DAPNET 2.0 Concept and Interface Definition

Ralf Wilke, DH3WR
Thomas Gatzweiler, DL2IC
Phillip Thiel, DL6PT

July 7, 2018

Abstract

This is the concept and interface description of the version 2 of the DAPNET. It's purpose in comparison to the first version released is a more robust clustering and network interaction solution to cope with the special requirements of IP connections over HAMNET which means that all network connections have to be considered with a WAN character resulting in unreliable network connectivity. In terms of consistence of the database, "eventually consistence" is considered to be the most reachable. There are "always right" database nodes inside the so called HAMCLOUD. In case of database conflicts, the version inside the HAMCLOUD cluster is always to be considered right.

Chapter 1

Introduction

more text

1.1 Key Features

1.2 Historic Background

write some history

1.3 Concept presentation

An overview of the DAPNET 2.0 concept is given in Fig. 1.1.

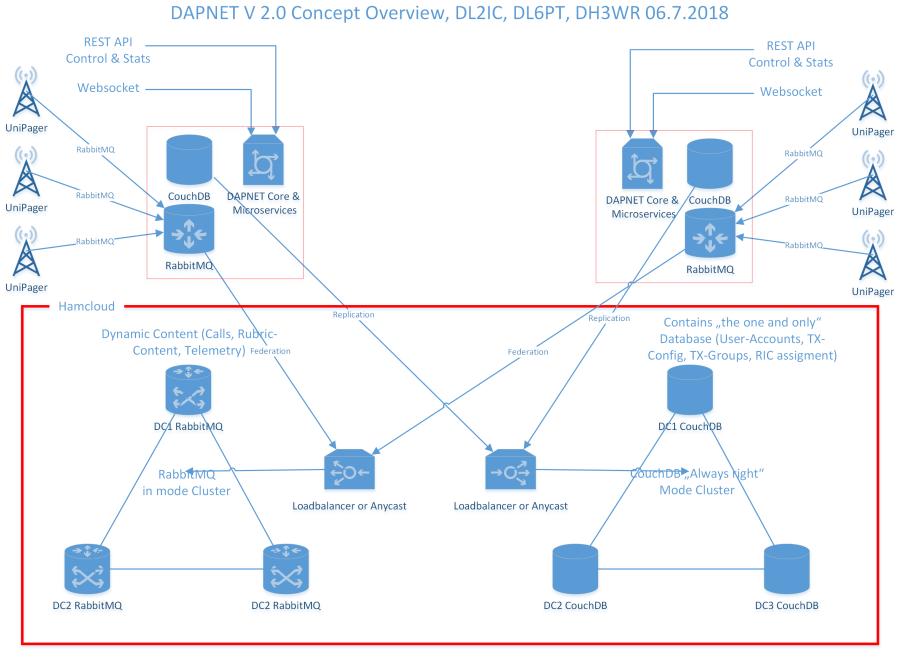

Figure 1.1: Overview of DAPNET Clutering and Network Structure

The details of a single node implementation are shown in Fig. 1.2.

