Adrián Yanes

OpenWeather:

A Peer-to-Peer Weather Data Transmission Protocol

Faculty of Electronics, Communications and Automation

Department of Communications and Networking (Comnet)

Thesis submitted for examination for the degree of Master of Science in Technology.

Otaniemi, Espoo, 31.08.2011

Thesis supervisor and instructor: Prof. Jörg Ott

Aalto University
School of Electrical
Engineering

Aalto University
School of Electrical Engineering

Abstract of the
Master's thesis

Author: Adrián Yanes

Title: OpenWeather: a peer-to-peer weather data transmission protocol

Date: 31.08.2011 **Language:** English **Number of pages:** 115 + 23

Faculty of Electronics, Communications and Automation
Department of Communication and Networking
Professorship: Networking Technology **Code:** S-38

Supervisor: Prof. Jörg Ott

The study of the weather is performed using instruments termed *weather stations*. These weather stations are distributed around the world, collecting the data from the different phenomena. Several weather organizations have been deploying thousands of these instruments, creating big networks to collect weather data.

These instruments are collecting the weather data and delivering it for later processing in the collections points. Nevertheless, all the methodologies used to transmit the weather data are based in protocols non adapted for this purpose. Thus, the weather stations are limited by the data formats and protocols used in them, not taking advantage of the real-time data available on them.

We research the weather instruments, their technology and their network capabilities, in order to provide a solution for the mentioned problem. OpenWeather is the protocol proposed to provide a more optimum and reliable way to transmit the weather data. We evaluate the environmental factors, such as location or bandwidth availability, in order to design a protocol adapted to the requirements established by the automatic weather stations.

A peer to peer architecture is proposed, providing a functional implementation of OpenWeather protocol. The evaluation of the protocol is executed in a real scenario, providing the hints to adapt the protocol to a common automatic weather station.

Keywords: P2P, peer to peer, weather stations, real-time, protocol standardization, embedded system, IETF, RFC

If you want to accomplish something in the world, idealism is not enough, you need to choose a method that works to achieve the goal. In other words, you need to be "pragmatic".

Richard Matthew Stallman

Preface

Before I started this thesis, my knowledge about weather stations and the technologies behind them was pretty limited. Nevertheless, in some way the weather data transmission got my attention. Probably my preference for open systems, libre software and my passion for network protocols, was the trigger to look for a topic that combines all of these areas.

During this thesis my main goal has been to show how a modern instrument as an automatic weather station, can be improved using concepts brought from open and standard technologies.

Furthermore, the impact that the weather has in our everyday deserves a deeper attention from the engineering point of view. Although the scientists are doing a great job finding new ways to understand the weather, they really need improvements in the technology field, to achieve even more better results.

OpenWeather looks for a digression. This research is looking for the attention of the scientists and the industry; for those vendors that are manufacturing instruments without a common standard, and for those scientists that are experimenting issues with the weather data acquisition. Both communities must find an agreement to standardize the methods and the technologies to transmit the weather data.

I truly think that if we start using protocols designed having in consideration the characteristics of the weather data, the result of their use will change completely the vision about what is the weather, what causes it and how it can be predicted.

Acknowledgements

Too many people have been involved directly or indirectly in this thesis, however, nothing of this would have happened without the support of my parents, Luisa Pilar and José Emilio. Thanks for your support during all these years; from the beginning you have trusted me blindly, thanks for teaching me the value of the knowledge, I love you.

Thanks to the Aalto University, and especially to professor Jörg Ott, for supervising my thesis.

Thank you to Antti Lauri from the SMEAR project & University of Helsinki, for inviting me to spend a weekend in the premises of the SMEAR project in Tampere, having the possibility to discuss with some engineers and scientists, the issues and particularities of one of the biggest weather stations in the world. Also thanks to Pasi P. Aalto and Erkky Siivola, from the University of Helsinki. Your patience and experience with weather stations have been really useful to me.

I really need to say a big thanks to Vaisala corporation, especially to Pekka Korhonen and Jing Lin, thanks for providing me with all the necessary hardware -included an amazing last generation weather station- to perform my research and for the support offered.

A big thank you to Gonzalo Mariscal, for supporting me and my constant requests during two years.

Several friends have been involved in all the OpenWeather matters, thanks to the guys of the Polyteknikkojen Radiokerho (Radio Club), to provide me the materials and the place to install the weather station. Thanks to Jose Azeredo Lima, for his support with some of the figures. Thanks to those friends that probably read this thesis even more times than me: Borja Tarraso, Sergey Vetrogronov and David Fernández, thanks for your dedication helping me, your support has been really important for me.

Finally, I want to dedicate this thesis not to one, two or three persons, I want to do it to a community, the community of the libre and open source software. To all of those developers that are working so hard to make the world a better place, without your work this thesis would never exist: respect.

Otaniemi, Espoo, August 2011.

Adrián.

