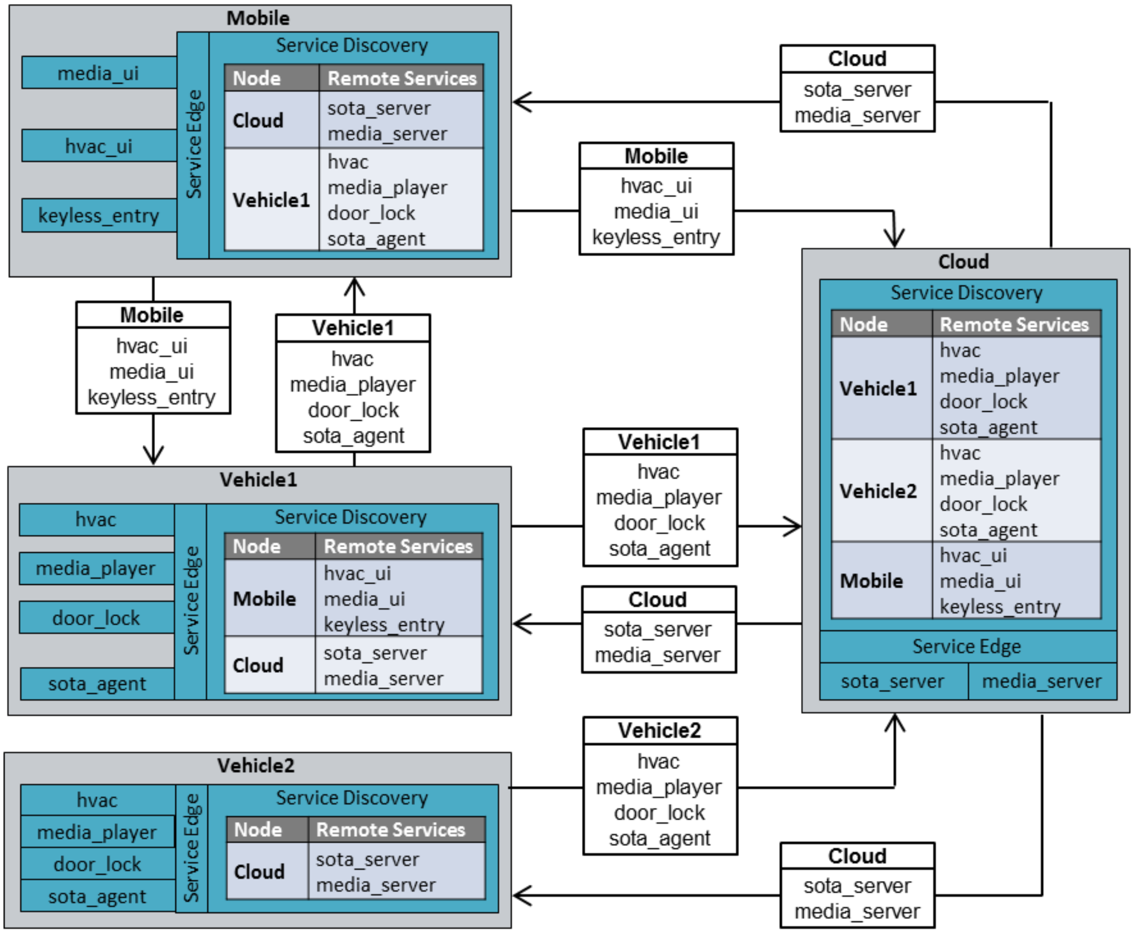


Figure 7 - Service Discovery mechanics.

In the case above, Mobile and Vehicle1 exchange services directly with each other on sight, as well as with Cloud, while Vehicle2 only exchanges services with Cloud.

The end result is that Mobile, having access to Vehicle1's services can control its media player, HVAC, and door locks, whole at the same time having access to Cloud's services. Meanwhile Vehicle2, having never seen Mobile, can only access Cloud's services.

The Service Discovery process between two nodes is described below.

7.4. Node Detection

When two or more nodes establish a communication channel, they need to detect the presence of each other before they can execute the Service Discovery process.
The exact means of how this is done is wholly dependent on the mechanics of the underlying network. There are, however, a number of best practices that can be employed for standard network models, described in the following chapters.

7.4.1. Vehicle connection to well-known server

When a vehicle's Data Link is instructed to connect to a backend server, identified by a pre-provisioned URL or IP address, the Data Link can ping the server once the communication channel is up.
The ping, which is a TCP/IP connection, does not have to transmit any information and can immediately be disconnected. The receiving server will ask the operating system for the peer address of the TCP connection to determine that there is a node at a given address that wants to execute Service Discovery.

If the protocol used to drive the Service Discovery process between two Data Links is connection based, the TCP connection can be used in the subsequent authorization and announce steps.

7.4.2. WiFi network connections

When a Data Link connects to a WiFi, LAN or CAN network, there may be a number of other nodes available to execute the Service Discovery process toward.

In these cases, each connected Data Link can send out a UDP/IP broadcast or multicast packet to a well-known address and port that all Data Links listen to. The payload of the packet is irrelevant since the receiving Data Link can look at the peer address of the packet to retrieve the IP address of the remote Node, and then initiate the Service Discovery Process against that Node.

7.4.3. P2P connections (Bluetooth, serial)

In a strict P2P connection, where two Data Links are in direct connection with other, the Service Discovery process can be initiated directly toward the remote end.

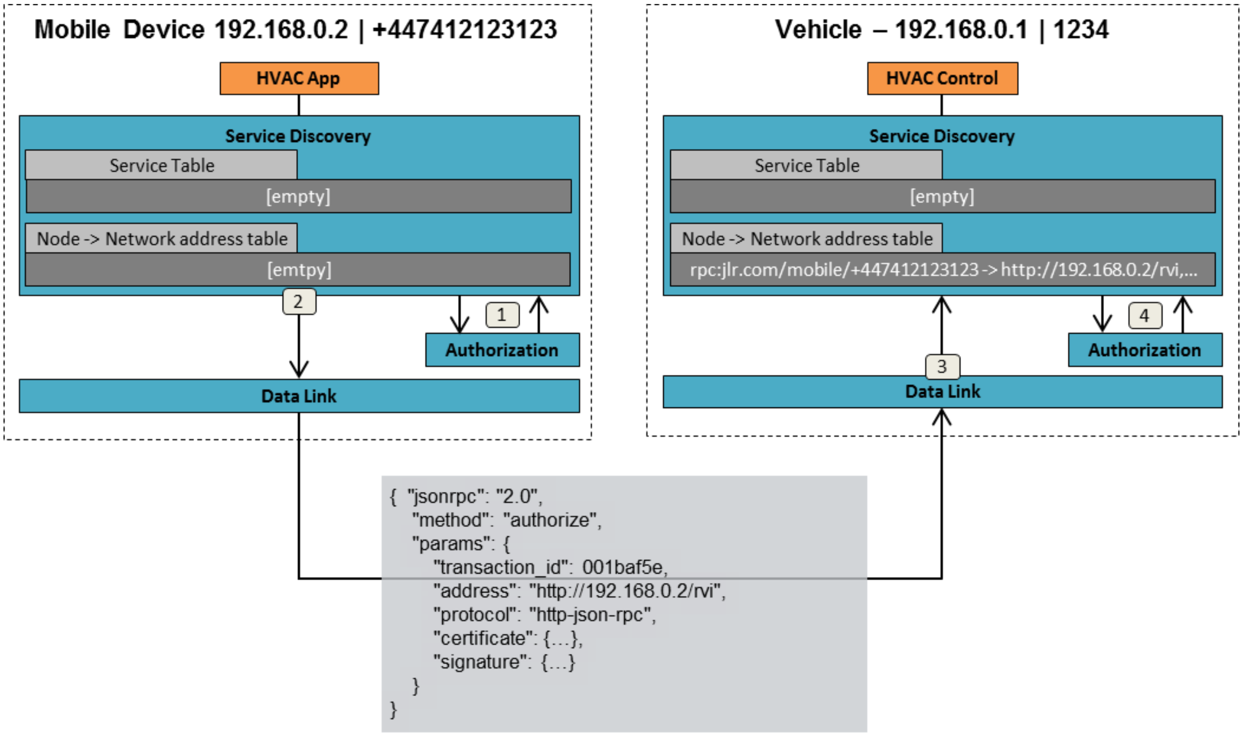
7.4.4. SMS

SMS communication between a vehicle and a central backend server can be handled as a connection to a well-known server described in chapter “Vehicle connection to well-known server”.

SMS communication between two nodes requires the Data Link to know the target’s MSISDN (phone number) since all mobile terminals are addressable at all times. There is no way for an SMS-connected Node to determine if another SMS-connected node is currently online or not. Instead the sending Node initiates its Service Disocvery process by submitting an SMS to the network, and then time out of there is no reply from the target node.

7.5. Authorization

When two nodes see each other, either for the first time, or on a recurring connection, they start by sending authorization information to the counterpart. In order to maintain the pairing paradigm, each node will send an Authorization package to each other node it sees.



Mobile Device 192.168.0.2 | +447412123123
HVAC App
Service Discovery
Service Table
[empty]
Node -> Network address table
[emtpy]
2
1
Authorization
Data Link
Vehicle – 192.168.0.1 | 1234
HVAC Control
Service Discovery
Service Table
[empty]
Node -> Network address table
rpc:jlr.com/mobile/+447412123123 -> http://192.168.0.2/rvi,...
4
Authorization
3
Data Link

```
{ "jsonrpc": "2.0",
   "method": "authorize",
   "params": {
      "transaction_id": 001baf5e,
      "address": "http://192.168.0.2/rvi",
      "protocol": "http-json-rpc",
      "certificate": {…},
      "signature": {…}
   }
}
```

Figure 8 – Mobile to Vehicle authorization.

In this case, the mobile device authenticates itself with the following information:

- **Network Address**
  Gives the network address where the node can be reached. Please note that this address may be transient and can change the next time two nodes see each other.

- **Protocol**
  Specifies the protocol to use when communicating with the node at the network address. In this case it is JSON-RPC over HTTP.

- **Certificate**
  A certificate, signed by the provisioning server, contains identities, public keys, and access rights of the sending node. The certificate also stores one or more service name prefixes that the sending Node is authorized to receive requests for.

- **Signature**
  Contains the node signature of the address and protocol elements in the authorization package. The signature is generated by the Node's private key and can be verified by the public key contained in the Node Certificate.

- **Transaction ID**
  A monotonically incremented ID. The received will store the last received ID from the node using the given certificate, and reject any new transactions with an ID that is lower than or equal to the stored ID. This stops replay attacks.

The steps of authorization process are as follow:

1. **Request certificate from local Authorization**
   The certificate for the given peer-to-peer pair is retrieved from Authorization.
   FIXME: How does Authorization know which P2P connection is up?

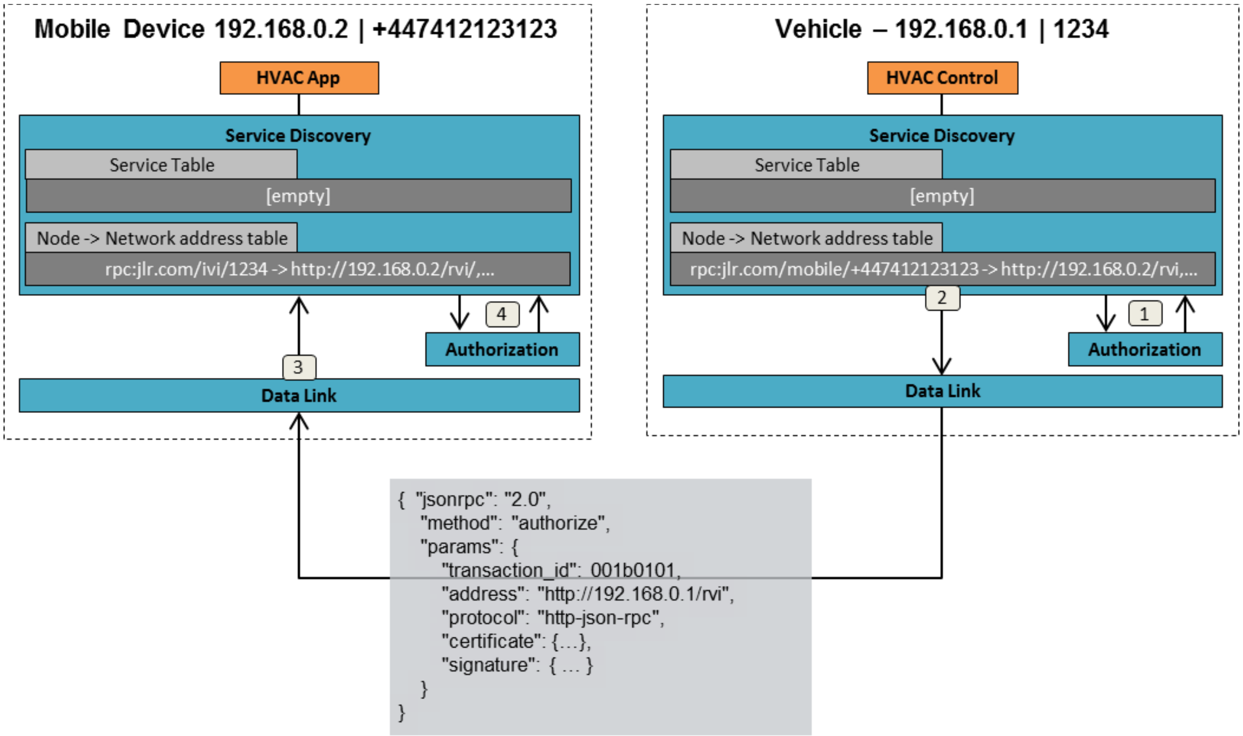
2. **Transmit authenticate package to remote node**
   This is done by having Service Discovery interact directly with Data Link, without involving Protocol.

3. **Forward incoming package to Service Discovery [Future v0.3]**
   The receiving Vehicle Data Link forwards the package to Service Discovery

4. **Validate authenticate package [Future v0.3]**
   Service Discovery on Vehicle forwards the package to Authorize, which will validate the certificate toward the provision server's public key and check that the provided service name prefix matches those in the certificate.

Once completed, the service name prefix(es) are added together with the network address in the address table of Service Discovery.

The Vehicle device sends a similar authorization package to the mobile device, as is shown below.


Mobile Device 192.168.0.2 | +447412123123
HVAC App
Service Discovery
Service Table
[empty]
Node -> Network address table
rpc:jlr.com/ivi/1234 -> http://192.168.0.2/rvi/,...
4
Authorization
3
Data Link
Vehicle – 192.168.0.1 | 1234
HVAC Control
Service Discovery
Service Table
[empty]
Node -> Network address table
rpc:jlr.com/mobile/+447412123123 -> http://192.168.0.2/rvi,...
2
1
Authorization
Data Link

```
{ "jsonrpc": "2.0",
   "method": "authorize",
   "params": {
      "transaction_id": 001b0101,
      "address": "http://192.168.0.1/rvi",
      "protocol": "http-json-rpc",
      "certificate": {…},
      "signature": { … }
   }
}
```

Figure 9 – Vehicle to mobile authorization.

Once authorization packages have been exchanged between two nodes, they know their counterpart’s address, and which service requests that should be forwarded to it.