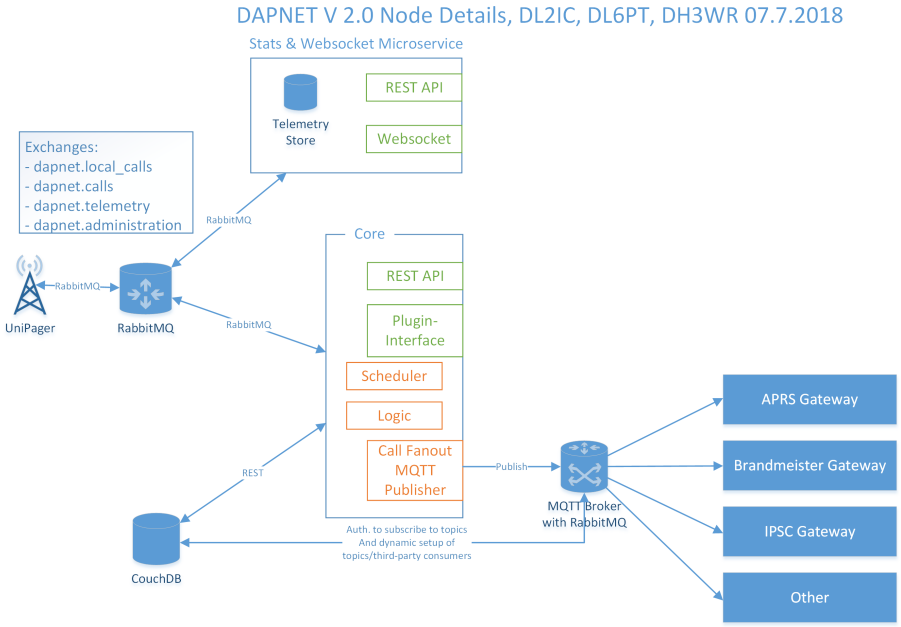

Figure 1.2: Node Details

1.4 Transmitter Software

1.4.1 Unipager

1.4.2 DAPNET-Proxy

1.5 DAPNET Network

1.5.1 Overview and Concept

1.5.2 Used third-party Software

1.5.3 HAMCLOUD Description

The HAMCLOUD is a virtual server combination of server central services on the HAMNET and provide short hop connectivity to deployed service on HAMNET towards the Internet. There are three data centers at Essen, Nürnberg and Aachen, which have high bandwidth interlinks over the DFN. There are address spaces for uni- and anycast services. How this concept is deployed is still tbd. More information is here https://www.swiss-artg.ch/fileadmin/Dokumente/HAMNET/HamCloud_-_Angebotene_Dienste_in_der_HamCloud.pdf and here http://hamnetdb.net/?m=as&q=hamcloud.

Define if uni- or anycast entry points will exist

1.5.4 Rubric Handling Concept

1.5.5 Queuing Priority Concept

Chapter 2

DAPNET Network Definition

2.1 Cluster Description

2.1.1 Real-time Message delivery with RabbitMQ

2.1.2 Distributed Database with CouchDB

2.1.3 Authentification Concept

2.1.4 Integration of new Nodes

2.2 Interface Definition

2.2.1 RabbitMQ Exchange

2.2.2 CouchDB Interface

2.2.3 Core REST API

2.2.4 Statistic and Telemetry REST API

2.2.5 Websocket for real-time updates on configuration, Statistics and Telemetry API

2.2.6 MQTT Fanout for third-party consumers

Chapter 3

Internal Programming Workflows

3.1 Sent calls

3.2 Add, edit, delete User

Show current users

1. Get current status via REST GET to /users on Core URL
2. Handle updates via Websocket

Add and Edit User

1. If edit: Get current status via REST GET to /users/<username> on Core URL
2. Show edit form and place data
3. On save button event, send REST POST to /users/<username> on Core URL

The core will update the CouchDB and generate a RabbitMQ administration message to inform all other nodes. This information is transmitted by the Stats and Websocket Micro-Service to all connected websocket clients to get them updated. This will also happen for the website instance emitting the edit request, so its content is also updated.

Delete User

1. Ask "Are you sure?"
2. If yes, sent REST DELETE to /users/<username> on Core URL

The core will update the CouchDB and generate a RabbitMQ administration message to inform all other nodes. This information is transmitted by the Stats and Websocket Micro-Service to all connected websocket clients to get them updated. This will also happen for the website instance emitting the edit request, so its content is also updated.