Contents

Acronyms

ACL Access Control List. 113

API Application Programming Interface. viii, 15, 48, 49

APRS Automatic Position Reporting System. viii, 11, 30–32, 36

ASCII American Standard Code for Information Interchange. 4, 26, 37, 41, 101, 103, 105, 108

ASOS Automated Surface Observing System. 18

AWOS Automated Weather Observing System. 18

AWS Automatic Weather Station. viii–x, 18–32, 34, 36–41, 43–46, 48–52, 54, 56–59, 61, 65–67, 73, 80, 81, 83, 87, 89–93, 95, 98–100, 104, 107, 108, 110–113

AX.25 Link Access Protocol for Amateur Packet Radio. 30, 31

BSON Binary-JSON. 101

CPU Central processing unit. 21, 32, 43, 66

CSV Comma-Separated Values. 5, 37, 111

CWOP Citizen Weather Observer Program. ix, x, 11, 13, 30, 36, 61, 62, 73, 112

DDoS Distributed denial-of-service. 113

DNS Dynamic Name Server. 75

DoS Denial-of-service. 113

ECMWF European Centre for Medium-Range Weather Forecasts. 12

FMI Finnish Meteorological Institute. viii, 12, 15

FTP File Transfer Protocol. 35, 36, 40, 41, 44, 46, 58, 111

GDPFS Global Data-processing and Forecasting System. 36, 45, 112

GOS Global Observing System. 36, 42, 45, 49, 54, 112

GPRS General Packet Radio Service. 25, 27, 28

GSM Global System for Mobile Communications. 25, 27, 28

GTS Global Telecommunication System and WMO Information System. 36

GUI Graphical User Interface. ix, 43, 95, 98, 99, 119, 120

HTTP Hyper Transfer Text Protocol. viii, 41, 49, 58–60, 77

ICAO International Civil Aviation Organization. 38, 112

IETF Internet Engineering Task Force. 35, 113

IO in / out. 21, 41

IP Internet Protocol. 46, 60, 71, 75, 89, 102, 105, 108

ISO International Standard Organization. 61, 67, 76

JSON JavaScript Object Notation. x, 47, 49, 51, 54, 58, 67–69, 71, 78, 81, 88, 91, 95, 96, 99, 101

kB Kilobyte. 21, 23, 110

kbit kilobits. 23, 73, 93, 94

MB Megabyte. 22, 23

Mbits Megabits. 23, 73

METAR Meteorological Service For International Air Navigation. 37, 38

MHz Megahertz. 21

NAT Network address translation. 52, 75

NMEA-0183 National Marine Electronics Association 0183. 4, 26, 37

NOAA National Oceanic and Atmospheric Administration. viii, 12, 19, 73, 112

NTP Network Time Protocol. 77, 101

OS Operating System. xi, 50, 56, 57, 94

P2P Peer to peer. 48–52, 55–60, 71, 72, 75, 76, 88, 91, 95, 98, 110, 112

PROM Programmable Read-Only Memory. 26

PTH Pressure, Temperature, Humidity. x, 44

PTU Pressure, Temperature and Humidity. x, 80, 81, 84, 88, 110

RAM Random-access memory. 21

RFC Request for Comments. 35, 36, 51, 52, 56, 61, 67, 76, 98

RS-232 Recommended Standard 232. 4, 23, 33, 34, 93

RS-422 Recommended Standard 422. viii, 4, 23, 24

RS-485 Recommended Standard 485. 4, 23

RTT Round-trip time. 94

SDI-12 Serial Data Interface at 1200 Baud. 4, 26

SHA Secure Hash Algorithm. x, 61, 62

SI Système international d'unités - International System of Units. 10, 11

SMB Server Message Block. 40, 41, 46

SOA Service-oriented architecture. ix, 47, 48, 52, 53, 57, 64

TCP Transmission Control Protocol. xi, 46, 63, 75, 95, 98, 102–110, 113

TLS Transport Layer Security. 113

TSV Tab-separated values. 5, 37

UML Unified Modeling Language. ix, 97, 100

UMTS Universal Mobile Telecommunications System. 25, 27, 28

URI Uniform Resource Identifier. 61

URL Uniform Resource Locator. 61, 62

USB Universal Serial Bus. 4, 23, 27, 34

UTC Coordinated Universal Time. 76

UTF Universal Character Set - Transformation Format. 67

UTM Universal Transverse Mercator. 74, 92

WMO World Meteorological Organization. 11, 12, 16, 18–21, 31, 34, 36, 37, 42, 49, 62, 79, 112, 113

XML eXtensible Markup Language. 69

List of Figures

List of Tables

Chapter 1

Introduction

From the beginning of the time, the weather has been an important factor in the human life. Its impact of it in our everyday, gives as result that during centuries we have been trying to understand and predict it as much as possible.

We all are familiar with some weather concepts, because it really has an impact on how we proceed in our life. For instance, it is really common to check the forecast before we start some outdoor activity or even without any special reason, only to know which kind of atmospherical conditions we are going to experiment the following days; this is possible by the meteorology.

The science of meteorology takes the role of the scientific study of the atmosphere, this implies to know certain phenomena behave and which kind of predictions can be made based on them, and of course the impact of them in our lives. To achieve this goal, the science of meteorology has been developing different techniques and methods to measure and collect the necessary data to make these predictions. The human history is full of inventions of different instruments designed to make this possible. In the past, these instruments were based just in mechanical principles with a high limited accuracy. Nowadays, we can find a huge set of alternatives based in digital mechanisms which allow us to predict the weather and understand the atmosphere phenomena with high precision and accuracy; giving us a better knowledge of our environment and at the end making our life easier. Even if in the last years the transition from pure mechanical instruments to the digital technology has been really fast, certain parts still have not been renovated or are under development.