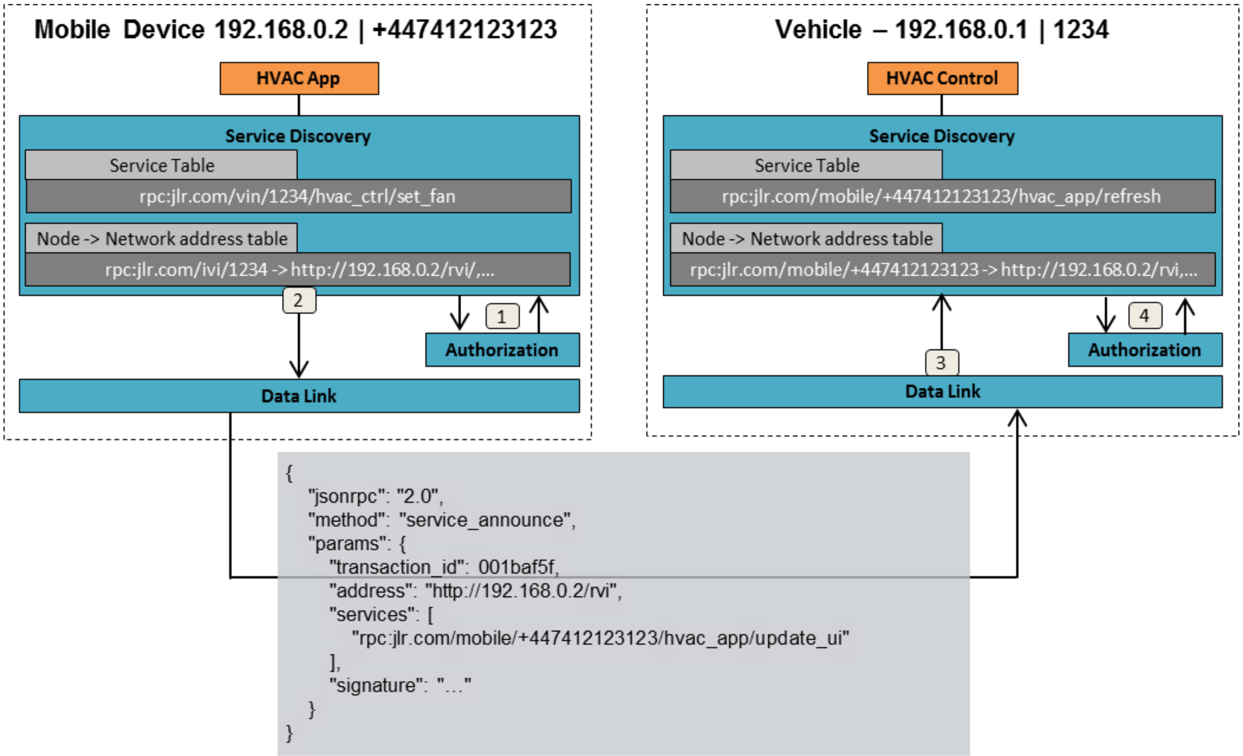
The prefix element will be matched against the “source” element of the certificate to ensure that a node does not try to serve traffic it is not authorized to handle.

The authorization is valid on the receiving node until the same certificate is used in another authorization package, or the accompanying certificate is revoked or expired.

7.6. Service Announcement

When a node has sent its authorization to another node, the received uses the sending node's certificate to determine which services the receiving node should announce to the sender. By pattern matching all available services against a received certificate's destination element, it can filter out those services eligible for announcement.

The announcement is carried out as follows:


Mobile Device 192.168.0.2 | +447412123123
HVAC App
Service Discovery
Service Table
rpc:jlr.com/vin/1234/hvac_ctrl/set_fan
Node -> Network address table
rpc:jlr.com/ivi/1234 -> http://192.168.0.2/rvi/,...
2
1
Authorization
Data Link
Vehicle – 192.168.0.1 | 1234
HVAC Control
Service Discovery
Service Table
rpc:jlr.com/mobile/+447412123123/hvac_app/refresh
Node -> Network address table
rpc:jlr.com/mobile/+447412123123 -> http://192.168.0.2/rvi,...
4
3
Authorization
Data Link

```
{
   "jsonrpc": "2.0",
   "method": "service_announce",
   "params": {
      "transaction_id": 001baf5f,
      "address": "http://192.168.0.2/rvi",
      "services": [
         "rpc:jlr.com/mobile/+447412123123/hvac_app/update_ui"
      ],
      "signature": "..."
   }
}
```

Figure 10 – Mobile Device service announcement.

The "service announce" package contains one or more services that the receiving node can invoke on the sending node.

The steps of announcement process are as follow:

1. **Sign announcement**
   The service announcement is sent to Authorization, which will sign the package using the private key of the Mobile node.

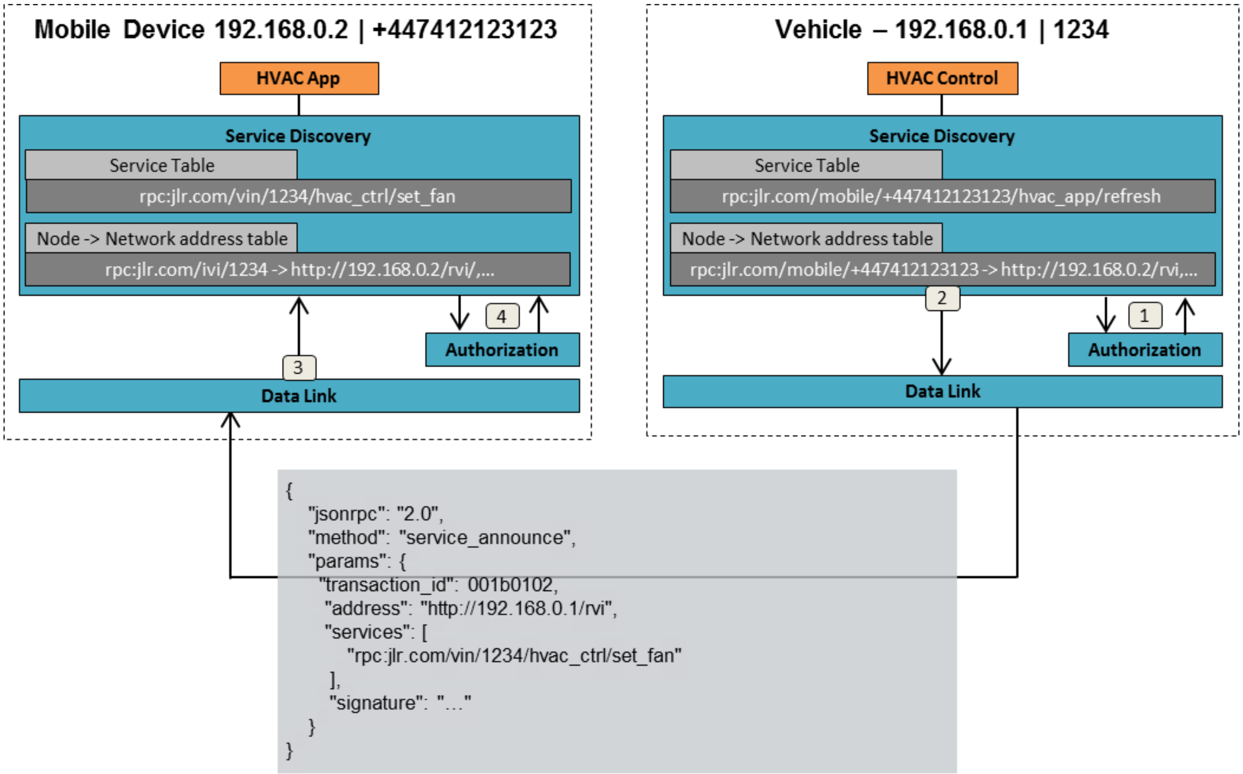
2. **Transmit service announcement package Vehicle**
   The package contains all services that the Mobile can invoke on Vehicle.

3. **Forward service announcement to Service Discovery**

4. **Validate package**
   The received package is forwarded to Authorize to match the signature against the certificate received in the authorization process.

Once the process has completed, all services announced by the mobile device are stored in the service table of Service Discovery on the vehicle side.

The Vehicle goes through the same process of service announcement:



Mobile Device 192.168.0.2 | +447412123123
HVAC App
Service Discovery
Service Table
rpc:jlr.com/vin/1234/hvac_ctrl/set_fan
Node -> Network address table
rpc:jlr.com/ivi/1234 -> http://192.168.0.2/rvi/,...
4
Authorization
3
Data Link
Vehicle – 192.168.0.1 | 1234
HVAC Control
Service Discovery
Service Table
rpc:jlr.com/mobile/+447412123123/hvac_app/refresh
Node -> Network address table
rpc:jlr.com/mobile/+447412123123 -> http://192.168.0.2/rvi,...
2
1
Authorization
Data Link

```
{
    "jsonrpc": "2.0",
    "method": "service_announce",
    "params": {
     "transaction_id": 001b0102,
      "address": "http://192.168.0.1/rvi",
      "services": [
          "rpc:jlr.com/vin/1234/hvac_ctrl/set_fan"
       ],
       "signature": "…"
    }
}
```

Figure 11 - Vehicle service announcement.

The service announcement is valid with the receiver until a new authenticate package is received using the same certificate that validates the service announcement's certificate, or the certificate is revoked or expires.

8. Request Routing

A request sent by one service to another goes through several steps. The following chapters describe a use case where a vehicle-based media player requests a song to be played from a cloud-based media server. The involved components are shown below.

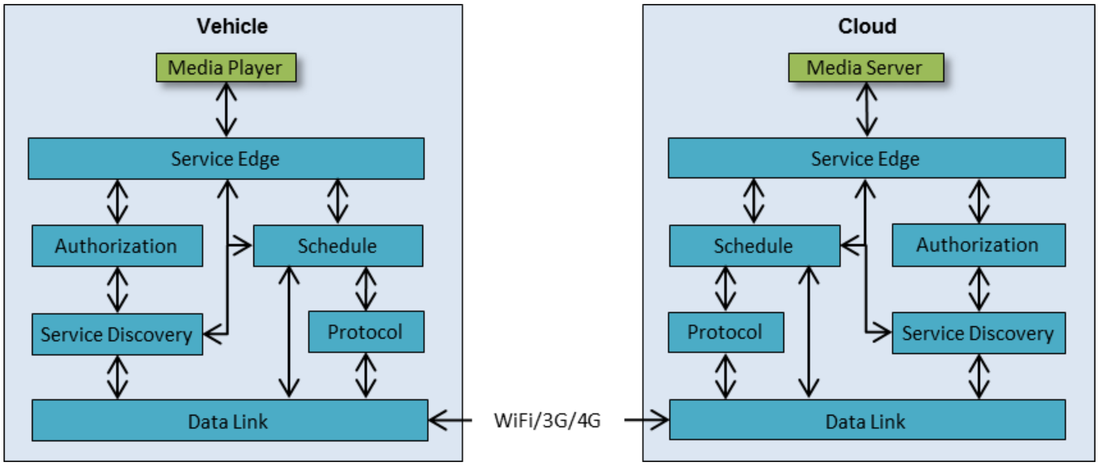


Figure 12 - Request Routing setup.

At the start of the request routing, the following preconditions are met:

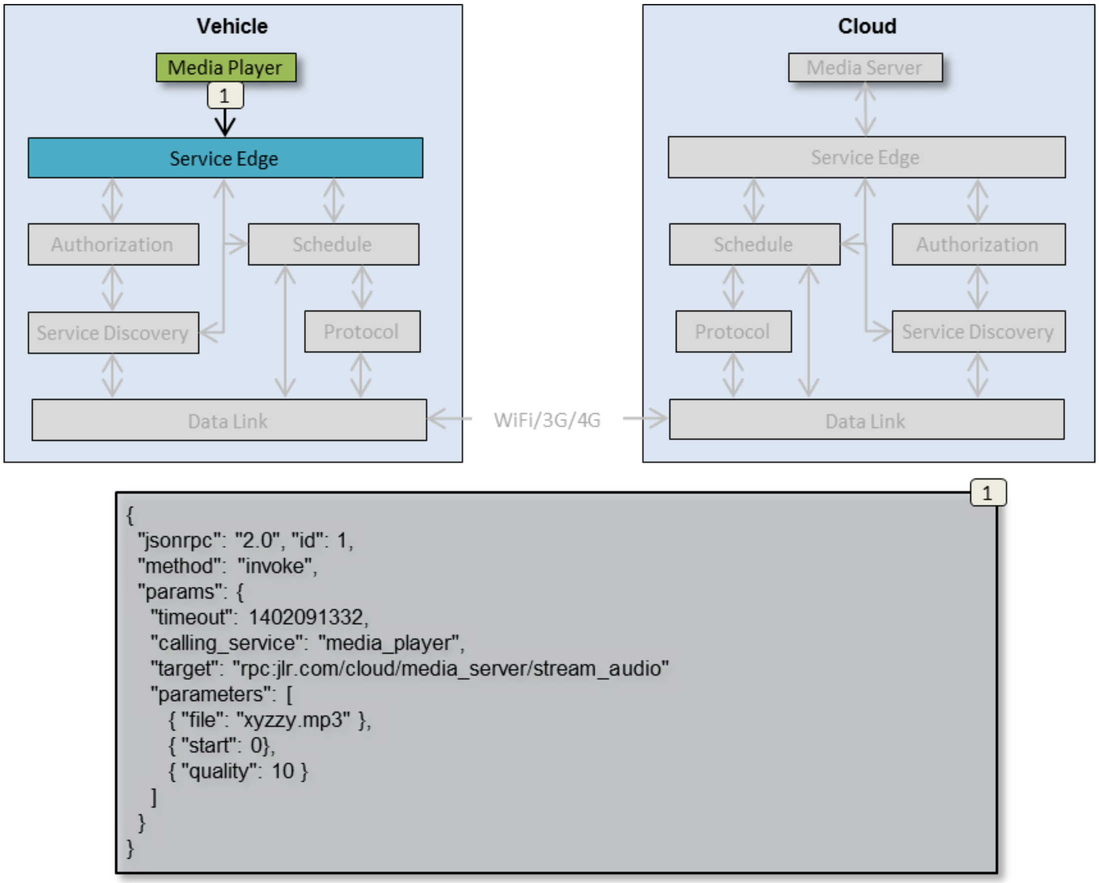
1. **Services are registered**
   The Media Player and Media Server are both registered with their Service Edge, as described in chapter “Service Registration”

2. **No certificates**
   The Vehicle does not have a certificate from the Cloud, and vice versa.

3. **No connection**
   The mobile connection between the cloud and the vehicle is yet to be setup.

8.1. Step 1 [Vehicle] - Submit request to Service Edge

The transaction is initiated when the Media Player on the vehicle sends a play request, addressed to the Media Server in the Cloud, to its local Service Edge.



```
{
  "jsonrpc": "2.0", "id": 1,
  "method": "invoke",
  "params": {
    "timeout": 1402091332,
    "calling_service": "media_player",
    "target": "rpc:jlr.com/cloud/media_server/stream_audio"
    "parameters": [
      { "file": "xyzzy.mp3" },
      { "start": 0},
      { "quality": 10 }
    ]
  }
}
```

Figure 13 - Initial request submit.

The request contains the following elements

1. **method**
   Always “invoke”. The actual request mechanism, rpc, message, subscribe, etc, is specified in “target”.
   NOTE: In v 0.2, the method is always “message”. This will be fixed in v0.3.

2. **timeout [Future v0.4]**
   Specifies the time stamp, as UTC, by which the reply for this request has to be returned to the

Media Player in order not to trigger a timeout error.