3.3 Add, edit, delete Subscriber

3.4 Add, edit, delete Node (tbd)

3.5 Add, edit, delete Transmitter

3.6 Implementation of Transmitter Groups

3.7 Add, edit, delete Rubrics

Show current configuration

1. Get current status via REST GET to /rubrics on Core URL
2. Handle updates via websocket

Add and Edit rubrics

1. If edit: Get current status via REST GET to /rubrics/<rubricname> on Core URL
2. Show edit form and place data
3. On save button event, send REST POST to /users/<rubricname> on Core URL

The core will update the CouchDB and generate a RabbitMQ administration message to inform all other nodes. This information is transmitted by the Stats and Websocket Micro-Service to all connected websocket clients to get them updated. This will also happen for the website instance emitting the edit request, so its content is also updated.

Delete rubric

1. Ask "Are you sure?"
2. If yes, sent REST DELETE to /users/<rubricname> on Core URL

The core will update the CouchDB and generate a RabbitMQ administration message to inform all other nodes. This information is transmitted by the Stats and Websocket Micro-Service to all connected websocket clients to get them updated. This will also happen for the website instance emitting the edit request, so its content is also updated.

3.8 Add, edit, delete Rubrics content

3.9 Add, edit, delete, assign Rubrics to Transmitter/-Groups

3.10 Ports and Loadbalacing Concept

3.11 Periodic Tasks (Scheduler)

3.12 Plugin Interface

3.13 Transmitter Connection

Transmitter connections consist of two connections to a Node. A REST connection for initial announcement of a new transmitter, heartbeat messages and transmitter configuration and a RabbitMQ connection to receive the data to be transmitted.

The workflow for a transmitter connection is the following:

1. Announce new connecting transmitter via Core REST Interface (6.1.1).
2. Get as response the transmitter configuration or an error message (6.1.2).
3. Initiate RabbitMQ connection to get the data to be transmitted (6.2.1).

The authentication of the transmitter's REST calls consist of the transmitter name and its AuthKey, which is checked against the value in the CouchDB for this transmitter.

3.14 Transmitter connections

If a transmitter wants to connect to DAPNET, the first step is to sign-in and show it's presence via the Core REST interface. This interface is also used for transmitter configuration like enabled timeslots and keep-alive polling.

3.14.1 Authentication of all HTTP-Requests in this context

All HTTP-requests issued from a transmitter have to send a valid HTTP authentication, which is checked against the CouchDB. It consists of the transmitter name and its AuthKey.

3.15 DAPNET-Proxy

Chapter 4

External Usage Workflows

4.1 General Concept of REST and Websocket-Updates

4.2 Website and App

4.2.1 Authentication

4.2.2 Calls

4.2.3 Rubrics

4.2.4 Rubrics content

4.2.5 Transmitters and Telemetry

4.2.6 Nodes

4.2.7 Users

4.2.8 MQTT consumers

4.2.9 Scripts and automated Software for DAPNET-Input

Chapter 5

Setup and Installation

5.1 Unipager

5.2 DAPNET-Proxy

5.3 DAPNET Core

5.4 Special issues for Core running in HAMCLOUD

Chapter 6

Protocol Definitions

6.1 Core REST API

6.1.1 Transmitter sign-on, configuration and heartbeat

POST /transmitter/bootstrap

```
{
  "callsign": "db0avr",
  "auth_key": "<secret>",
  "software": {
    "name": "UniPager",
    "version": "1.0.2"
  }
}
```

6.1.2 Answers to the bootstrap REST call

200 OK

```
{
  "timeslots": [true, true, false, true, ...],
  "nodes": [
    {
      "host": "node1.ampr.org",
      "port": 4000,
      "reachable": true,
      "last_seen": "2018-07-03T07:43:52.783611Z",
      "response_time": 42
    }
  ]
}
```

423 Locked

```
{
  "error": "Transmitter temporarily disabled by config."
}
```

423 Locked

```
{
  "error": "Transmitter software type not allowed due to serious bug."
}
```

6.1.3 Transmitter Heartbeat

POST /transmitter/heartbeat

```
{
  "callsign": "db0avr",
  "auth_key": "<secret>",
  "ntp_synced": true
}
```