The purpose of this thesis is to study some possible improvements of these parts, more specifically in the protocols used to transmit the weather data collected in different instruments to the places in which the data is processed for its broadcasting.

When I started researching some weather instruments their technology caught my attention, mainly in all the aspects of measuring a phenomenon with precision and feasibility, and at the same time I was confused about how the protocols used in them are full of legacy and low efficiency, in terms of data transmission and real time data availability.

Nowadays, we have functional and reliable weather data systems to study the different phenomena, however, the potential of the real time data gets blocked by the methods used on the weather data collection. Even if at the end, we have the capability to process and interpret the data, a huge amount of effort is needed to make this happen, due to the methods and technology used for the collection. This fact got my attention when I was trying to find some research area in which the protocols and the information theory could help to make this process more useful, faster and reliable.

After understanding and verifying how the weather instruments work, I found really important to ask some meteorological scientists what the state of the art is, concerning atmosphere data collection. I had the great opportunity to visit the SMEAR [40] project for a weekend, study how the data is collected, transmitted, processed and stored. At the same time, the scientists that are using this data to study the atmosphere, could confirmed that some huge improvements can be made in order to improve the data transmission (this affirmation is mostly based on the technical issues that they are experimenting in their research).

This fact and the interest in peer to peer protocols and the real data transmission, were the final trigger to start this thesis and try to find a possible solution to improve the speed and reliability of the weather data transmission.

Applying the concept of "peer" to any group of sensors which are collecting weather data and assuming that also the scientist is a peer that fetches and exchanges data with other scientists (also considered peers), it was the foundation to research, looking for a protocol that allows the weather stations to exchange and route data with other weather stations and at the same time provides a infrastructure to access data collected in real time.

1.1 Background

A weather instrument is an artifact which main task consists in the data collection from one to multiple atmosphere phenomena. These instruments are designed thinking in a specific use case: a particular natural phenomenon, and at the same time with a well defined goal: the collection of data that helps to study and predict phenomenon.

Nowadays, we can find several solutions to achieve this goal. Science has found different ways to measure the same phenomenon in different ways and with different reliability. However, common techniques are used around world to measure the same phenomenon. Sometimes the reasons for using a certain technique can go from the complexity and reliability of it, to the cost of it. The standard way to measure a particular phenomenon is developing a specific instrument (also named *sensor*) for it, this instrument is able to measure and understand it better.

Some popular concept to refer these sensors is *"weather stations"*, nevertheless, this term is not correct at all due to the amount of instruments in a weather station can be barely different compare with other vendors' instruments. Notwithstanding, this term is accepted as common to refer the group of sensors used to collect weather data (**we will use this concept from now on to refer to a group of sensors creating an identity named "weather station"**).

The following list[1] enumerates some common instruments in a weather station:

- Thermometer for measuring air and sea surface temperature
- Barometer for measuring atmospheric pressure
- Hygrometer for measuring humidity
- Anemometer for measuring wind speed
- Wind vane for measuring wind direction
- Rain gauge for measuring precipitation
- Disdrometer for measuring drop size distribution
- Transmissometer for measuring visibility
- Ceiling projector for measuring cloud ceiling

All of these instruments have a defined mechanism to measure a specific phenomenon and collect the data to be processed later. These instruments or sensors are applying some physic principle to get this data and converted it into digital information for future transmission.

After the data is collected in the instrument[2], it is transmitted to some organization, such as a meteorological institute, to interpret the data and get some conclusions concerning the current status of the weather and future predictions.

With information collected in different instruments around the world, we can know the status of the weather and how it will be in the future, all of these weather stations around the world are "weather data pickers", and the success of the final weather prediction resides in the efficiency and reliability in which this data is collected, transmitted and processed.

For a while, all of this process has been optimized in several ways, like creating better instruments, infrastructures and organizations focused only in this field. However, the standardization process only impacted on measure techniques and

[1]These sensors are an example based on the market's offer, notwithstanding the amount of different sensors to measure the phenomena increases really fast, being difficult to track them all.

[2]A device named datalogger is involved in this process.

data units, putting in a secondary plane, other parts of the process such as communication protocols, digital interfaces used, etc.

1.2 Problem statement

The nature of the data collected in the weather stations involves to place them in different locations around the world. It is creating a trickier scenario for the data collection. Multiple weather stations are located in inaccessible places, but their location is mandatory to deploy feasible models for weather predictions. Commonly, these instruments are placed in different locations in which sometimes the environment is not friendly at all to be combined with digital technology; some examples of these are isolated places such as mountains, roads or forests. These environmental conditions bring issues as lower **bandwidth availability**, **difficulties to get enough energy 24x365** and the variable weather conditions in which some instruments are subdue with the implications of these in terms of lifetime.

The industry has been developing different instruments to achieve this objective and avoid the mentioned issues. However, the main effort has been to develop instruments with high accuracy, low power consumption, resistance, and small size; **resting importance to the methods used in the transmission efficiency of the data collected**.

