Arrowhead Adapter for Arrowhead services

1. **Introduction**

The Arrowhead Adapter module acts as a „legacy gateway”: it represents legacy or custom UDP-based communications as Arrowhead services. Moreover, it interacts with the mandatory core systems: it registers as service provider(s) to the Arrowhead Service Registry module and serves consumer queries, converting from the custom data format into SenML [1]. It can also request orchestration automatically and connect to another Arrowhead service when the legacy device requests its usual information.

The Arrowhead Adapter module is also compatible with REST Providers and applies an MQTT broker (Eclipse Mosquitto) to serve MQTT Subscribe/Publish events, or UDP/SenML to MQTT translation.

The latest version of the Adapter module (v0.2.9) supports Plug&Play operation on Linux systems and compatible with the Arrowhead v3.2 M3 and M4 modules. The operator should build the source on the appropriate platform, set the proper information in the configuration files and simply run the module. We compiled and tested the functionalities on a Raspberry PI 3 [2] (Raspbian operating system).

Figure 1 represents a system overview and the currently available operation modes.

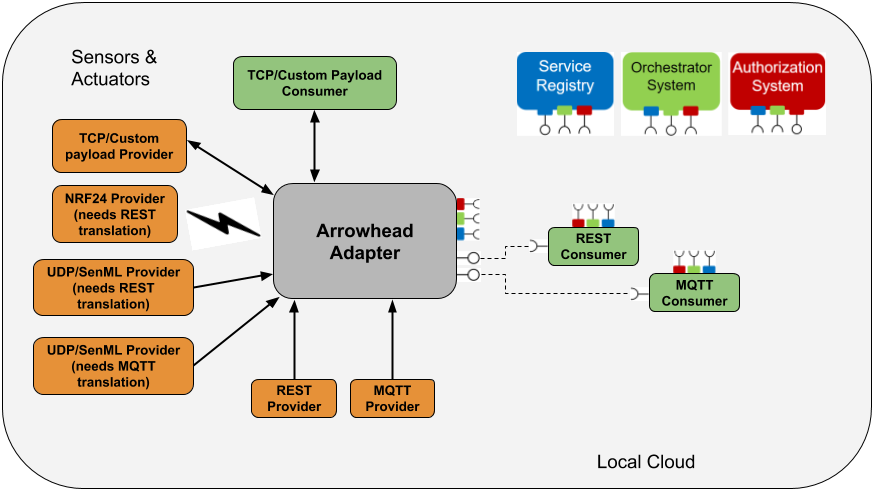


Figure 1 – Arrowhead Adapter v0.2.9 operation modes

1. **Basic operation**

The Arrowhead Adapter module defines a *mote interface* for local communication (port 65000 on localhost or the eth0/wlan0 interface IP address). This *mote interface* accepts UDP messages from providers or consumers with SenML [1] content.

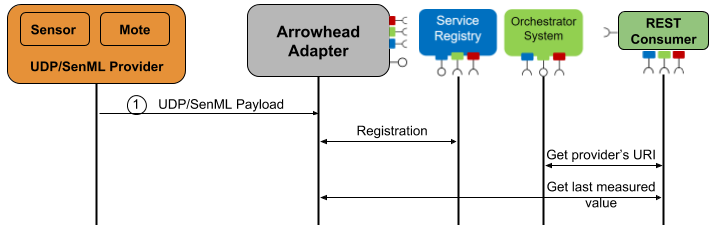
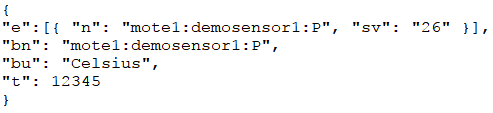


Figure 2 – Simplified UDP/SenML Provider communication flow

Figure 2 represents a high level communication sequence chart. After a SenML message reception, the Adapter module examines the sensorID/moteID parameters. When the sensor is not registered yet, the Adapter module registers a special URI into the Service Registry module, and stores the last measured value with some additional information in a local database.

The Adapter module supports HTTP and HTTPS communication between the Arrowhead core system modules.

① In a case of a UDP/SenML Provider communication, the Adapter module accepts the following SenML format:



* bn means **b**ase **n**ame
* bu means **b**ase **u**nit
* t means the **t**ime of the measurement
* sv means the measured **v**alue in **s**tring format
* n means the sensor **n**ame, which should be equal to the bn parameter

To differentiate the provider and consumer communication on the UDP socket, the **b**ase **n**ame parameter of the SenML message must be in *moteID:sensorID:Flags* format.