Answers to the heartbeat REST call

```
200 OK
```

```
{
  "status": "ok"
}
```

If network wants to assign new timeslots without disconnecting (for dynamic timeslots)

```
200 OK
```

```
{
  "status": "ok",
  "timeslots": [true, true, false, ...],
  "valid_from": "2018-07-03T08:00:52.786458Z"
}
```

If network wants to initiate handover to other node

```
503 Service unavailable
```

```
{
  "error": "Node not available, switch to other node."
}
```

6.2 RabbitMQ

There are 4 exchanges on each RabbitMQ instance available:

1. dapnet.calls: Messages coming from other nodes but the node where the instance is running
2. dapnet.local_calls: Messages coming from the local node instance
3. dapnet.telemetry: Messages containing telemetry from transmitters
4. dapnet.administration: Update on the CouchDB from other Nodes to inform the Websocket microservice about changes

6.2.1 Transmitters

Valid Messages are:

dapnet.calls

dapnet.calls

6.2.2 Telemetry

6.2.3 Administration

If there is any update on the CouchDB, the other CouchDB instances are notified and updated automatically, but there is no trigger event to notify the websocket service about the change. As changes should be display immediately on the website or app, they have to be announced via websocket. In order to generate a trigger for changes in the CouchDB, the RabbitMQ exchange **dapnet.administration** is used and filled. All websocket microservices consume this exchange.

Transmitter related

New transmitter added

```
{
  "type": "transmitter",
  "action" : "added",
  "name": "db0abc",
  "data" : {
  (Complete Data dump as stored in CouchDB)
  }
}
```

Existing transmitter changed

```
{
  "type": "transmitter",
  "action" : "changed",
  "name": "db0abc",
  "data" : {
  (Complete Data dump as stored in CouchDB)
  }
}
```

Transmitter deleted

```
{
  "type": "transmitter",
  "action" : "deleted",
  "name": "db0abc"
}
```

User related

New User added

```
{
  "type": "user",
  "action" : "added",
  "name": "db1abc",
  "data" : {
  (Complete Data dump as stored in CouchDB)
  }
}
```

Existing user changed

```
{
  "type": "user",
  "action" : "changed",
  "name": "db1abc",
  "data" : {
  (Complete Data dump as stored in CouchDB)
  }
}
```

User deleted

```
{
  "type": "user",
  "action" : "deleted",
  "name": "db1abc"
}
```

Rubric related

New Rubric added

```
{
  "type": "rubric",
  "action" : "added",
  "id": "...",
  "data" : {
  (Complete Data dump as stored in CouchDB)
  }
}
```

Existing rubric changed

```
{
  "type": "user",
  "action" : "changed",
  "id": "...",
  "data" : {
  (Complete Data dump as stored in CouchDB)
  }
}
```

Rubric deleted

```
{
  "type": "user",
  "action" : "deleted",
  "id": "..."
}
```

Rubric content related

Check against CouchDB structure

New Rubric content added

```
{
  "type": "rubric_content",
  "action" : "added",
  "id": "...??",
  "data" : {
  (Complete Data dump of all ten rubric messages as stored in CouchDB)
  }
}
```

Existing rubric changed

```
{
  "type": "rubric_content",
  "action" : "changed",
  "id": "...",
  "data" : {
  (Complete Data dump of all ten rubric messages as stored in CouchDB)
  }
}
```

Rubric content deleted

```
{
  "type": "rubric_content",
  "action" : "deleted",
  "id": "..."
  "data" : {
  (Complete Data dump of all ten rubric messages as stored in CouchDB, some may be empty)
  }
}
```

6.2.4 MQTT API for third-party consumers

6.3 Telemetry

Telemetry is sent from transmitters to the RabbitMQ exchange **dapnet.telemetry** as defined in section 6.2. It is also used in the same way on the websocket API to inform the website and the app about the telemetry in real-time in section **??**.