It is a fact that these instruments are getting more complex, reliable and tiny with the time. Nevertheless, there is non defined standard to transmit and process the data collected from the instruments to the locations in which this data is useful (meteorological organizations, computation centers, databases, etc). The common practice is that **the vendors choose their own data format / protocol for this purpose**, and depending on the manufacturer the instrument formats and transfers the information using some standard for peripheral devices such as **Recommended Standard 232 (RS-232)**, **RS-422**, **Recommended Standard 485 (RS-485)**, or **Universal Serial Bus (USB)**. At the same time one of the following serial communications protocol is commonly used to transmit the collected data:

- RAW American Standard Code for Information Interchange (ASCII)[3]
- Serial Data Interface at 1200 Baud (SDI-12)
- National Marine Electronics Association 0183 (NMEA-0183)

These are the standards that the industry established to transmit the data from the instruments that they are manufacturing. However, **the mentioned standards are generic** for serial communications data transmission, **without any direct or indirect relation or adaptation to the weather data**. That means that the

[3]The concept of RAW refers to a serial communication in which is not used any special data format, just data formatted using ASCII as character-encoding scheme.

industry chooses only to take care of the data transmission for their own instruments, creating their own data formats, timings of transmission, data definition and so on. This common practice between vendors is causing the non-existence of an international standard and by default the incompatibility of these instruments with others brands, plus the possibility to combine the output data of different instruments from different brands.

The use of a non-adapted protocol for the data transmission decreases the efficiency and the possibility of a easy manipulation of the data. Even assuming that the industry chose this way to transmit the data based in the mainstream digital solutions, in terms of serial communications, **is possible and feasible to deploy a standard to format and transmit this information in a more optimized and reliable way**; this will imply the participation of different vendors to standardize this format. The process of standardization is a well-known practice in different fields of the industry due to the advantages that it brings in terms of compatibility, interoperability, safety, repeatability, or quality; at the same time standardization is supported in multiple cases (depending of the industry) for international laws.

Thus, the choice made by the industry makes the optimization of the data manipulation really painful, in addition, it is rare to have one point of weather collection with only one brand of instruments. It entails that at the end of the data transmission, the data collection scenario must be combined with different software from different vendors and different parameters. This makes the process of the weather data collection even more arduous, since the original format in which the data is transmitted is completely useless and must be converted to be combined with other data.

The absence of a standard is forcing to pre-process the weather data after its transmission, even if this is something needed in any network data transmission at some point; the format used in the process can save a lot of CPU cycles, memory and bandwidth. This absence forces the weather data collection centers to convert the data in a useful format for future computation, and this is happening through custom software developed by the vendor's instruments or in some cases, custom software developed by the organization itself. As an example of this, the SMEAR project[40] has developed several parsers and scripts to manipulate this data before it can be processed, wasting time and resources that can be easily solve through a standardization.

We needed to highlight that most of the end users of this software are scientists that need the data to get some conclusions about the weather. It means that at the end of the data collection workflow, it is manipulated through software focused in mathematics computation like MathLab, which does not support any data format used by the weather instruments, forcing the scientists to have the data in dummy formats as Comma-Separated Values (CSV), Tab-separated values (TSV), or just plain text, to be able to use it.

The following figure shows an example of how the data is processed and where the conflictive points are:

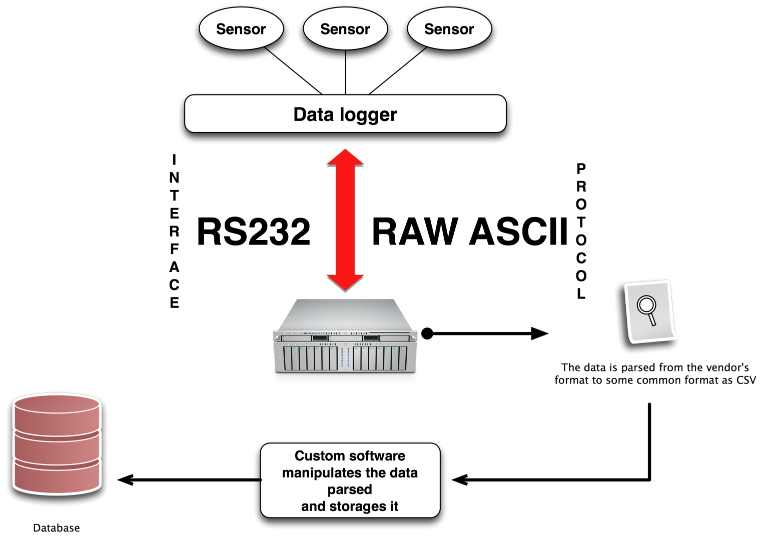

Sensor
Sensor
Sensor
Data logger
INTERFACE
RS232
RAW ASCII
PROTOCOL
The data is parsed from the vendor's format to some common format as CSV
Custom software manipulates the data parsed and storages it
Database

Figure 1.1: Common scenario to collect, transmit, manipulate and storage data in a weather station.

As it is observable in the figure 1.1 the parsing and the implication of specific software in the process, is causing the implementation of unnecessary subprocess as parsing, packaging and data conversion. At the same time, the process described is decreasing the possibilities to have easily accessible information in real time.

Though meteorology needs big amounts of data collected in different places and the analysis of this data is made using different times frequencies, we can not ignore how useful the data of our environment can be if its accessible in real time. As example of this can be that industry has been focusing, in the last years, on developing technologies that allow the users to get information on demand and in real time, this is supported by the principle that with more detailed and updated information we can act with more precision and feasibility.