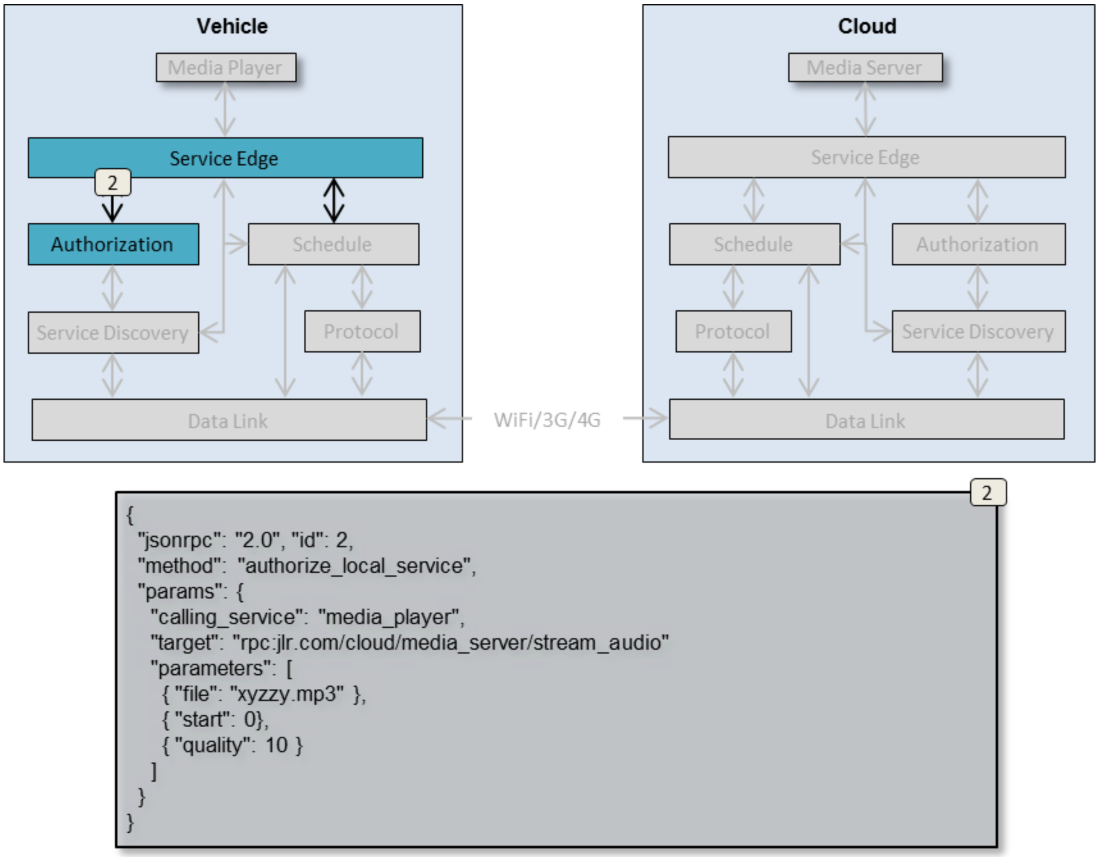
3. **calling_service**
   Specifies the service that is sending the request, in this case the media player. The name is used as a key by the local Service Edge when validating credentials.

   target
   Specifies where the request is to be sent. This node address will be resolved to a network address by Service Discovery. The service will have built-in knowledge of available remote services it wants to access. Since the service name tree encompasses all services on all nodes, a single service name entry is enough to identify a given service, no matter where it is running.
   Note: in v0.2, the `type:` prefix is not supported and will always be set to message.

4. **parameters**
   Provides additional, arbitrary data to deliver to the target service.

8.2. Step 2 [Vehicle] - Authorize service request

Service Edge forwards the request it received from the Media Player to Authorization to be validated.



```
{
  "jsonrpc": "2.0", "id": 2,
  "method": "authorize_local_service",
  "params": {
    "calling_service": "media_player",
    "target": "rpc:jlr.com/cloud/media_server/stream_audio"
    "parameters": [
      { "file": "xyzzy.mp3" },
      { "start": 0},
      { "quality": 10 }
    ]
  }
}
```

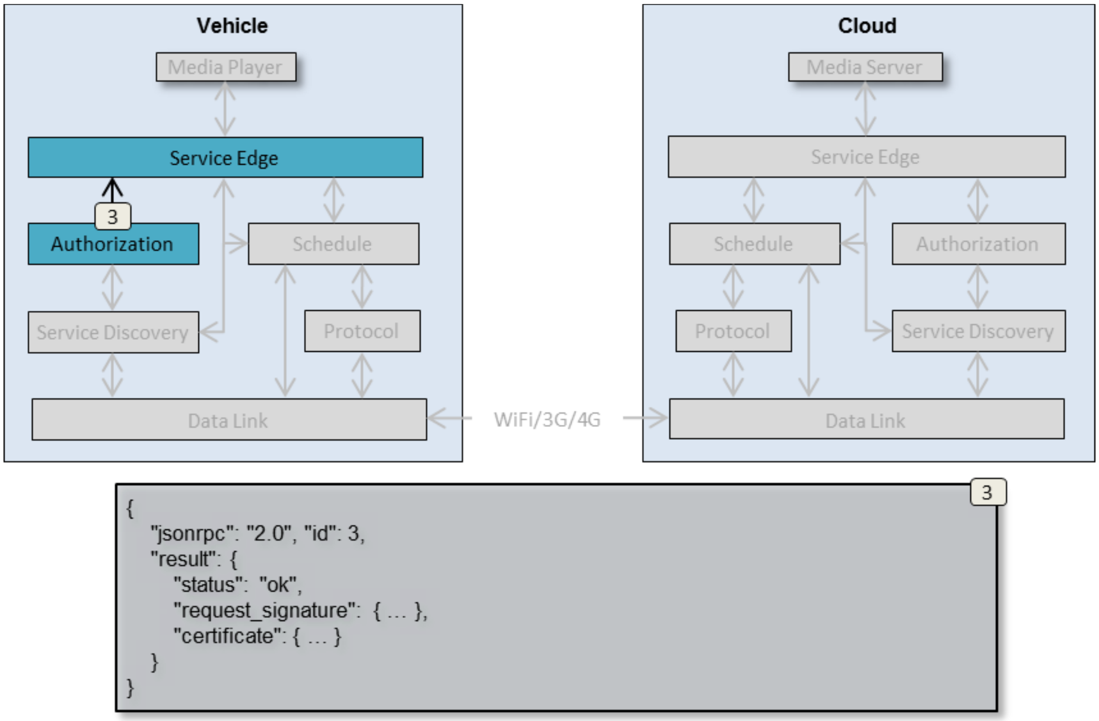
Figure 14 - Validate service request.

The request is identical to the one received from Media Player, save that the method has been replaced with authorize_local_service. Since this is a validation of a locally connected service, the service name (specified in target) can be used as a key to lookup and validate the transaction.

The “target” element is pattern matched against the “destinations”elements of all certificates the Vehicle node have provisioned. If there is a match, the given certificate will be sent with the request to its targeted remote (Cloud) node to prove that Vehicle has the right to invoke targeted service on it.

8.3. Step 3 [Vehicle] - Auhtorize service reply

If Authorization successfully validates the request, it will reply with the certificate to present to the remote node.



```
{
   "jsonrpc": "2.0", "id": 3,
   "result": {
      "status":  "ok",
      "request_signature":  { … },
      "certificate": { … }
   }
}
```

Figure 15 - Validate service reply.

The certificate returned by Authorization is either pre-installed, or distributed using a dedicated certificate transmission service from a provisioning service to the node.

The authorization reply contains the following elements.

1. **status**
   Contains the result of the authorization request.

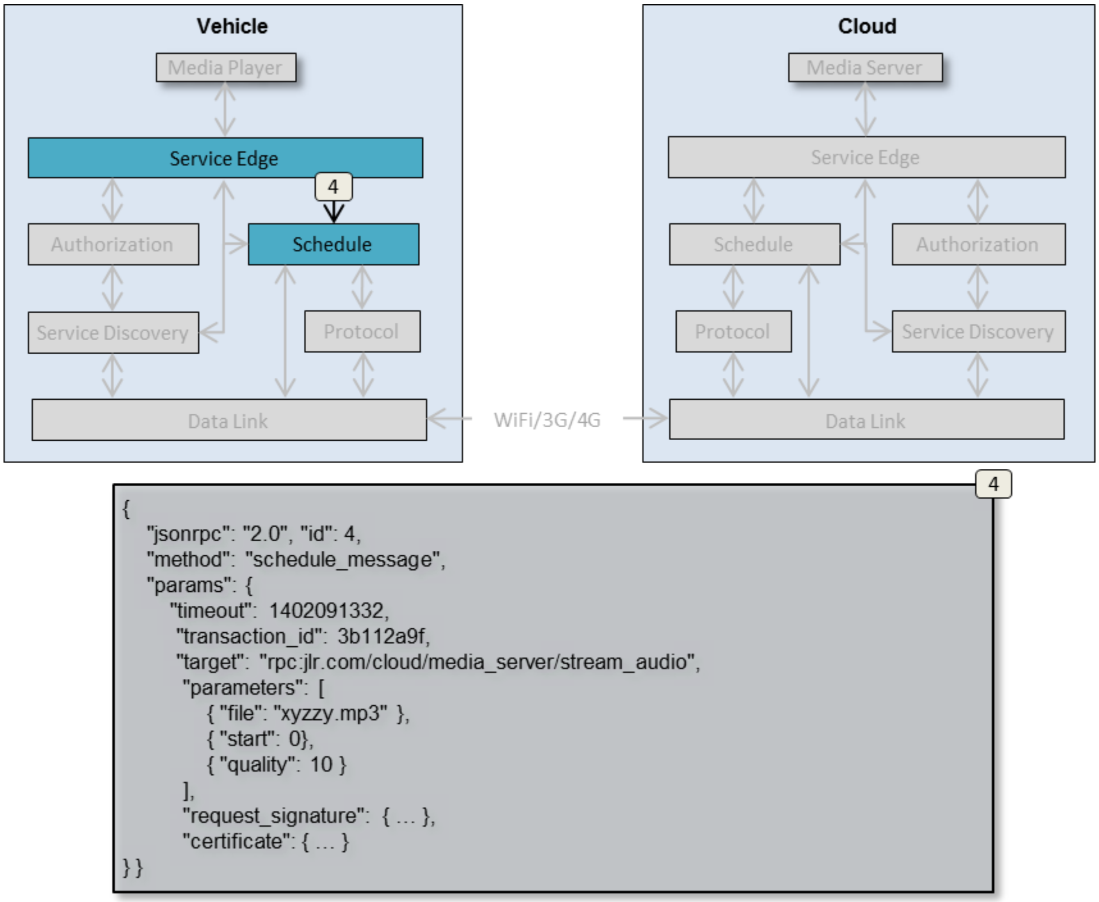
2. **request_signature**
   The signature for the authorized request. This signature, generated by Authorization's private key, is to accompany the request through all steps to its targeted destination.

3. **certificate**
   Contains the certificate, with the public key used to validate request_signature, that can be used by the target node to validate the request. The certificate is generated by the Provisioning Server and is signed by its private key.

8.4. Step 4 [Vehicle] - Schedule request

Service Edge assigns a node-unique transaction id that ties the request to the originating Media Player. This id will follow the transaction throughout its travel to its targeted service.

Service Edge then forwards the original request to Schedule in order to schedule it for transmission to the destination Cloud node.



```
{
    "jsonrpc": "2.0", "id": 4,
    "method": "schedule_message",
    "params": {
       "timeout": 1402091332,
        "transaction_id": 3b112a9f,
        "target": "rpc:jlr.com/cloud/media_server/stream_audio",
         "parameters": [
            { "file": "xyzzy.mp3" },
            { "start": 0},
            { "quality": 10 }
         ],
         "request_signature": { … },
         "certificate": { … }
}}
```

Figure 16 – Schedule request for transmission.

The request sent over to Schedule has the following elements:

1. **transaction_id**
   A unique transaction id that will follow the request to its destination, and accompany the reply on its way back. Vehicle Service Edge can map the transaction ID to the callback URL of the Media Player.

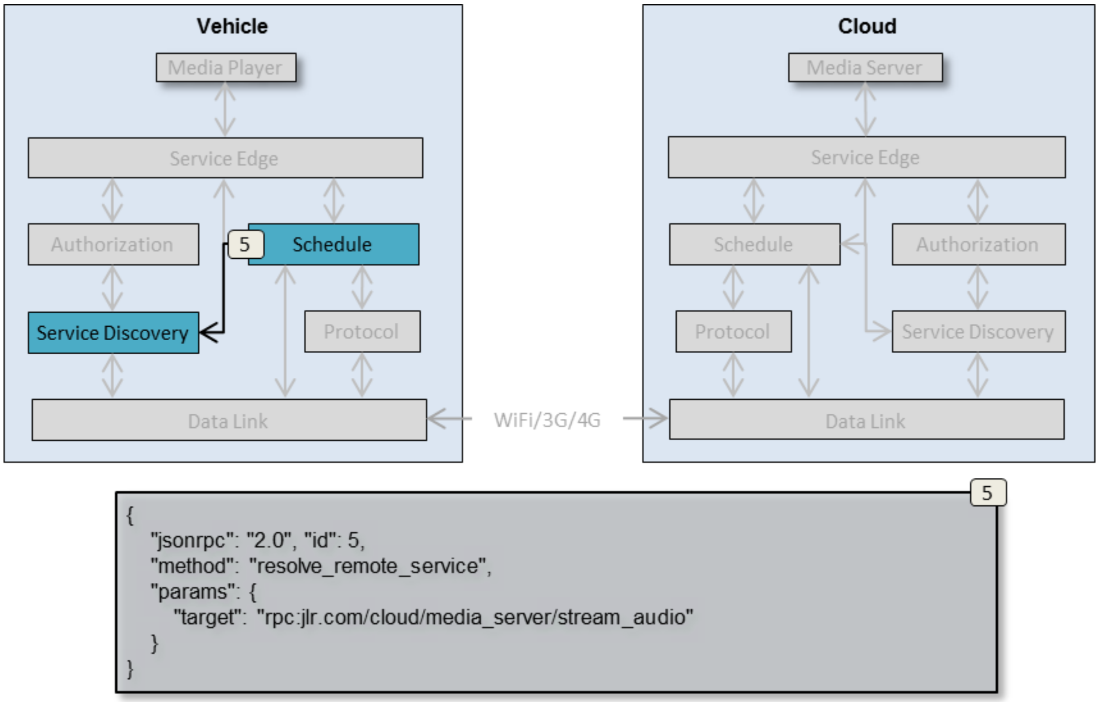
2. **target, network_address, arguments**
   Request routing and payload information carried over from previous steps.

3. **request_signature**
   The request signature, generated by Authorization, to prove that this request is created by the Vehicle node.

4. **certificate**
   The certificate, signed by the Provisioning Server, used to validate the signature, hence the request itself.

8.5. Step 5 [Vehicle] - Resolve network address

Schedule sends a name resolve request to Service Discovery in order to translate the service name, specified as the request's target, to a network address that can be handled by Data Link.