This is sent every minute in complete. If there are changes, just a subset is sent. The n̈amek̈ey is always mandatory.

```
{
    "name": "db0acb",
    "OnAir" : true,
    "Telemetry" : {
        "ConnectionStatus": {
            "Connected" : true,
            "ConnectedtoNodeName" : "db0xyz"
            "ConnectedtoNodeIP" : "1.2.3.4"
            "ConnectedtoNodePort" : 1234
            "ConnectedSince" : "<timestamp-format>",
                "NTPSynced" : true,
            "NTPOffestMilliseconds" : 124
            "NTPServerUsedIP" : ["134.130.4.1", "12.2.3.2"]
        },
                "QueueStatus" : {
                "QueuedMessages" : {
                "Total" : 1234,
                "Prio1": 1234,
                "Prio2": 1234,
                "Prio3": 1234,
                "Prio4": 1234,
                "Prio5": 1234,
                "Prio10": 1234
                }
                },
        "PrefinedTemperatures" : {
            "Unit" : "C" | "F" | "K",

            "AirInlet" : 12.2,
            "AirOutlet" : 14.2,
            "Transmitter" : 42.2,
            "PowerAmplifier" : 45.2,
            "CPU" : 93.2,
            "PowerSupply" : 32.4
        },
        "CustomTemperatures" : {
            "Unit" : "C" | "F" | "K",
            [
                {"Value" : 12.2, "Description" : "Aircon Inlet"},
                {"Value" : 16.2, "Description" : "Aircon Outlet"},
                {"Value" : 12.3, "Description" : "Fridge Next to Programmer"}
            ],
        },
        "PowerSupply" : {
            "OnBattery": false,
            "OnEmergencyPower": false,
            "DCInputVoltage" : 12.4,
            "DCInputCurrent" : 3.23
        },
        "RFoutput" : {
            "OutputPowerForwardinWatts": 12.2,
            "OutputPowerReturninWatts" : 12.2,
            "OutputVSWR" : 1.2
        }
        "transmitterconfiguration" : {
                "ConfiguredIP": "123.4.3.2",
                "timeslots" : [true, false,..., false],
                "SoftwareType" : "Unipager" | "MMDVM" | "DAPNET-Proxy",
        "SoftwareVersion" : "v1.2.3", | "20180504" | "v2.3.4",
                "CPUHardwareType" : "Raspberry Pi 3B+"
                "RFHardware" : {
```

```
                        "C9000" : {
                                "UnipagerPowered" : true,
                                "ArduinoPADummy" : true,
                                "ArduinoPADummySettinginWatts" : 123,
                                "ArduinoPADummyPort" : "/devttyUSB0"
or
                                "RPC-CardPowered" : false,
                                "RPC-Version" : "XOS/2.23pre",
                        },
                "Raspager" : {
                                "RaspagerMod" : 13,
                                "RaspagerPower" : 63,
                                "ExternalPowerAmplifier": false,
                                "RaspagerRFVersion" : "V2"
                        },
                        "Audio" : {
                                "TXModel" : ["GM1200", "T7F", "GM340", "FREITEXT"],
                                "AudioLevelUnipager" : 83,
                                "TxDelayinMilliseconds" : 3
                        },
                        "RFM69" : {
                                "Port" : "/dev/ttyUSB0"
                        },
                        "MMDVM␣DualHS..." : {
                                "DAPNETExclusive" : true
                        }
                },
                "DAPNET_Proxy" : {
                        "ConnectionStatus" : "connected",
                        "ConnectionStatus" : "connecting",
                        "ConnectionStatus" : "disconnected"
                }
        }
}
```

6.4 Statistic and Telemetry REST API

6.5 Websocket API

The idea is to provide an API for the website and the app to display real-time information without the need of polling. A websocket server is listing to websocket connections. Authentication is done by a custom JOSN handshake. The connection might be encrypted with SSL if using the Internet or plain if using HAMNET.

6.5.1 Telemetry

The data is the same as received from the **dapnet.telemetry** exchange from the RabbitMQ instance. It is defined in section 6.3.