The absence of a weather data transmission protocol is impeding us to know how powerful can be the combination of multiple weather data sources in real time. It can provide the mechanisms to deploy different models and perform analysis of the data based in the real current situation of the weather, regardless the location or brand of the weather station. Even if nowadays we have enough precision understanding the atmosphere phenomena to predict future weather conditions, we still need to advance in the physics to deeper understand the impact of these phenomena and how they work, providing us a better knowledge of our environment, and at the end, improving our quality of life.

Currently the weather data information is collected in real-time (because the sensors are taking samples of the current environment), notwithstanding the technology used in the process of the data transmission does not take this in consideration, non using a standardized and optimized process for this specific data.

This fact is, at some point, blocking the possibility to explore how useful this data could be for us, but it is not accessible at all because engineering issues. On the other hand, the absence of a common protocol even to exchange non real-time data, generates a big amount of issues in terms of data combination and comparison; causing several problems of incompatibility between the organizations focus in the weather study, and forcing the use of extra resources in operations such as data normalization (something that can be fixed through a common data format).

1.3 Research objectives and scope

The purpose of this thesis is to identify the points in the weather data transmission in which the process is not optimized according to the nature of the data. At the same time, a protocol is proposed as proof of concept, showing how the weather data transmission can be improved without too much effort from the vendor's side.

The foundation of this research is to find a path having in mind a real scenario as the SMEAR project[40], in which the process of the data transmission and manipulation can be improved offering new use cases for the data, in terms of real time acquisition, manipulation and storage.

The following points identify the approach of the research briefly:

- Identify the blocker points in data transmission concerns
- Study how the weather data transmission and manipulation can be improved
- Develop a protocol prototype specification that provides an improvement in the current scenario

As final objective the author is looking forward to motivate the vendors to start a standardization process to improve the mentioned problems. Based on the opinions shared with atmospherical scientists, the absence of accessible real-time comes from the engineering side, and it is needed to develop some technology that ensures an easy a feasible method to access this data.

1.4 Motivations

After working with weather instruments, understanding how they work and how they transmit the data, I noticed the issues previously exposed. However, my vision was not enough to be sure about the key-problem treated (because it was only based in end-user weather instruments). When I had the opportunity to see how the biggest weather station in the world (concerning gas measurements) is fetching, transmitting and manipulating data, and at the same time, talk with some scientists about my suspicions were confirmed by them: this process can be optimized.

In addition, the absence of a standard in something so important as the weather data transmission, gave me enough reasons to perform this research, based in the idea that maybe some conclusions can be directly applied to the industry.

Finally, my vision about certain user rights is implied in this research as well. I am convinced that a society informed has always more possibilities to have a better quality of life. In the last years several misunderstandings and confusions have been happening concerning the current situation of the climate in our planet. Unfortunately, the absence of accessible and understandable information generates confusion in our society. Although this is an issue in which the science has been leading from the beginning of the time, I support that the improvement of the methods used in the science, are always helping us to make the information more accessible, hence to have the possibility to spread the knowledge with less effort. In this case, I think this research can contribute to improve how we transmit and understand the weather data paradigm. It is a good moral reason to me to perform this study.

History shows how the proper use of technologies adapted to specific scenarios, promotes the advance of linear sciences as Maths or Physics, and always these new findings are supported by new technologies. To find these new technologies, it is needed to analyze from the engineering point of view, which things can be improved and how; this philosophy turns this thesis in an exercise to find how a science as meteorology can benefit from communications technologies around it if they are optimized for its needs.

1.5 Outline of the thesis

This thesis is structured as follows: the second chapter gives a general overview about how the weather data collection is structured, and which organizations are interacting on this activity. The third chapter explains briefly how a weather instrument works and what kind of technologies are involved in the process, after that, it is analyzed how the meteorological networks composed by these instruments work. The fourth exposes the technical deficiencies found by the author on the weather data transmission. In chapter five the OpenWeather protocol is presented, a prototype protocol developed by the author, adapted to the needs exposed in the previous chapters. Chapter six specifies from the technical point of view how the protocol works, its operations and architecture, accompanied with justification of the technical decisions taken on the thesis, concerning its implementation. Chapter seven evaluates the implementation of the protocol in a real scenario based in a specific weather instrument. Finally, chapter eight summarizes the conclusions of this thesis.

Chapter 2

The impact of the weather data

Even if it is obvious for all of us, weather is one of the most important factors of the environment, with a high impact in our life. At the same time most of us are not familiar with the repercussions of the weather, what is causing different phenomena and the implications of them. Finally, our needs concerning the weather are limited by the availability of the data that is given to us. The role of the weather forecast broadcasting resides in different organizations. However, the advantages of the technology are bringing us the capability to have a more frequent and reliable access. The following sections analyzes how the weather data is spread and in which points of its diffusion can be improved.

2.1 Weather data collection and diffusion

Depending of the region of the world, we can find more or less geographical locations in which a weather station has been placed to collect information about different phenomena. It is important to clarify they are several categories of phenomena with different needs in terms of data collection requirements. In addition, we have different units and time frequencies to make this data useful.