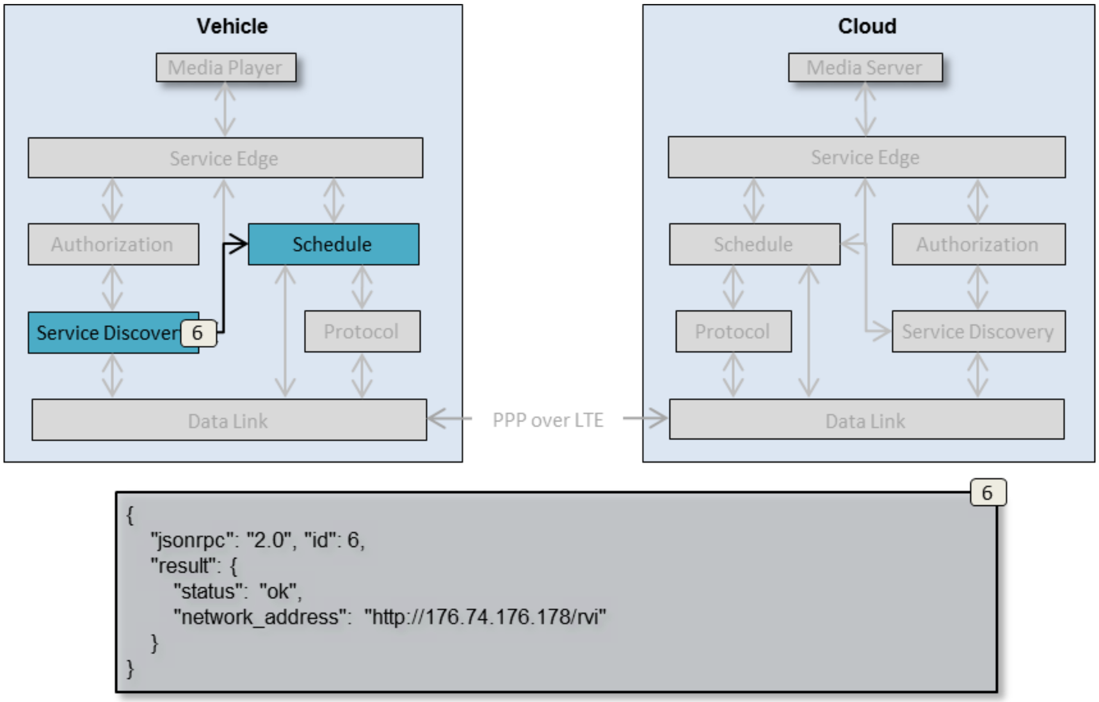
```
{
    "jsonrpc": "2.0", "id": 5,
    "method": "resolve_remote_service",
    "params": {
        "target": "rpc:jlr.com/cloud/media_server/stream_audio"
    }
}
```

Figure 17 - Address resolve request.

The single argument provided to Service Discovery is the service name for which a network address is needed.

8.6. Step 6 [Vehicle] - Return resolved network address

Service Discovery uses either install-time provisioned data or received service discovery announce data to pattern match and resolve the received node address to a network address, which is sent back to Schedule.

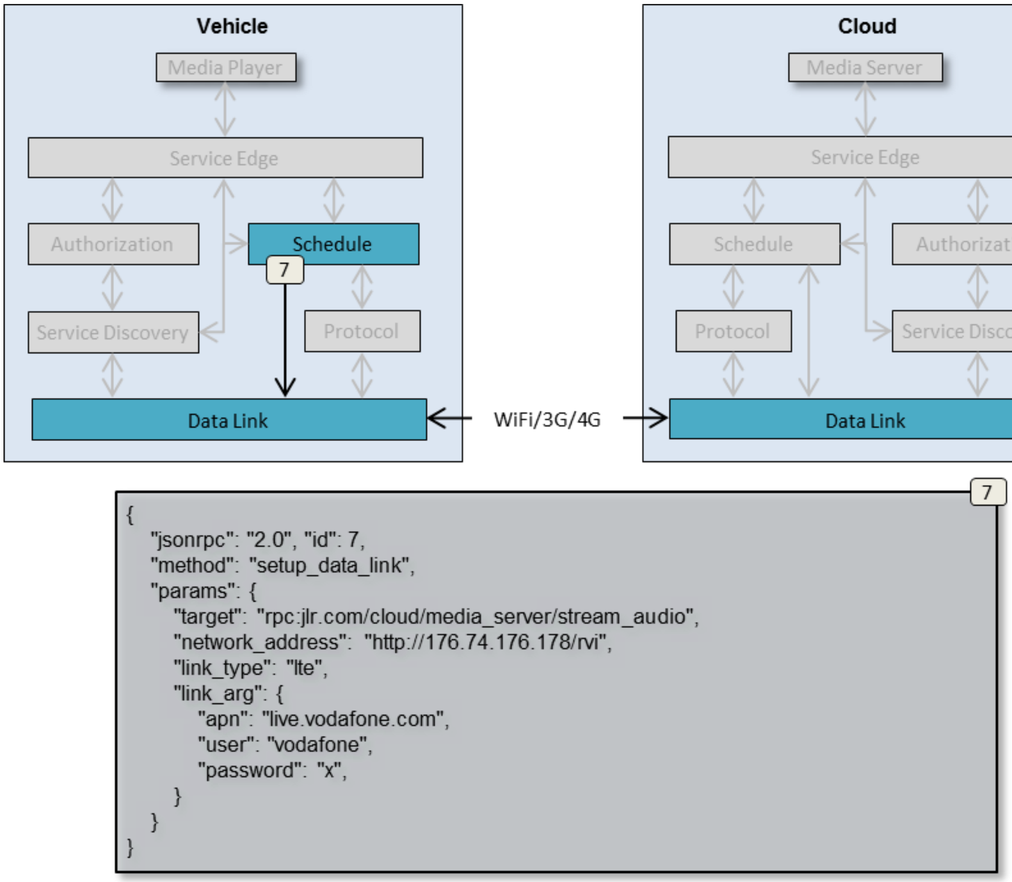


```
{
   "jsonrpc": "2.0", "id": 6,
   "result": {
      "status":  "ok",
      "network_address":  "http://176.74.176.178/rvi"
   }
}
```

Figure 18 – Address resolve reply.

8.7. Step 7 [Vehicle] - Setup Communication Channel

Schedule uses its internal database to determine the type of data link needed to transmit the request to its target node, and then requests Data Link to setup a connection to the network address resolved by Service Edge.



```
{
  "jsonrpc": "2.0", "id": 7,
  "method": "setup_data_link",
  "params": {
    "target": "rpc:jlr.com/cloud/media_server/stream_audio",
    "network_address": "http://176.74.176.178/rvi",
    "link_type": "lte",
    "link_arg": {
      "apn": "live.vodafone.com",
      "user": "vodafone",
      "password": "x",
    }
  }
}
```

Figure 19 - Data link setup request.

The request contains the following information:

1. **target**
   The original target specified by the originating Media Player service

2. **network_address**
   The network address resolved from the target by Service Discovery

3. **link_type**
   A symbolic name for the type of communication channel that Data Link is to establish. In this

case it is an Long Term Evolution 4G link (over ppp).

4. **link_arg**
   Additional data to pass on to Data Link and its handled for the given link type. For an LTE link an access point node, user name and password is needed.
   The link_arg element can also contain priority information, depending on how urgent Schedule deems the request to be.

Data Link uses the provided network_address, link_type and link_arg to setup a ppp link over LTE.

FIXME: Where do we get link_type and link_arg from? Locally provisioned in Data Link?

8.8. Step 8 [Vehicle] - Authorize and Announce

Vehicle and Cloud authorizes themselves and announces their services as described in chapter "Service Registration"

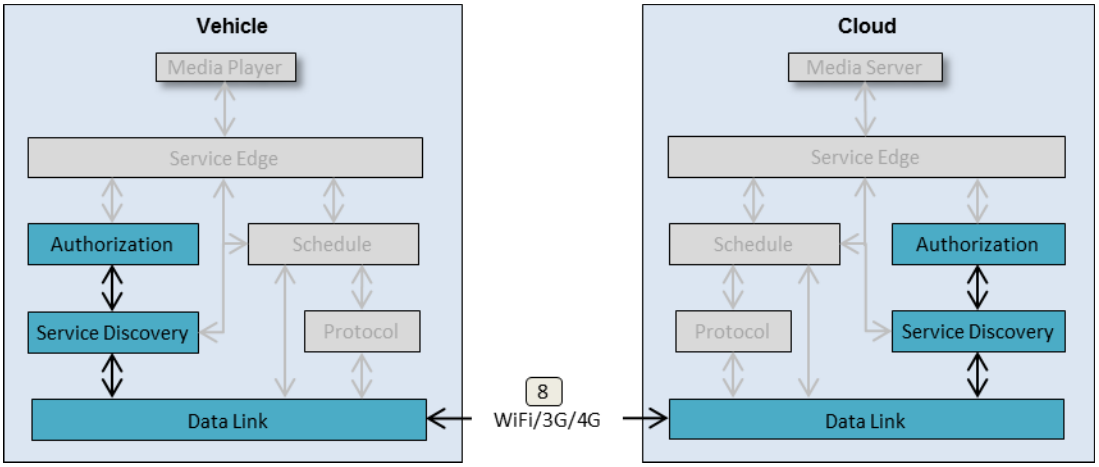
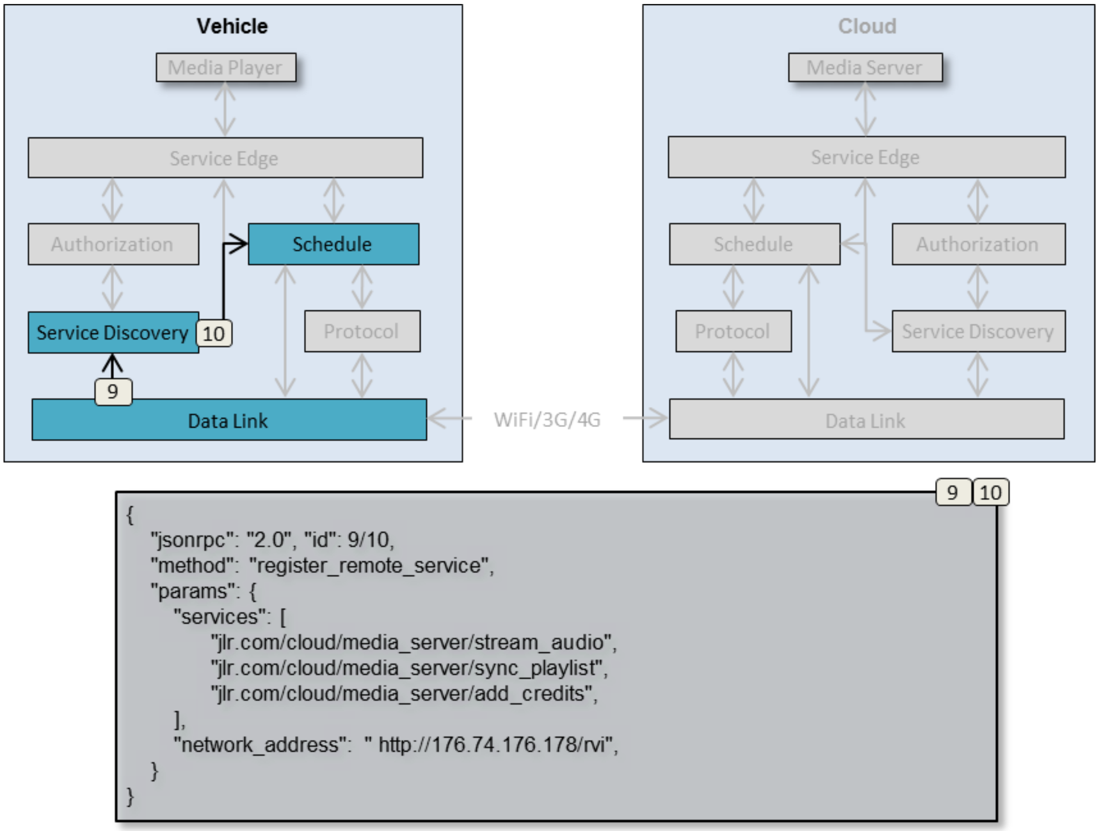


Figure 20 - Authorize and Announce.

Once the authorization and announcement process has completed, Vehicle and Cloud are aware of each other's available services.

8.9. Step 9/10 [Vehicle] – Report Service Availability

Once the Vehicle and the Cloud have authorized each other and announced their services, Data Link on Vehicle announces all newly available services in the Cloud to Service Discovery. Service Discovery forwards the information to Schedule.



```
{
    "jsonrpc": "2.0", "id": 9/10,
    "method": "register_remote_service",
    "params": {
       "services": [
            "jlr.com/cloud/media_server/stream_audio",
            "jlr.com/cloud/media_server/sync_playlist",
            "jlr.com/cloud/media_server/add_credits",
       ],
       "network_address": " http://176.74.176.178/rvi",
    }
}
```

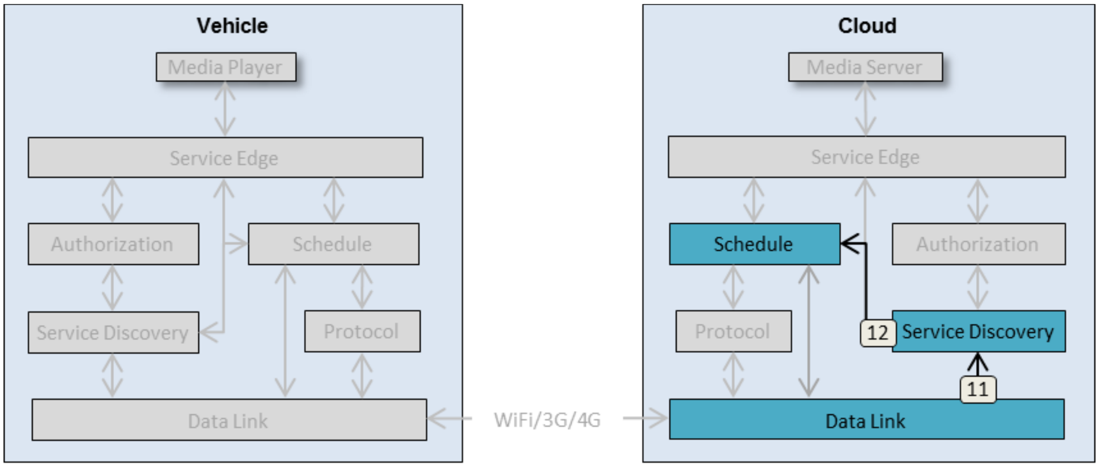
Figure 21 – Vehicle communication channel available.

The network address contains the address reported by the Cloud during its authorization phase described in chapter “Authorization”. This address can be used by Data Links throughout the network to contact the Cloud node.

The services element lists all services announced by the Cloud during discover phase described in chapter “Service Announcement”. Included in this list is the “`jlr.com/cloud/media_server/stream_audio`” service that is targeted by the request currently waiting in Schedule.

8.10. Step 10/11 [Cloud] - Report Service Availability

In a similar manner to step 9/10, Data Link on Cloud announces the services now available on the Vehicle.



```
{
   "jsonrpc": "2.0", "id": 11/12,
   "method": "register_remote_service",
   "params": {
      "services": [
         "rpc:jlr.com/vin/1234/media_player/play_audio",
         "rpc:jlr.com/vin/1234/media_player/show_banner_ad",
         "rpc:jlr.com/vin/1234/media_player/set_volume",
      ],
      "network_address":  " http://231.74.99.2/rvi",
   }
}
```

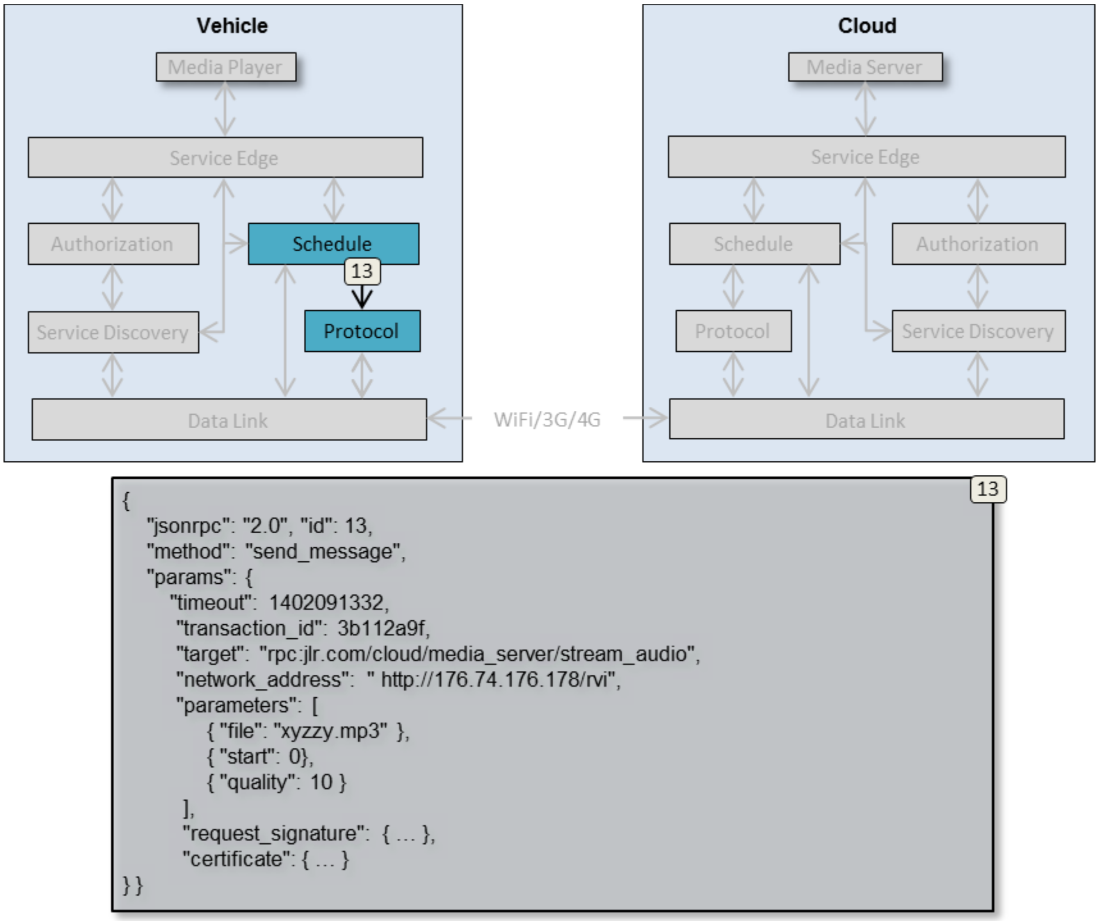
Figure 22 – Cloud communication channel available.