6.5.2 Administration

The data is the same as received from the **dapnet.administation** exchange from the RabbitMQ instance. It i defined in section 6.2.3.

6.6 CouchDB Documents and Structure

als Tabelle darstellen

6.6.1 Users

```
{
        "_id": "dl1abc",
        "password": "some hash",
    "email": "user@example.com",
    "admin": true,
    "enabled":true,
    "created_on":<DATETIME>,
    "last_change_by":"dh3wr",
    "email_valid":true
    "avatar_picture": <couchdb attachment??>
}
```

6.6.2 Nodes

```
{
        "_id" : "db0abc",
        "status" : "OFFLINE" | "ONLINE" | "ERROR",
        "last_update" : DATETIME,
    "version" : "1.2.3",
    "ip_address" : "1.2.3.4",
    "latitude" : 34.123456,
    "longitude" : -23.123456,
    "hamcloudnode" : true,
    "owners" : ["dl1abc","dh3wr","dl2ic"],
    "avatar_picture": <couchdb attachment??>
}
```

6.6.3 Transmitters

```
{
        "_id" : "db0abc",
    "auth_keys" : "hdjaskhdlj",
    "enabled" : true,
        "status" : "UNKNOWN" | "OFFLINE" | "ONLINE",
        "site_type" : "PERSONAL" | "WIDERANGE",
        "aprs_reporting_enabled" : true,
    "last_update" : "<DATETIME>",
        "ast_connect" : "<DATETIME>",
        "connected_since" : "<DATETIME>",
    "ip_address" : "1.2.3.4",
    "device_type" : "Unipager",
    "device_version" : "1.3.2",
    "latitude" : 23.123456,
    "longitude" : -31.123456
    "rf_power_watt": 12.3,
    "cable_loss_db" : 4.2,
    "antenna_gain_dbi" : 2.34,
        "antenna_agl_m" : 23.4,
        "antenna_type" : "OMNI" | "DIRECTIONAL",
        "antenna_direction" : 123.2,
    "owners" : ["dl1abc","dh3wr","dl2ic"],
    "groups" : ["dl-hh", "dl-all"],
    "frequency_MHz" : 439.9875,
    "emergency_power_available" : false,
    "infinite_emergency_power" : false,
        "emergency_power_duration_hours" : 23.0
    "antenna_pattern" : <couchDB attachment>,
    "avatar_picture" : <couchDB attachment>
}
```

6.6.4 Subscribers

check if [] is valid JSON

```
{
        "_id" : "dl1abc",
        "description" : "Peter",
        "pagers" : [
                {
```

```
                "ric" : {123456, "A"} ... "ric" : {123456, "D"},

                        "uuid" : "0023-1233-aefe-1234-3423-9812",
                "name" : "Peters Alphapoc",

                        "type" : "UNKNOWN" | "Skyper" | "AlphaPoc" | "QUIX" | "Swissphone" | "SCALL_X

                        "is_enabled" : true
        },
        ...
     ],
    "owner" : ["dh3wr", "dl1abc"]
}
```

6.6.5 Subscriber Groups

```
{
        "_id" : "ov-G01",
        "description" : "Ortverband Aachen",
        "member_subscribers" : ["dl1abc", "dh3wr"],
    "owner" : ["dh3wr", "dl1abc"],
}
```

6.6.6 Rubrics List

```
{
        "uuid" : "<UUID>"
    "number" : 14,
    "description" : "Wetter DL-HH",
    "label" : "WX DL-HH",
    "transmitter_groups" : ["dl-hh","dl-ns"],
    "transmitters" : ["db0abc"],
        "cyclic_transmit_enabled" : true,
    "cyclic_transmit_interval_minutes": 123,
        "owner" : ["dh3wr", "dl1abc"]
}
```

6.6.7 Rubric's content

<UUID> of rubric (as defined in 6.6.6)

```
{
        "uuid" : "<UUID>",
        ["content_message1",..,"content_message10"],
}
```