Fortunately, nowadays, most of the known phenomena have a solid basement of understanding, meaning this that we can measure them and get some conclusions and to act in consequence. The Système international d'unités - International System of Units (SI) is used as the recognized standard of units for these measurements[1].

The figure 2.1 shows the scenario abstracting the data to a generic input:

[1]Some countries as Burma, Liberia, the United States or the United Kingdom, have other local standards coexisting with the SI. This implies some adaptions concerning the weather data. Due to the local units it is necessary to include unit conversions in the data manipulation process.

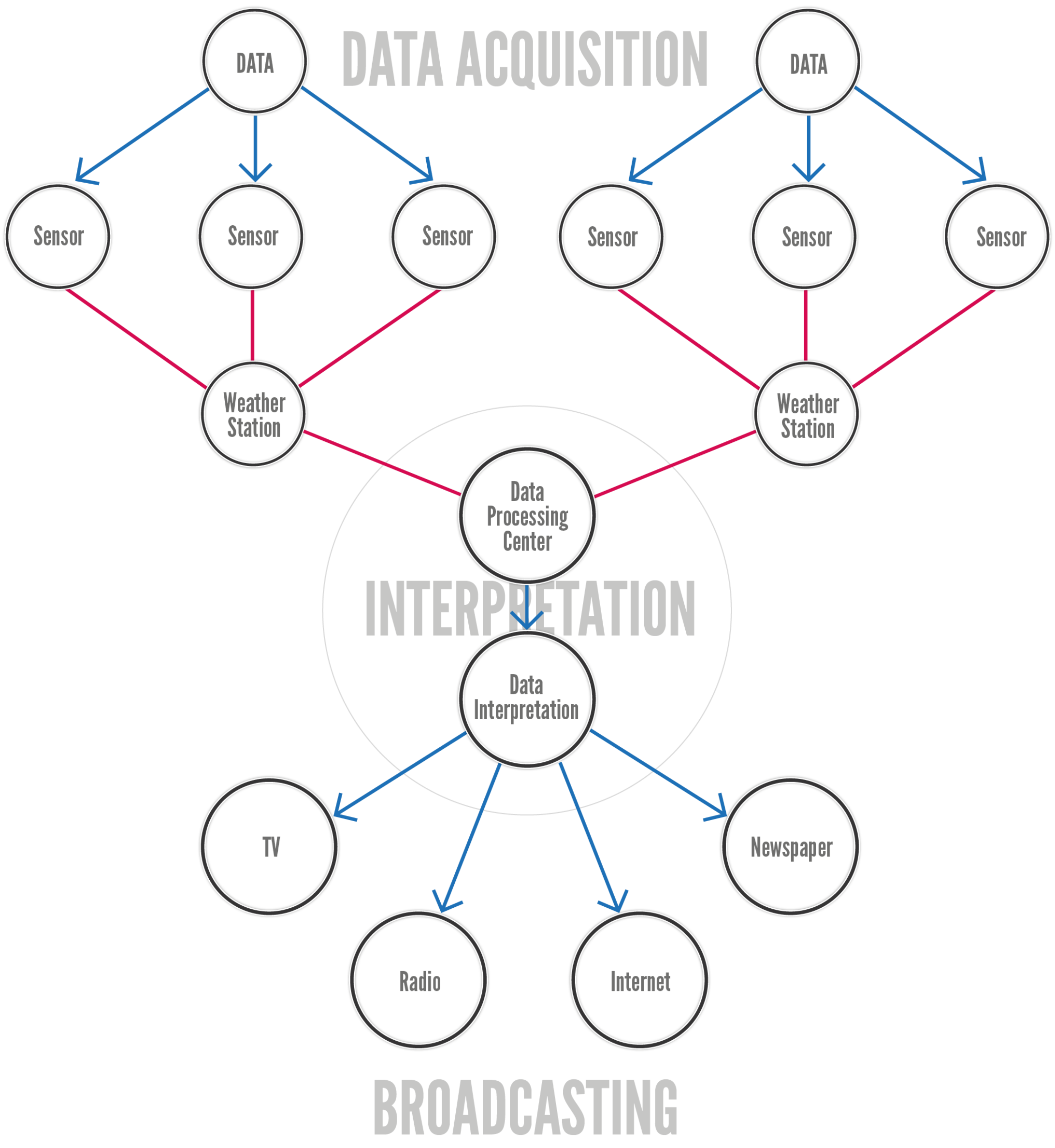


Figure 2.1: Layers abstracted in the weather collection data workflow.

As we can see the scenario gives as an abstract input of data from the different environmental phenomena. After that, the data is sent to the data processing center (commonly a governmental & scientist organizations). At the end, the data is interpreted and the conclusions are spread. The Physics are giving us the possibility to understand these phenomena based in the observation and correlation of them; for this it is needed to establish direct dependencies between the phenomena.