The network address contains the address reported by the Vehicle during its authorization phase described in chapter “Authorization”. Other nodes in the network, including that of the Cloud, can use this network address to access the Vehicle node.

The services elements lists all services announced by the Vehicle during discover phase described in chapter “Service Announcement”.

8.11. Step 13 [Vehicle] - Request encoding and transmission

When schedule receives a list of newly available services from Service Discovery in step 10, it will match that list against the target of all requests awaiting transmission in Schedule. Since one of the services in the list is “`jlr.com/cloud/media_server/stream_audio`”, the stored stream audio request is retrieved and is forwarded to Protocol.



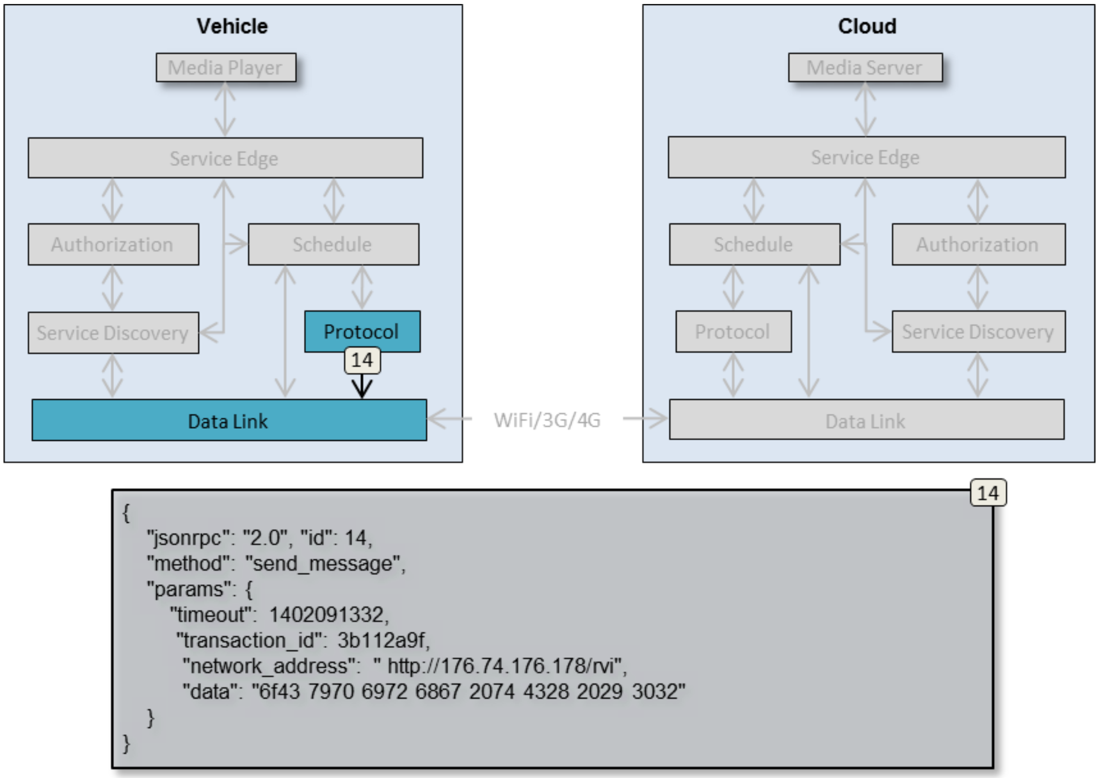
```
{
   "jsonrpc": "2.0", "id": 13,
   "method": "send_message",
   "params": {
      "timeout": 1402091332,
       "transaction_id": 3b112a9f,
       "target": "rpc:jlr.com/cloud/media_server/stream_audio",
       "network_address": " http://176.74.176.178/rvi",
       "parameters": [
           { "file": "xyzzy.mp3" },
           { "start": 0},
           { "quality": 10 }
        ],
        "request_signature": { ... },
        "certificate": { ... }
}}
```

Figure 23 - Prepare request for transmission.

The request forwarded to Protocol is identical to that received by Schedule from Service Edge in step 6.

8.12. Step 14 [Vehicle] - Transmit data

Protocol encodes the request to a data payload and sends Data Link for further transmission.



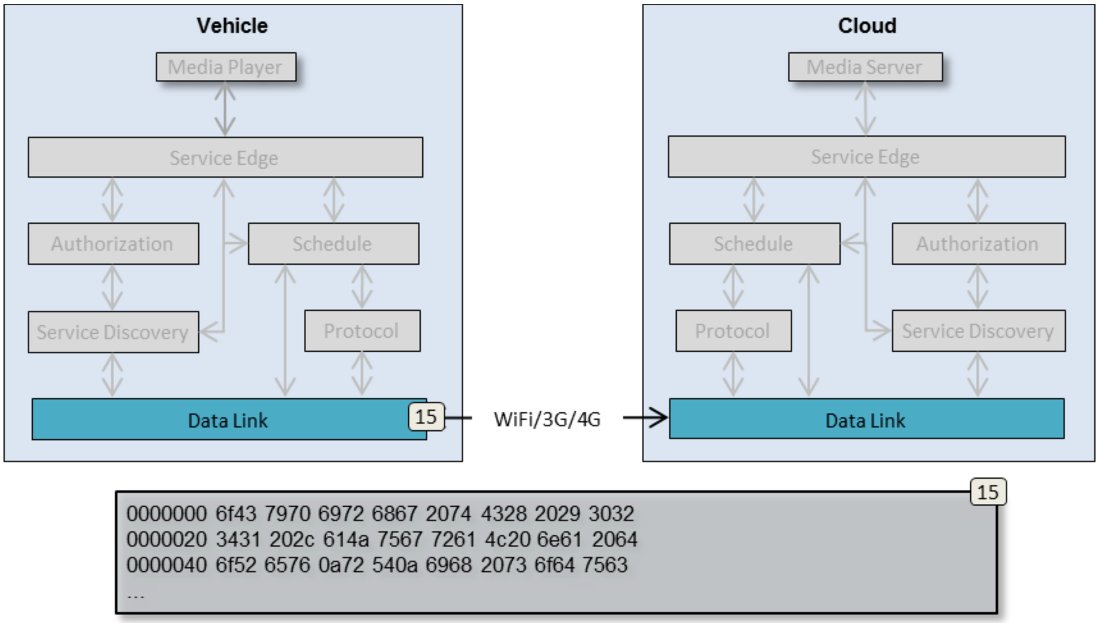
```
{
   "jsonrpc": "2.0", "id": 14,
   "method": "send_message",
   "params": {
      "timeout": 1402091332,
       "transaction_id": 3b112a9f,
        "network_address": " http://176.74.176.178/rvi",
        "data": "6f43 7970 6972 6867 2074 4328 2029 3032"
   }
}
```

Figure 24 – Forward encoded data for transmission.

The information provided to Data Link is the target address, and the encoded data.

8.13. Step 15 [Cloud] – Transmit data

Data Link transmits the encoded data to the network address of the Cloud node, where the local Data Link receives it.

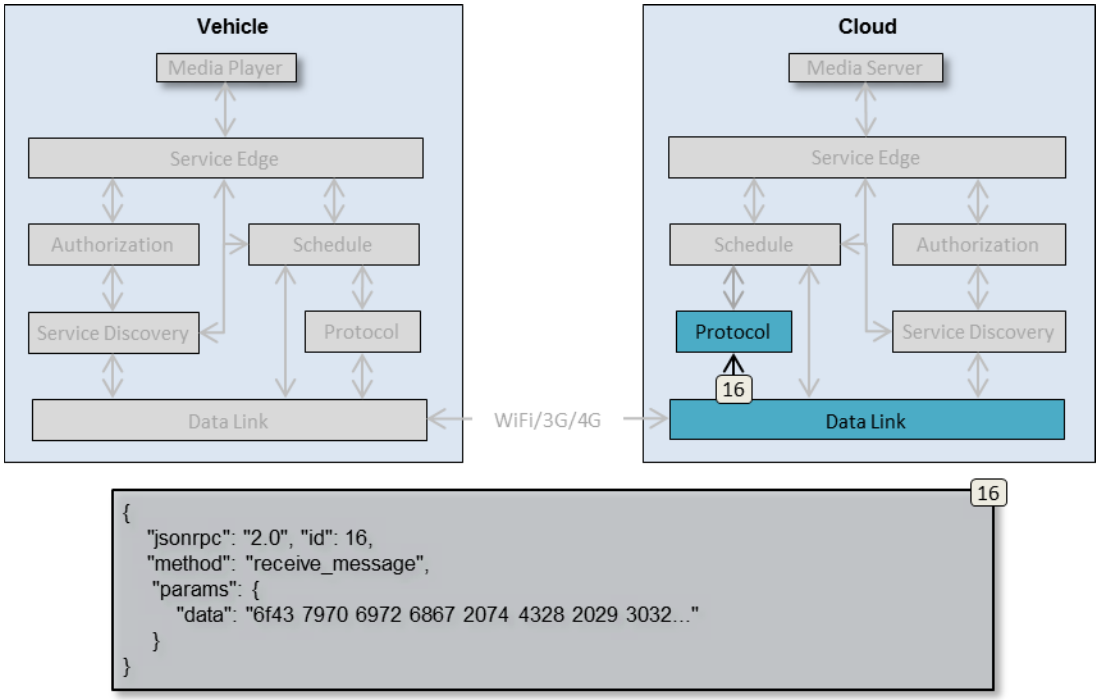


```
0000000 6f43 7970 6972 6867 2074 4328 2029 3032
0000020 3431 202c 614a 7567 7261 4c20 6e61 2064
0000040 6f52 6576 0a72 540a 6968 2073 6f64 7563
...
```

Figure 25 – Transmit Data.

8.14. Step 16 [Cloud] - Forward data to Protocol

Data Link forwards the raw data to Protocol in order to have the binary payload decoded and processed.



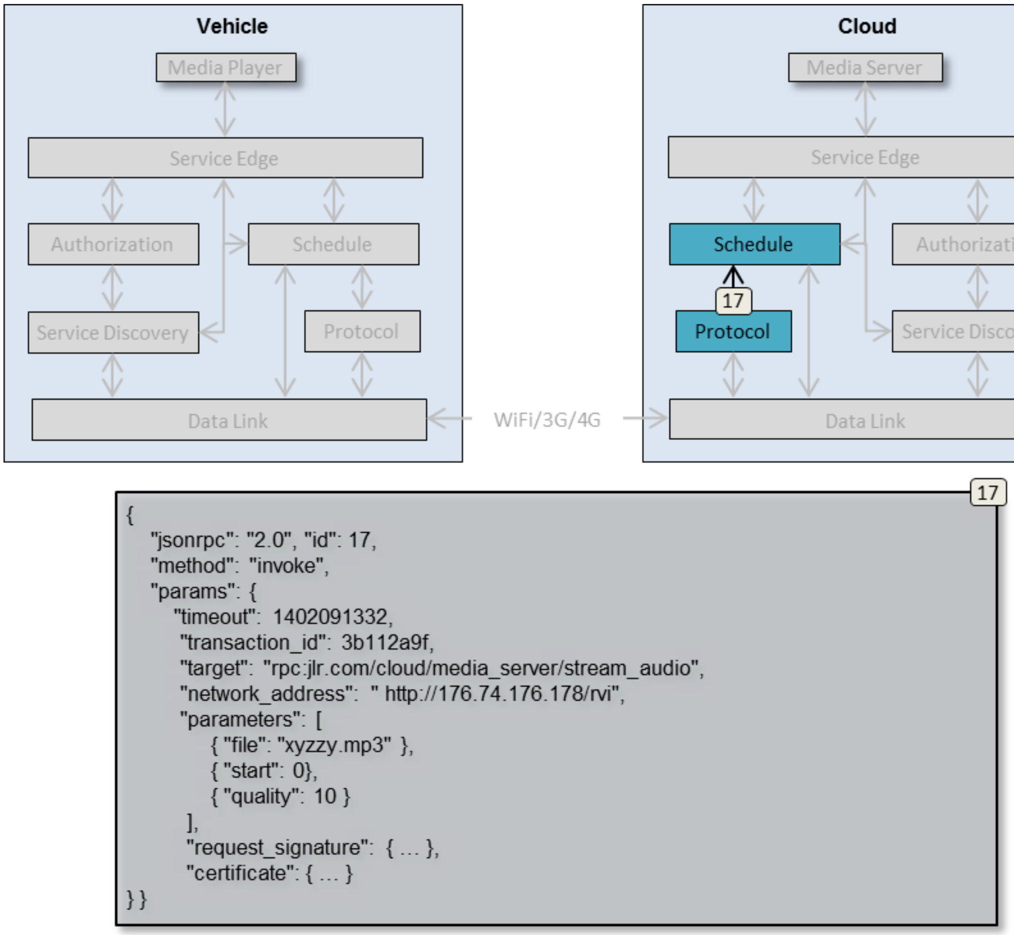
```
{
   "jsonrpc": "2.0", "id": 16,
   "method": "receive_message",
    "params": {
       "data": "6f43 7970 6972 6867 2074 4328 2029 3032..."
    }
}
```

Figure 26 – Forward binary data to Protocol.

8.15. Step 17 [Cloud] - Decode payload

Protocol in the Cloud node uses the received data payload to reconstruct the original request sent by Vehicle's Media Player. The reconstructed request is forwarded to Schedule.