We differentiate the following Flags inside the SenML message in version 0.2.9:

* *P* – means a basic Provider, which needs REST translation
* *PS* – means a basic Provider, which needs REST translation using HTTPS
* *PM* – means a basic UDP/SenML Provider, which needs MQTT translation
* *C* – means a basic Consumer. The interface definition for a custom communication must be implemented in the Adapter module.

The URI generation for a provider is also based on this special base name format. For example, when the Adapter’s IP address is 10.0.0.2 and the port for the REST communication is 4567, the registered URI will be: <http://10.0.0.2:4567/moteID/sensorID/serviceInterface>

The Arrowhead Adapter module defines a HTTP/HTTPS REST *interface* for the providers. This *REST interface* accepts HTTP/HTTPS POST messages. The POST message must contain the same SenML content, which is defined for the UDP providers. The IP address for the REST communication is equal with the IP address of the eth0/wlan0 interface, and the Port number is defined by the port value of the [AHServiceRegistry] entry in the adapter.ini configuration file. The HTTPS port number is HTTP port + 1.

② In a case of a pure UDP/SenML Consumer, the bn field is the only mandatory parameter:



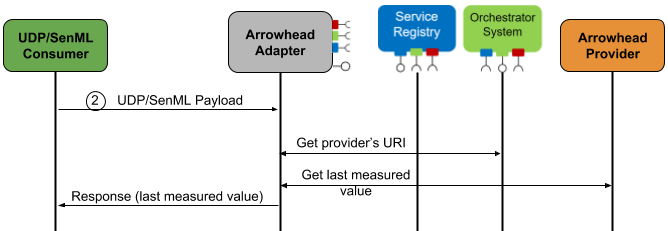


Figure 3 – Simplified UDP/SenML Consumer communication flow

*Figure 3* represents a high level communication sequence chart. After the consumer’s SenML message reception, the Adapter module examines the sensorID/moteID parameters.

When the orchestration information is already available for the consumer, the Adapter responds with the last measured value. When the orchestration information is unknown, the Adapter module sends an orchestration request to get the provider’s URI. After a successful orchestration the Adapter module requests a measured value based on the defined interface. (We note that custom service interfaces should be implemented within the Adapter module).

A HTTP/HTTPS REST capable consumer can request directly the providers URI for the measured value from the Adapter module (see Figure 3).

1. **Interfaces**

The Adapter v0.2.9 module applies the following connection points:

* UDP server socket (port 6500 on localhost or the eth0/wlan0 interface IP address) to receive and send messages from the legacy device
* TCP/HTTP or TCP/HTTPS client socket towards Arrowhead-ServiceRegistry
* TCP/HTTP or TCP/HTTPS client socket towards Arrowhead-Orchestrator
* TCP/HTTP or TCP/HTTPS server socket to receive communication from REST consumers (HTTP/HTTPS GET) or REST providers (HTTP/HTTPS POST)
* MQTT Broker for handling PUBLISH/SUBSCRIBE events

1. **Configuration files for Providers**

* adapter.ini, [AHServiceRegistry] section
  + *baseUri* means the HTTP URI of the ServiceRegistry module
  + *secureBaseUri* means the HTTPS URI of the ServiceRegistry module
  + *port* means the Adapter’s TCP port for the ServiceRegistry communication
  + *port* is also used for REST provider communication (HTTP POST)
* *adapter.ini, [AHAdapter] section*
  + *address* means the IPv4 address of the network interface (e.g. eth0), which is used to connect to the ServiceRegistry
  + *address6* means the IPv6 address of the network interface (e.g. eth0), which is used to connect to the ServiceRegistry. When the *address* field is empty, the *address6* value will be used during the communication.
  + *moteIntfUDPPort* means the UDP port for the UDP/SenML communication
  + *secureAHIntf* defines HTTP/HTTPS connection between Adapter and Core System modules, (0 – HTTP, 1 - HTTPS)
  + Each field is mandatory in this configuration file.
* moteTable.json
  + The configuration file contains the systemName parameters for the current mote. The moteID must be equal with the moteID parameter in the providedService.json configuration file.
* providedService.json
  + The configuration file contains several Arrowhead-specific information, which will be used during the registration procedure. The sensorID and moteID parameters must be equal with the parameters in the SenML Base Name field (see Provider communication section).
  + Each field is mandatory in this configuration file.
* whitelist.csv
  + The configuration file contains the allowed providers.
  + The first column must be equal with the Base Name parameter of the SenML message, which is sent by the Provider.
  + The second column is the port for the communication, which parameter is not used yet.
  + The third column is the type of the sensor, which is not used yet.