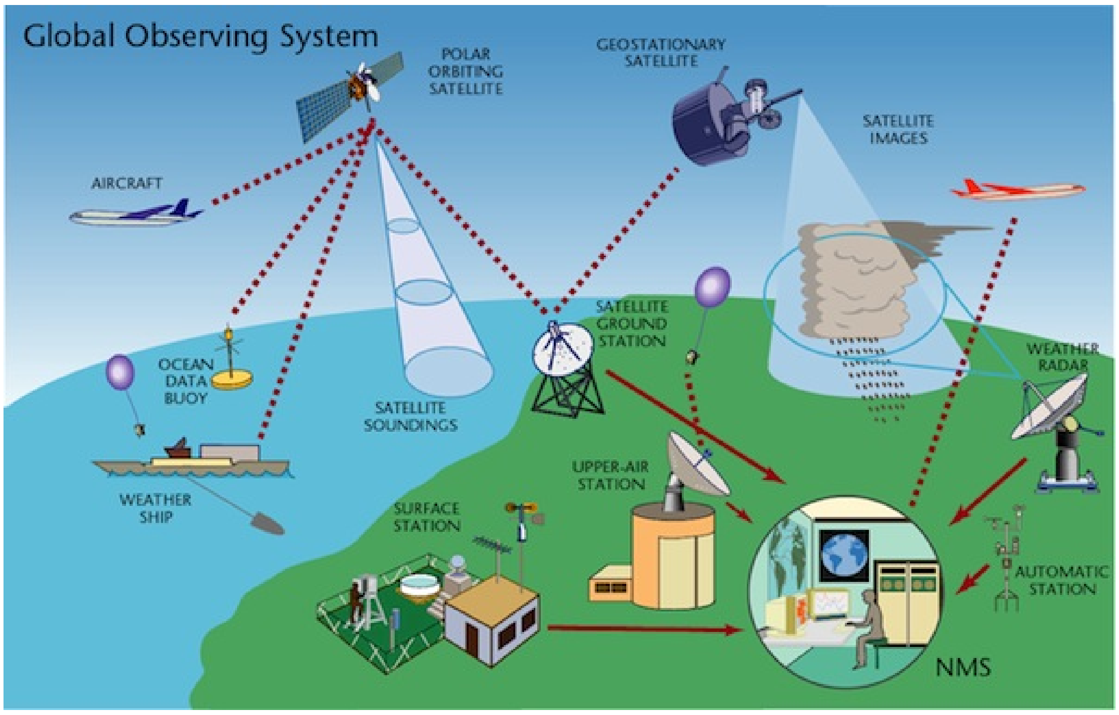


Figure 2.2: Weather data collection workflow. World Climate Data and Monitoring Programme.[3]

Commonly, we can find several governmental and scientist organizations around the world, focused in the weather data collection. As example of this,in Finland we have the Finnish Meteorological Institute(FMI) [23], or different example can be a worldwide organizations such as the World Meteorological Organization (WMO)[44], in charge of the coordination of the exchange and collection of weather data between organizations around the world. These organizations are the official source of information for weather data. Even so, they are not the only ones.

Thousands of individuals are helping with the weather data collection as well. Those individuals in possession of some weather instruments can collaborate transmitting the data to some governmental organization, for instance the program CWOP [18] has over **20,000 members in 149 countries**. This is possible using technologies like APRS [24] system, which is mentioned in CWOP website[18] as the following:

"The Automatic Position Reporting System (APRS) is a part of ham radio that provides an ideal way for weather station operators to distribute their weather data much further than the regions within their transmitter range. APRS was originally intended for position information data but actually provides a means for automatic transmission of all sorts of digital data. This is especially true now that the original

[3]The World Climate Data and Monitoring Programme (WCDMP) is a programme of the World Climate Programme that facilitates the effective collection and management of climate data and the monitoring of the global climate system, including the detection and assessment of climate variability and changes.[44]

APRS packet radio concept has been enhanced to include the capabilities of the Internet. The reporting of citizen weather data is a particularly useful application of the APRS Internet Service (APRS-IS)."

2.1.1 Governmental organizations

Denominated as meteorological institutes or meteorological agencies, it is possible to find a big group of organizations around the world, which purpose is to study the weather. Almost all of these organizations are funded by the governments, moreover of these state and local organizations, other country-region organizations exist to coordinate the study of the weather in a bigger extension area. As an example, the FMI[23] is in charge of studying the weather in the region of Finland. At the same time the FMI is member of the European Centre for Medium-Range Weather Forecasts (ECMWF)[19], organization in charge *"to provide operational medium- and extended-range forecasts and a state-of-the-art super-computing facility for scientific research."* .The same scenario can be found in different continents as America with organizations as NOAA[32].

These worldwide organizations are creating the infrastructure to collect the weather data around the world. It is necessary to highlight that the study of the weather is an expensive activity, involving a big amount of resources such as high-tech instruments, installation of these instruments in different locations (with the extra cost that it implies) and use of computation centers to evaluate the data. Due to these facts, we can find that the amount of weather stations around the world and the effort or size of these organizations can vary significantly depending of the economy of the region. This means that the weather infrastructure in the occidental world is well designed, implemented and functional. However, in other areas like Africa, the amount of available weather stations decrease for economical reasons. In addition, and due to the nature of the weather, organizations like NOAA and ECMWF are installing weather collection points outside their official operation areas[4], thus getting better samples to evaluate the global weather conditions.

These state-region organizations have a huge cooperation between them. Scientists are pretty conscious about the need to get samples of weather data from different regions to evaluate it, thus, they are fomenting the cooperation of the weather data exchange. The WMO defined the proceedings of measurement for meteorological variables[46], providing a common basement to perform the measurements related with the weather. Furthermore, the WMO is conscious about the issue of data exchange, in chapter four the process of standardization that WMO is supporting and the issues of it are analyzed deeply.