NOTE: In ver 0.2, the decoded payload is forwarded directly to service edge, thus bypassing step 18

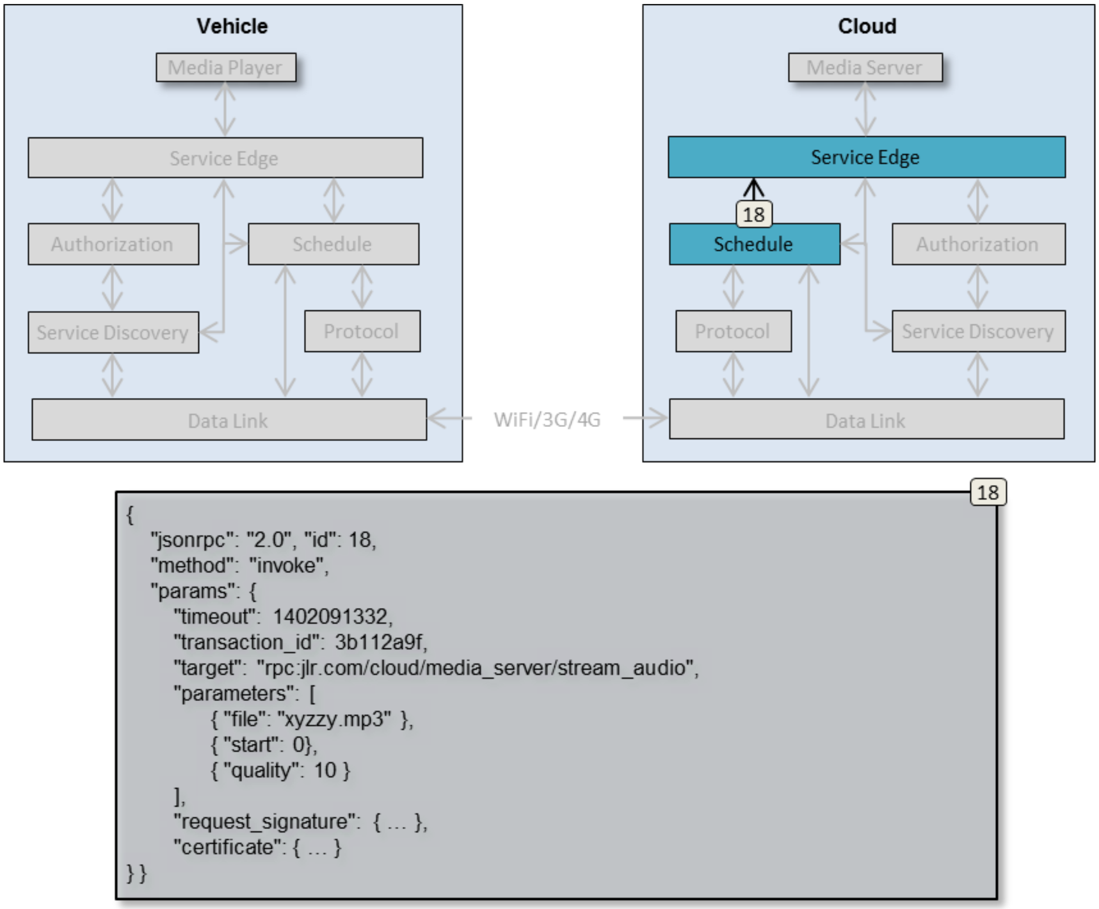


```
{
   "jsonrpc": "2.0", "id": 17,
   "method": "invoke",
   "params": {
      "timeout": 1402091332,
       "transaction_id": 3b112a9f,
       "target": "rpc:jlr.com/cloud/media_server/stream_audio",
       "network_address": " http://176.74.176.178/rvi",
       "parameters": [
           { "file": "xyzzy.mp3" },
           { "start": 0},
           { "quality": 10 }
        ],
        "request_signature": { … },
        "certificate": { … }
}}
```

Figure 27 – Forward decoded content to Schedule.

8.16. Step 18 [Cloud] - Forward request to Service Edge

Schedule sends the decoded request to Service Edge in order to have it forwarded to the targeted Media Server.
NOTE: In ver 0.2, the decoded payload is forwarded directly to service edge, thus bypassing step 18

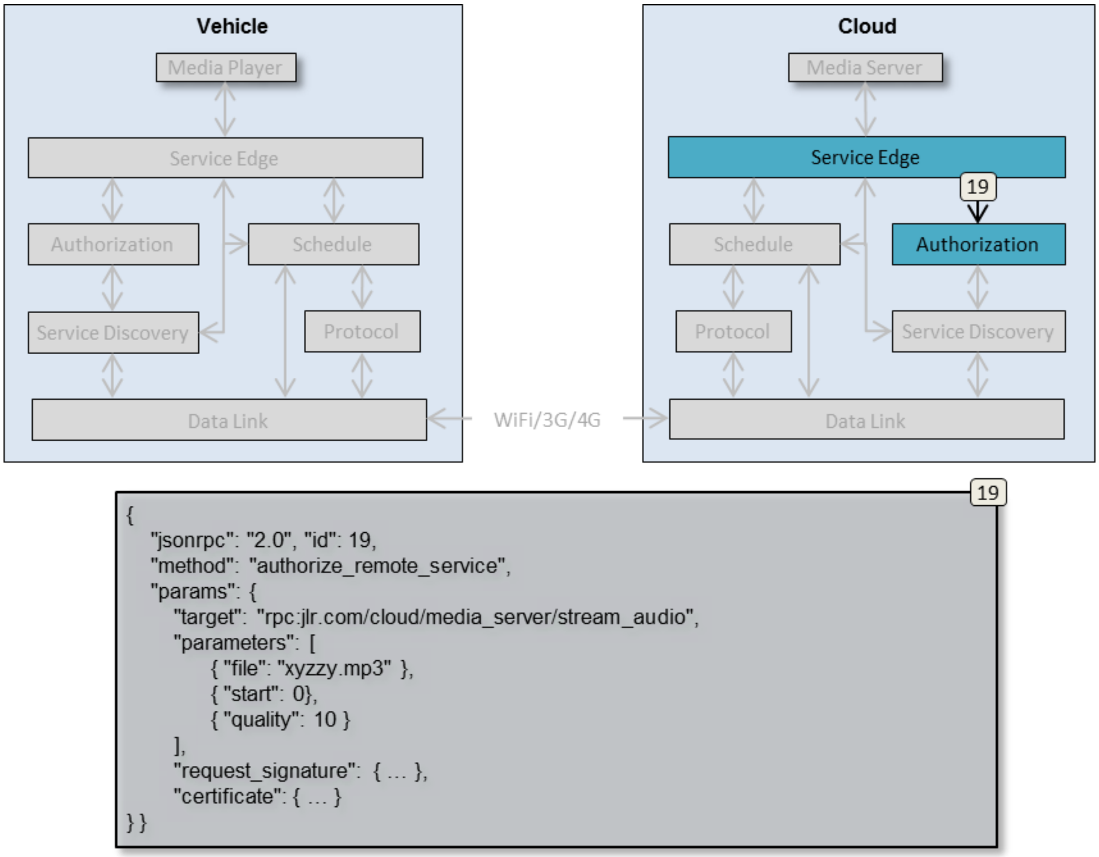


```
{
   "jsonrpc": "2.0", "id": 18,
   "method": "invoke",
   "params": {
      "timeout": 1402091332,
      "transaction_id": 3b112a9f,
      "target": "rpc:jlr.com/cloud/media_server/stream_audio",
      "parameters": [
           { "file": "xyzzy.mp3" },
           { "start": 0},
           { "quality": 10 }
      ],
      "request_signature": { … },
      "certificate": { … }
}}
```

Figure 28 – Forward content to Service Edge.

8.17. Step 19 [Cloud] - Authorize remote request

Service Edge forwards the received request to Authorization in order to have the request and its certificate validated.



```
{
  "jsonrpc": "2.0", "id": 19,
  "method": "authorize_remote_service",
  "params": {
    "target": "rpc:jlr.com/cloud/media_server/stream_audio",
    "parameters": [
        { "file": "xyzzy.mp3" },
        { "start": 0},
        { "quality": 10 }
    ],
    "request_signature": { … },
    "certificate": { … }
}}
```

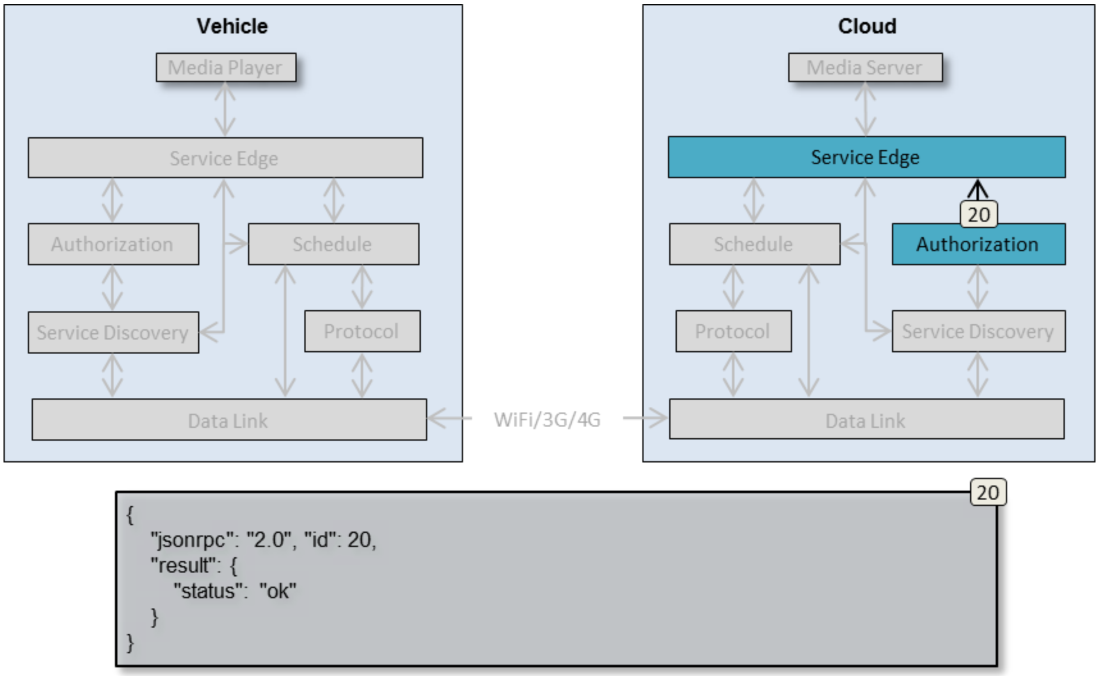
Figure 29 – Authorize remote request.

Authorization will use the request_signature and certificate to validate the request.

If the request is successfully validated, a positive reply is sent back to Service Edge.

8.18. Step 20 [Cloud] - Return Authorization data

Authorization, upon successfully validating the request, returns a positive reply.

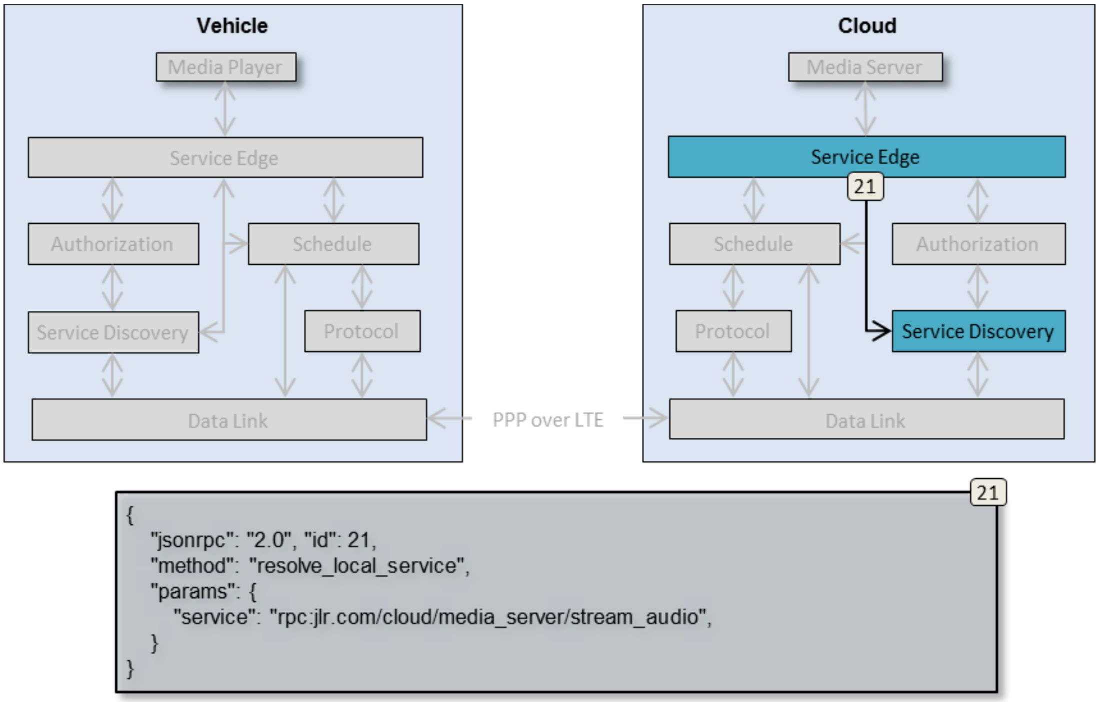


```
{
    "jsonrpc": "2.0", "id": 20,
    "result": {
        "status": "ok"
    }
}
```

Figure 30 – Return authorization result.

8.19. Step 21 [Cloud] - Request Media Server URL

Service Edge queries its local Service Discovery, requesting the URL of the Media Server based on the service name.



```
{
    "jsonrpc": "2.0", "id": 21,
    "method": "resolve_local_service",
    "params": {
        "service": "rpc:jlr.com/cloud/media_server/stream_audio",
    }
}
```

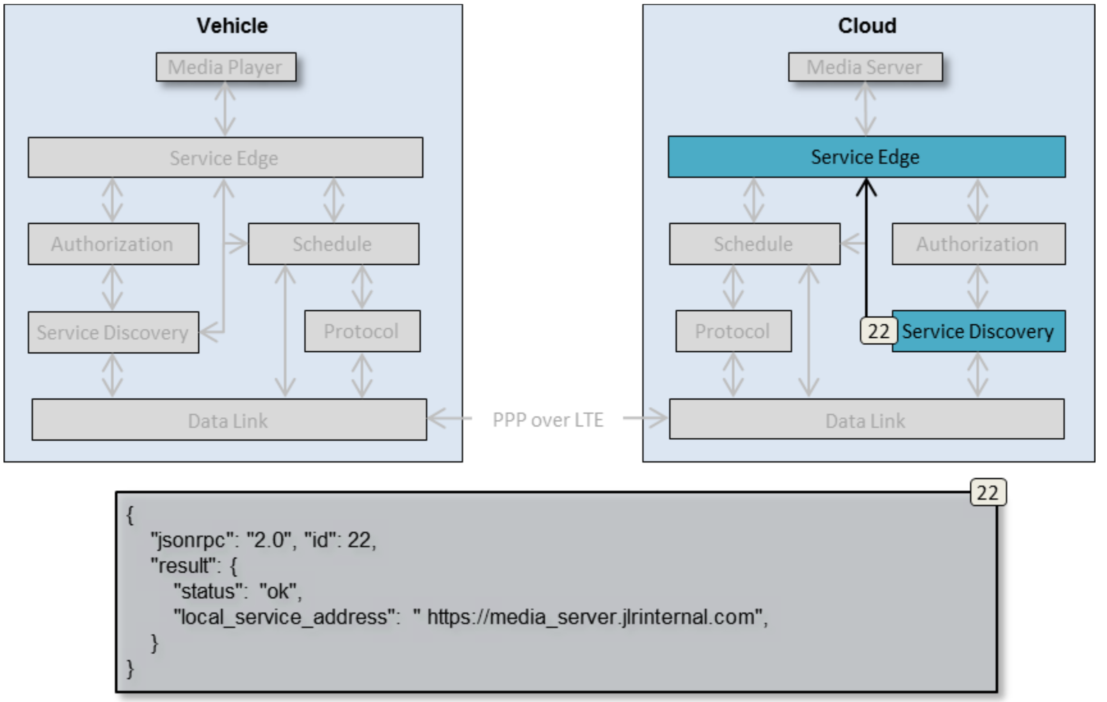
Figure 31 – Resolve service name to local service address.

The service element is extracted from the original request and is used by Service Discovery to map to a network address where the targeted Media Server can be reached.