1. **Configuration files for Consumers**

* adapter.ini, [AHOrchestrator] section
  + *baseUri* means the HTTP URI of the Arrowhead-Orchestrator module
  + *secureBaseUri* means the HTTPS URI of the Arrowhead-Orchestrator module
  + *port* means the Adapter’s TCP port for the Arrowhead-Orchestrator communication
* *adapter.ini, [AHAdapter] section*
  + *address* means the IPv4 address of the network interface (e.g. eth0), which is used to connect to the Orchestrator
  + *address6* means the IPv6 address of the network interface (e.g. eth0), which is used to connect to the Orchestrator. When the *address* field is empty, the *address6* value will be used during the communication.
  + *moteIntfUDPPort means the UDP port for UDP/SenML communication*
  + *secureAHIntf* defines HTTP/HTTPS connection between Adapter and Core System modules, (0 – HTTP, 1 - HTTPS)
  + Each field is mandatory in this configuration file.
* consumedServices.json
  + The configuration file contains HTTP or HTTPS request forms for the orchestration. When a consumer sends an UDP/SenML message with a C flag, the Adapter module uses this configuration file to orchestrate for the requested provider URI.

Each configuration file must be in the same directory as the executable Adapter module.

1. **Build & Run**

The Adapter project uses the libmicrohttpd and libcurl libraries for the communication, and openssl for the cryptography algorithms.

Install the following Linux packages (sudo apt-get install on Raspbian):

openssl, libgnutls28-dev, libgnutlsxx28, libssl1.1, libssl1.0-dev, libcurl3, libcurl3-gnutls, libcurl4-gnutls-dev, libcrypto++-dev, libcrypto++-utils, libcrypto++6, libgpg-error-dev, automake, texinfo, g++

The project uses (or tested with) libmicrohttpd-0.9.59. Download, compile and install it from source with HTTPS support. Before the *make* command, update the microhttpd.h and connection.c source files from SensorAdapter/changes/libmicrohttpd directory. Use the following commands:

* tar -xvzf libmicrohttpd-0.9.59.tar.gz
* ./configure --with-gnutls
* make
* sudo make install

Create libmicrohttpd.so.12 file in /usr/lib or usr/local/lib directory (or where *ldd adapter* command points):

* cd /usr/lib
* sudo ln –s /usr/local/lib/libmicrohttpd.so.12.46.0 libmicrohttpd.so.12

In a case of missing libcrypto.so:

* cd /usr/lib
* sudo ln –s libcrypto.so.1.0.0 /lib/arm-linux-gnueabihf/libcrypt-2.24.so

The Adapter project contains a Makefile to build the sources for the current architecture.

* make
* ./Adapter [-secure]

To support UDP/SenML -> MQTT translation, download, build and install the Eclipse Mosquitto project, using the following commands. Before make, copy (if necessary, overwrite) the source files from SensorAdapter/changes/MQTT directory to the downloaded mosquitto/src directory.

* tar -xvzf mosquitto-1.4.15.tar.gz
* make
* sudo make install

To build the Eclipse Mosquitto project, the following linux packages must be installed (sudo apt-get install):

* libc-ares-dev
* uuid-dev
* xsltproc
* libjson-c-dev

Note:

There is a Provider and Consumer example Linux project (tested on Raspbian) written on C++. The Provider example registers a basic REST/JSON-SenML provider into the Arrowhead Service Registry module, and defines a HTTP or HTTPS interface for the Consumer. The Consumer example project orchestrates from the Arrowhead Orchestrator the Provider’s URI (on HTTP or HTTPS), and sends HTTP or HTTPS requests to receive the latest measured value. (See Linux package dependency in section VI. Build & Run)

The example project source is available on github [3].

1. **References**
2. C. Jennings, Z. Shelby, J. Arkko, A. Keranen, C. Bormann, “Media Types for Sensor Measurement Lists (SenML)”, online available: https://tools.ietf.org/html/draft-ietf-core-senml-14
3. Raspbian operating system, online available: https://www.raspberrypi.org/downloads/raspbian/
4. Example Arrowhead Provider and Consumer projects, online available: <https://github.com/hegeduscs/arrowheadclient/tree/M4-dev/ArrowheadCpp>