8.20. Step 22 [Cloud] - Return Media Server local address

Service Edge searches its database with local service registrations, described in chapter “Service Registration, for a service that prefix matches the provided service name.

Once found, the network address of the service (specified in the address element of its register _service request) is returned.

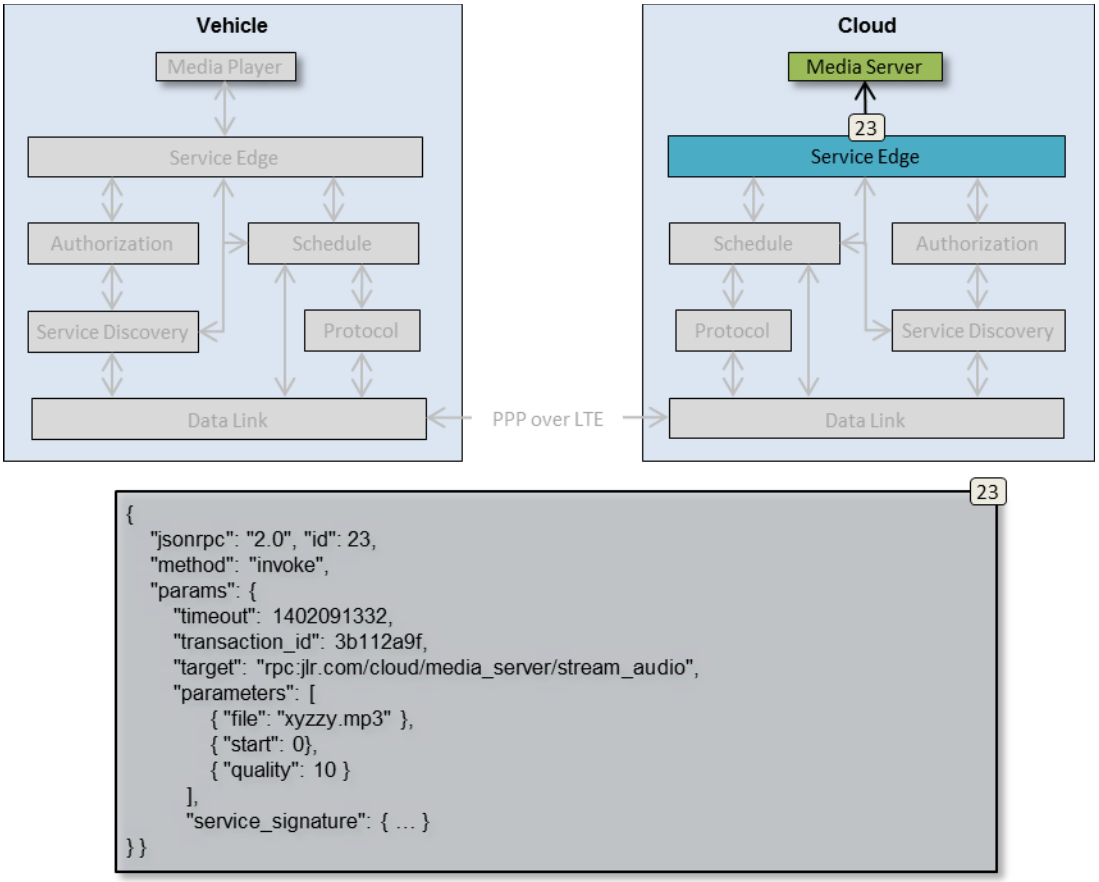


```
{
    "jsonrpc": "2.0", "id": 22,
    "result": {
        "status":  "ok",
        "local_service_address":  " https://media_server.jlrinternal.com",
    }
}
```

Figure 32 – Return local service address.

8.21. Step 23 [Cloud] - Forward request to Media Server

The request is sent from Service Edge to the local Media Server, using the address returned by Service Discovery. The request is signed by Authorization, using the Service Edge's private key so that it can validate the incoming request using the Service Edge's pre-provisioned public key.



```
{
   "jsonrpc": "2.0", "id": 23,
   "method": "invoke",
   "params": {
      "timeout": 1402091332,
      "transaction_id": 3b112a9f,
      "target": "rpc:jlr.com/cloud/media_server/stream_audio",
      "parameters": [
           { "file": "xyzzy.mp3" },
           { "start": 0},
           { "quality": 10 }
        ],
        "service_signature": { … }
}}
```

Figure 33 - Forward request to Media Server.

8.22. Reply Routing

The processing and transportation of a request reply is identical to that of the original request, with the following exceptions.

1. **method element value**
   The “method” element of the request will be changed from “invoke” to “reply”.

2. **transaction_id element value**
   In step 6 of the reply routing, Service Edge will reuse the transaction id of the original request instead of creating a new identity for the reply. This allows the Vehicle Service Edge receiving the reply to reconnect the reply with the original request from the Media Player service.

3. **Step 7-10 are optional**
   Since the reply is sent back from Cloud to Vehicle shortly after the request was sent from Vehicle to Cloud, chances are that the communication channel between the two nodes is still up. In these cases, Step 7-10 are skipped, and Cloud immediately sends the reply to Protocol for encoding and transmission in step 11.

9. Node and Service Provisioning [Future v0.4]

Before a Node, and its locally connected services, can access remote services on other Nodes, it needs to be provisioned with a certificate specifying its access rights, public keys, and other information. See chapter “Certificates” for details.

All certificates in an RVI system are signed by the private key of a provisioning server. The public counterpart of that key is installed in all Nodes’ Authorization as a part of the RVI software install and setup process.

Using its private key, the provisioning server can generate new certificates and have them distributed to Authorization on the Node the certificate was created for. A single Node can host multiple certificates, each specifying a different set of access rights for the Node. The total sum of all certificates’ access rights will be the access rights of the Node as a whole.

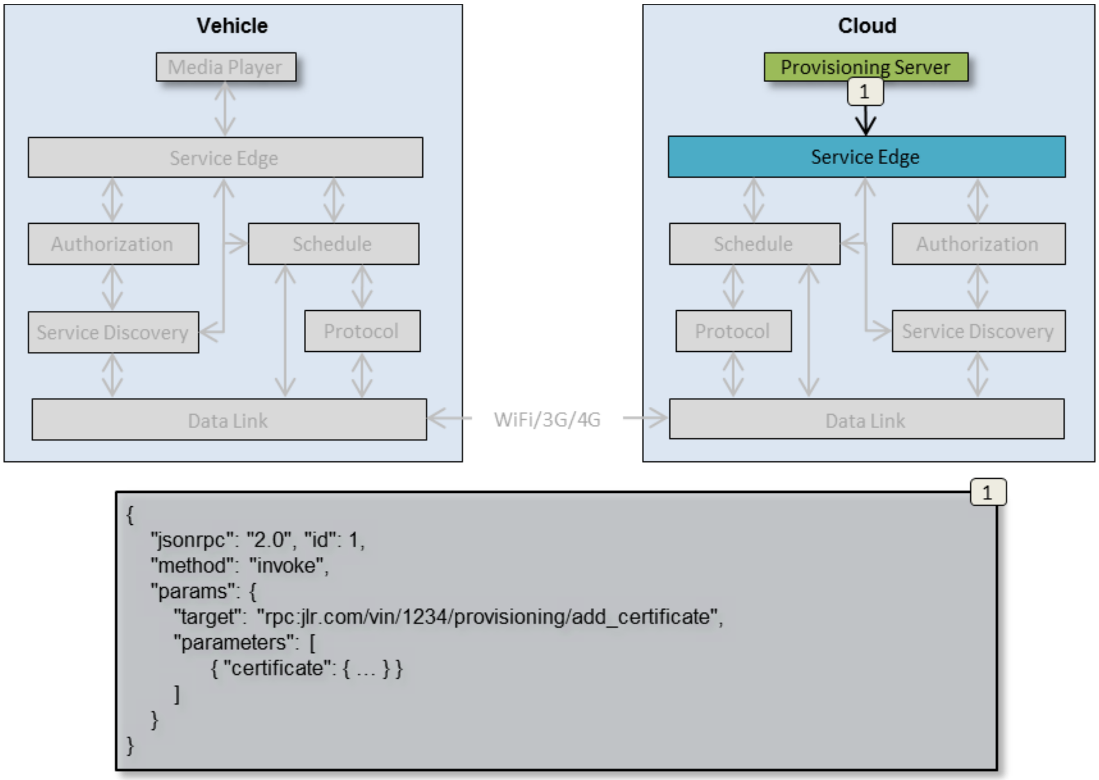
The provisioning server connects and registers to a Node as regular, locally connected service. This Node has a pre-installed certificate, giving it access rights to Authorization on all Nodes for which the provisioning server is to create certificates.

Authorization on each node, in addition to its dedicated authorization and validation channel, is also registered with its local Service Edge as a regular service. Through this local connection, Authorization announces its provisioning services to those remote services that can present the pre-installed certificate described above.

The following chapters describe the distribution process for new certificates.

9.1. Add certificate Step 1 - Send out the request

The provisioning server, having created a new certificate to give Vehicle a set of access rights, sends a request to the Vehicle's provisioning service and its "add_certificate" RPC command.



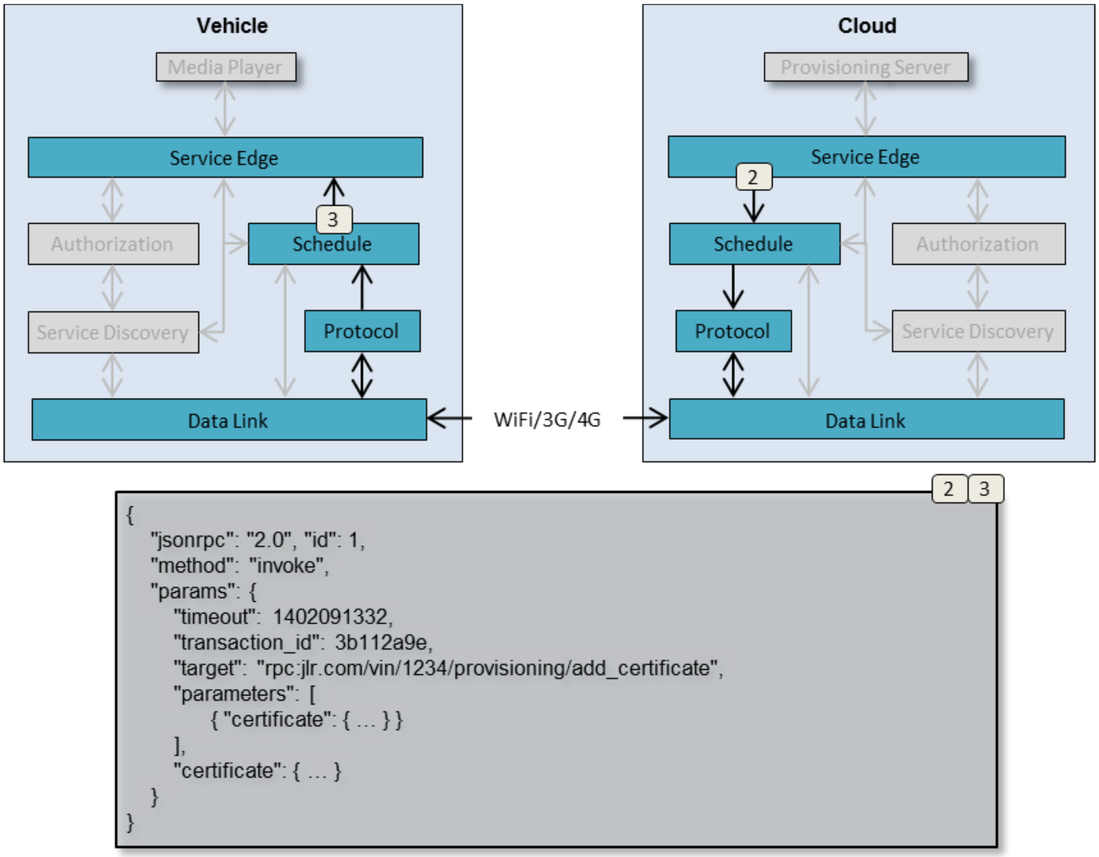
```
{
    "jsonrpc": "2.0", "id": 1,
    "method": "invoke",
    "params": {
        "target": "rpc:jlr.com/vin/1234/provisioning/add_certificate",
        "parameters": [
             { "certificate": { ... } }
        ]
    }
}
```

Figure 34 - Request containing certificate

The "certificate" element contains the new certificate to distribute.

9.2. Add certificate Step 2 – Forward request to target node

The request is validated and forwarded to its target Vehicle node as described in chapter “Request Routing”, up until step 19 (forward the request to its target service).

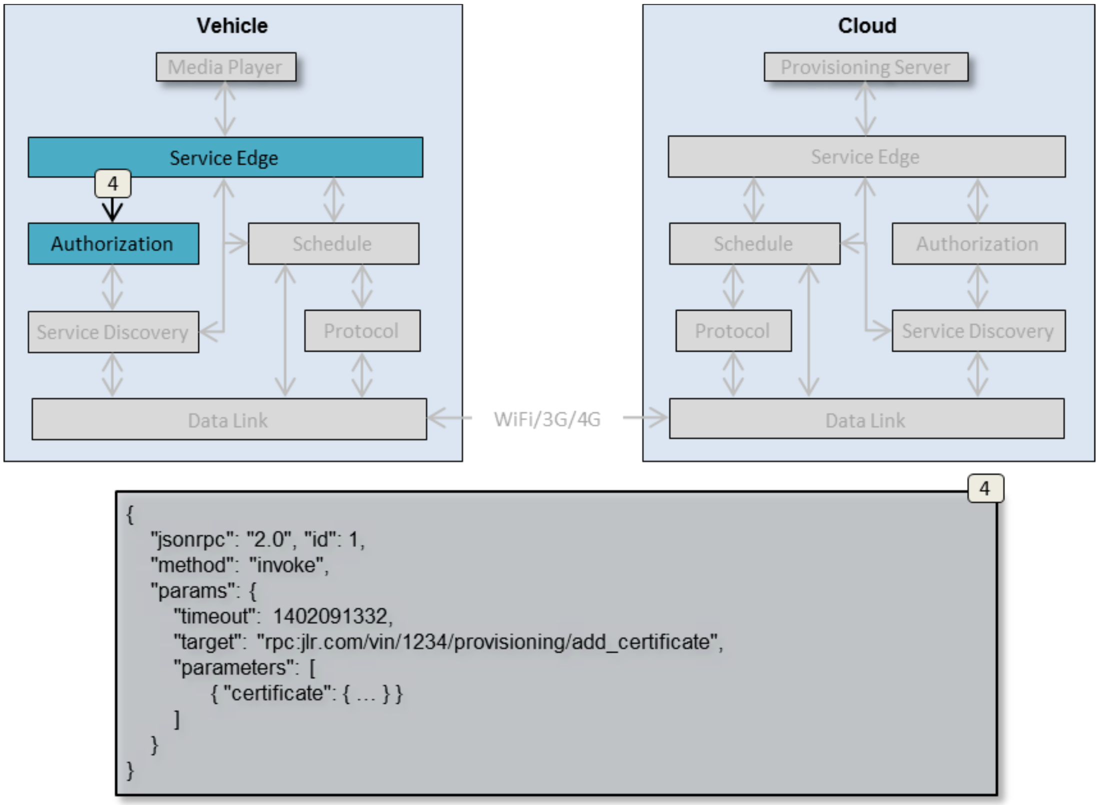


```
{
   "jsonrpc": "2.0", "id": 1,
   "method": "invoke",
   "params": {
      "timeout": 1402091332,
      "transaction_id": 3b112a9e,
      "target": "rpc:jlr.com/vin/1234/provisioning/add_certificate",
      "parameters": [
           { "certificate": { … } }
      ],
      "certificate": { … }
   }
}
```

Figure 35 - Provisioning request transmission to target Node

9.3. Add certificate Step 3 – Deliver request to target Authorization

The target element of the request maps to the services registered by Vehicle’s Authorization. Thus the request is delivered internally to Authorization instead of a locally connected service.



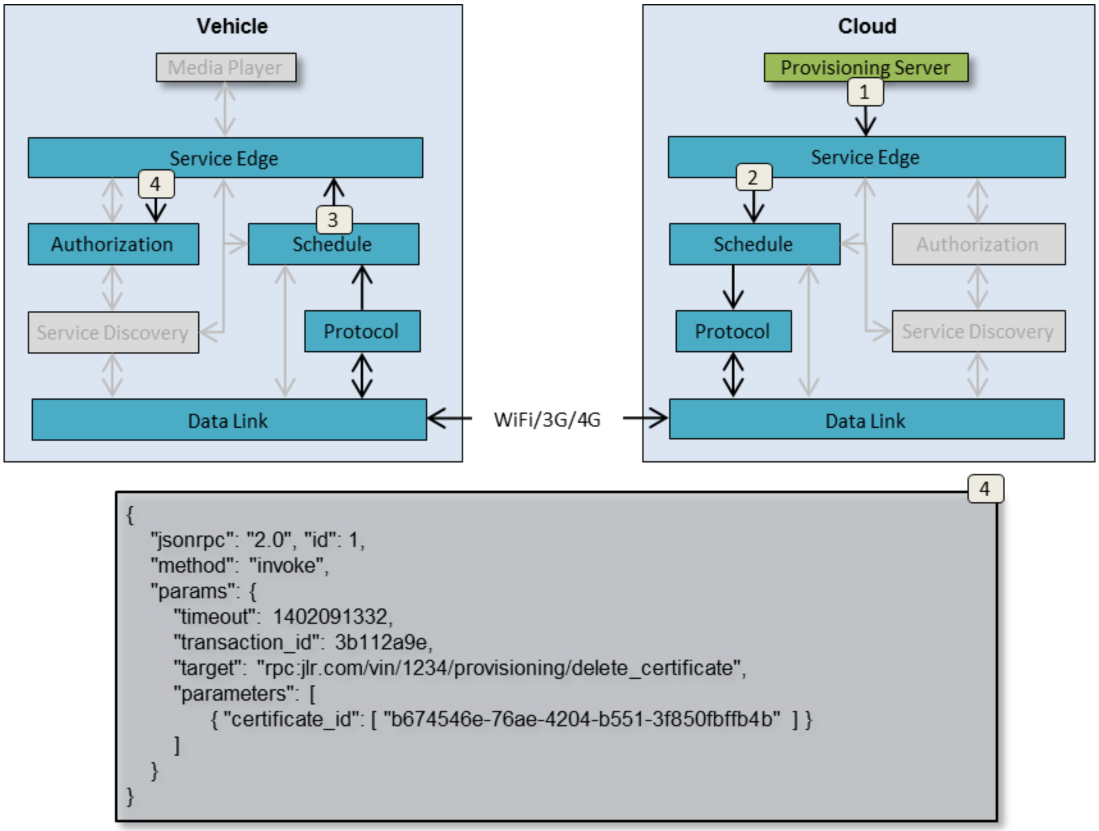
```
{
   "jsonrpc": "2.0", "id": 1,
   "method": "invoke",
   "params": {
      "timeout": 1402091332,
      "target": "rpc:jlr.com/vin/1234/provisioning/add_certificate",
      "parameters": [
           { "certificate": { … } }
      ]
   }
}
```

Figure 36 - Internal provisioning request delivery

Authorization, acting as a service, will store the received certificate in its internal persistent storage and use it in all future authorization processes toward other nodes.

9.4. Delete certificates

Certificate deletion follows the same flow of events as when a certificate is added, but with a different service invoked.



```
{
    "jsonrpc": "2.0", "id": 1,
    "method": "invoke",
    "params": {
        "timeout": 1402091332,
        "transaction_id": 3b112a9e,
        "target": "rpc:jlr.com/vin/1234/provisioning/delete_certificate",
        "parameters": [
            { "certificate_id": [ "b674546e-76ae-4204-b551-3f850fbffb4b"  ] }
        ]
    }
}
```

Figure 37 - Delete certificate

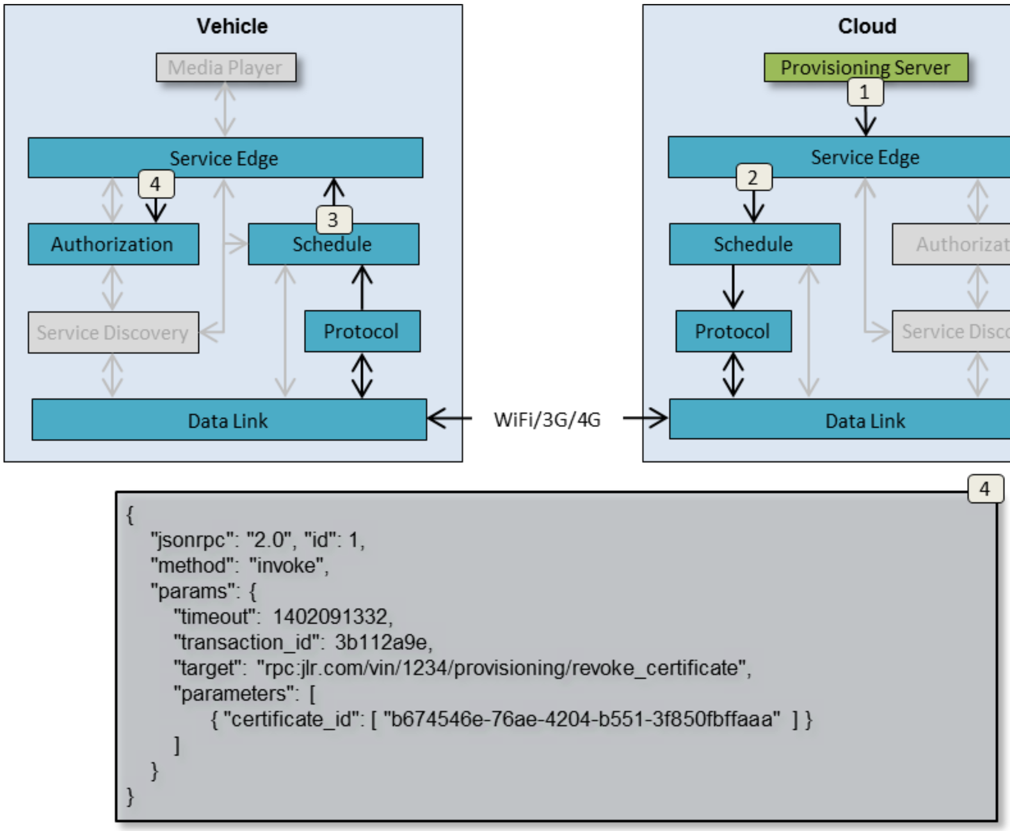
The certificate_id element specifies the unique ID of the certificates to remove from the target node. Authorization will permanently delete the given certificates from its persistent storage.

9.5. Revoke certificates

Certificate revocation is the process of announcing to multiple nodes that a given certificate shall no longer be accepted when received from remote nodes. This effectively expires a certificate before its active period has run out.

While certificate deletions affects only what a single node sends out as credentials to other nodes, revocations are sent out to multiple nodes and affects which credentials are accepted from other nodes.

In order for certificate revocation to be efficient, a mass distribution mechanism with a high success rate, such as SMS, will be necessary.



```
{
    "jsonrpc": "2.0", "id": 1,
    "method": "invoke",
    "params": {
        "timeout": 1402091332,
        "transaction_id": 3b112a9e,
        "target": "rpc:jlr.com/vin/1234/provisioning/revoke_certificate",
        "parameters": [
            { "certificate_id": [ "b674546e-76ae-4204-b551-3f850fbffaaa"  ] }
        ]
    }
}
```

Figure 38 - Certificate revocation

The certificate(s) specified by certificate_id element will be permanently added by the receiving Authorization to a list of blacklisted certificates that should no longer be recognized if received from remote nodes.

9.6. P2P Access Granting

If a node wants to grant another specific node access to its services without having to invoke a remote provisioning server, the granting node (which will give out access rights) can issue a P2P certificate with specified access rights to the given services. This certificate follow the same specification and use cases as regular certificates, with the single exception that it is signed by a private key owned by the granting node.

Once created, the granting node distributes the certificate to the granted node (which will receive access rights) as described in chapter “Add certificate Step 1 - Send out the request” and subsequent chapters.

The granted node can then use the certificate to sign its requests to the granting node, which will use the public key of the certificate to validate the request.

The created P2P certificate is only useable between the two given nodes since it explicitly gives access to the granted node (through the certificate’s “sources” element), and has been signed by the granting node. The granted node cannot use the certificate anywhere else since no other node has the certificate’s public key to validate the request.

10. Service Edge feature set

Service Edge supports a number of operations toward connected services, Authorization, and Service Discovery.

10.1. Local Service Registration

A local service will be able to connect to Service Edge and register itself as described in chapter “Service Registration”. Once successfully registered the local service shall be able to send and receive requests, as well as subscribe to and receive service availability reports from Service Edge.

10.2. Service availability reporting

When a remote or local service becomes available, other services, as well as Service Discovery, shall have the option of being informed of the event.

A subscription command sent to Service Edge will contain a service name prefix for which any matching services shall be reported.

There are two events that can trigger a service availability report to a locally connected service or Service Discovery.

First, a service availability report may be sent by Service Discovery as a reaction to a received service announcement from Data Link, which means that there is now a data link available to a remote service that matches the subscribed-to service name prefix. In these cases a locally connected service will be informed that a remote service is available.

Second, a local service may register directly with Service Edge, thus making the service available for locally and remotely originated requests. In these cases, Service Edge will report the event to Service Discovery, a service availability subscriber who in its turn will forward it to relevant remote nodes.

10.3. Process requests from local services

A locally connected service may send requests (RPCs, replies, messages, and metrics) to Service Edge for processing and forwarding to its target service. See chapter “Request Routing” for details.

10.4. Process requests from remote services

Service Edge can also receive incoming requests from Protocol, which are to be forwarded to a locally connected service. Service Edge will use Service Discovery to map

11. Service Discovery feature set

11.1. Register local services

Service Edge shall register locally connected services with Service Discovery, specifying their network address and their service names to match incoming requests against. The received information will be used to resolve the targeted service name to the network address of the locally connected service that can handle the request.

11.2. Register node to network address mappings

Service Discovery will also receive information about remotely available services from Data Link. Such services will be stored, with their corresponding network address, in Service Discovery. When the communication channel supporting the remote service communication disappears, Service Discovery will be informed.

Any subscribers to remote service availability will be informed of remote services becoming available and unavailable, as long as the service name matches the pattern provided with the subscription request.

11.3. Resolve service to network address

Service Discovery will be able to map a name, describing a service, to a network address that requests can be sent to. Information about the service name-to-network address mapping is provided by the two authorize and announce stages described in chapter “Service Registration”.

11.4. Process incoming service announcements

Data Link will, during its authorize and announce stage, receive service announcements from other nodes with information about network addresses, and services. The remote service information is forwarded by Data Link to Service Discovery, which will match the reported services against its subscription tables and send out reports. Service Edge is a typical such subscriber and will, in its turn, forward the information to its locally connected services.

11.5. Send outgoing service announcement

When Data Link reports that a remote node is available for communication through its authorization step, Service Discovery will be informed.

Service Discovery will respond by filtering all available locally connected services through the “destinations” element of the certificate provided by the remote node. See chapter “Certificates” for details.

All local services matching the destinations entry will be forwarded to Data Link, which in its turn will construct an announce message with the service list to the remote node. See chapter “Service Announcement” for details.

11.6. Process communication channel availability reports

Data Link will report when communication channels to other nodes become available and disappear. Such reports will trigger service availability reports to Service Edge, who will forward them to their subscribing services.

12. Authorization feature set

12.1. Authorize local requests

Once registered, locally connected services will submit “invoke” requests to be sent to a targeted remote service. See chapter “Step 1 [Vehicle] - Submit request to Service Edge” for details.

Service Edge will forward the received “invoke” request to Authorization to have it validated. Authorization will check that the signature, generated by the service’s private key, can be verified with its public key.

Authorization will also verify that the Node that both Service and Authorization executes on has the right to invoke the targeted service. This is done by pattern matching the “destinations” element of the Node’s certificate againstthe name of the targeted service. See chapter “Certificates” for details.

12.2. Authorize remote requests

A remote request received and decoded by Protocol and forwarded to Service Edge, will be sent to Authorization for validation. See chapter “Step 19 [Cloud] - Authorize remote request” for details.

The request will contain a certificate, describing the rights of the sending node, and a signature generated by the private counterpart of the certificate’s public key. Authorization will validate the certificate, signature, and the remote node’s access rights to the targeted service.

12.3. Provision certificates

Authorization must provision, delete, and blacklist certificates when directed to do so from a provisioning server. Please see chapter “Node and Service Provisioning” for details.

13. Schedule feature set

13.1. Process local requests

A request received from local Service Edge will be stored until a data link to its target node is available. If the request times out before such a link becomes available, or it times out before a reply has been received (when applicable), a timeout is sent back to Service Edge to be forwarded to the service originating the request.

13.2. Process remote requests

Remote requests received from Protocol needs to be stored in cases where the targeted locally connected service is not available. Timeouts have to be sent back to the originating Node in case the service does not become available to process the request within the timeout interval.

13.3. Process data link availability reports

When Data Link reports that a communication channel, and its associated services, is available, Schedule will traverse all pending requests received from locally connected services. Those requests targeting now available services are forwarded to Protocol for encoding and transmission to the remote Node.

13.4. Process service availability reports

When Service Edge reports to Schedule that a locally connected service is available, Schedule will traverse all pending requests received from remote nodes. Those requests targeting the now available local service are sent to Service Edge to be forwarded to the service itself.

14. Protocol feature set

14.1. Encode and transmit requests

Protocol shall receive requests from Schedule, encode them to an implementation-specific protocol, and send them to their destination network addresses. Protocol will only be called upon by Schedule after Data Link has reported that a communication channel to the destination node is up. Thus, a network error should be reported back to Schedule so that it can re-queue the request for a later retransmission.

14.2. Receive and decode requests

Inbound communication from a remote Protocol should be decoded and forwarded to Schedule.

15. Data Link feature set

15.1. Setup communication channel

Data Link shall be able to setup a communication channel to a remote node, identified by a network address. Once the communication channel is up, a report is sent to any subscribing local processes, such as Schedule.

15.2. Disconnect communication channel

When a communication channel has been idle for a period of time, or is explicitly ordered to be disconnected, Data Link shall shut it down and free the associated resources.

15.3. Transmit data payload

In cases where Protocol, instead of transmitting a data packet itself, elects to have Data Link transmit the payload, Data Link shall forward the payload to its remote counterpart.

15.4. Receive data payload

When Data Link receives a data payload from a remote counterpart, it shall forward it to a suitable Protocol for further processing.

15.5. Send node authorization

When Data Link has established a communication channel with a remote node, it shall send an authorization package as described in chapter "Authorization".

15.6. Receive remote Node authorization

When Data Link has established a communication channel with a remote node, it shall be prepared to receive and process an authorization package as described in chapter "Authorization".

15.7. Send service announcement

Once Data Link has authorized a remote counterpart, it shall send a service announcement package to the remote Node as described in chapter "Service Announcement".

15.8. Receive service announcement

Once Data Link has sent its own authorization to a remote counterpart, it shall be prepared to receive and process a service announcement from the remote node as described in chapter "Service Announcement".

15.9. Report communication channel availability

Data Link shall report when communication channels to remote nodes becomes available, or disappear, to all local components who have subscribed to such information. A typical example of such a subscriber is Schedule.