[4]Both organizations are restricted to America and Europe, nevertheless, these organizations have permission to place collection points out of their area to improve the quality of the studies and to encourage the international cooperation.

2.1.2 Corporations

As it was mentioned previously, weather has a big impact in our life. It implies that not only practical advantages can be extracted from the study of it, also the study of the weather is generating a big range of economical activities.

Industries like construction or military, have even more interest in know which phenomena are occurring and the future predictions of them. This interest have fomented a whole parallel industry of services of weather data reports.

At the same time, some professional forecast services have appeared as an alternative for independent studies in particular regions of the world. Although this economical activity is mainly deployed by private corporations some governmental organizations are offering also private services.

2.1.3 Individuals

The program CWOP mentioned in the section 2.1, is a perfect example about how individuals can help to collect and to study the weather data. Furthermore, non official programmes have been appearing around the world; using the Internet as foundation, different communities of weather observers are contributing to create individual networks of data exchange, in which a user can access the data of different weather stations around the world.

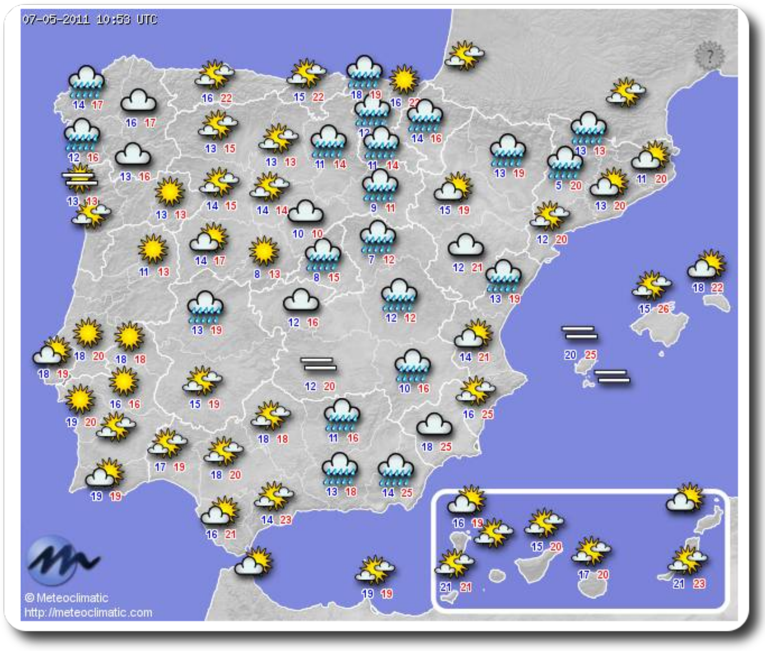


Figure 2.3: Meteoclimat screenshot showing weather forecasts.[5]

Meteoclimatic[29], is a good example of this:" a big network of automatic non professional weather stations", in which hundreds of users share the data collected from their weather stations without any commercial purpose. Often, these communities share efforts with governmental organizations in programs as CWOP, however, the turn up of theses communities are supported by the demand of the users to have a system in which their data is useful for other individuals, and at the same time give them some independency from governmental organizations, in terms of data availability.

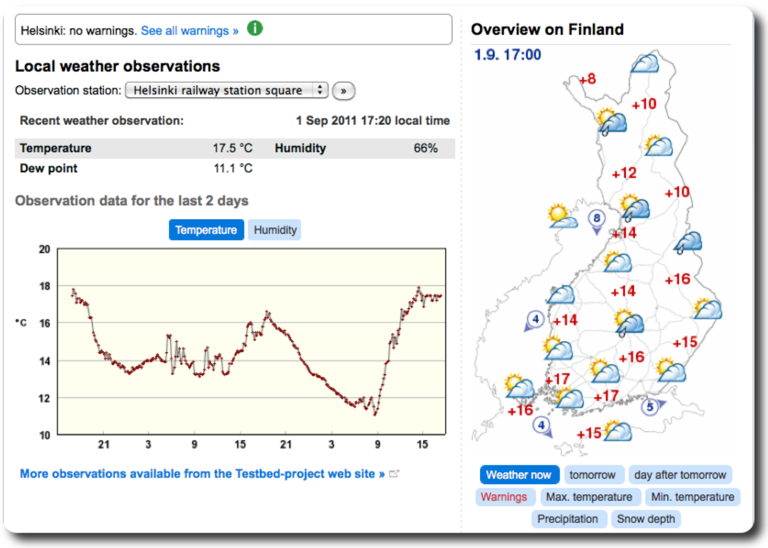
2.1.4 Weather data publication

The previous sections mention which organizations are involved on the process of data collection. However, the process does not end here; after the collection and evaluation of the data, the final step is to spread and make it useful. The implications of the broadcasting concerning the weather forecast are multiple and they are out of the scope of this thesis. Even so, the spreading of the data is limited for the protocols used in the acquisition of it. As mentioned in section 2.1.3, some communities of individuals appeared, taking the role of data availability disposal for the end user. Proving this the fact that the way in which the information is managed by the governmental and private organizations, sometimes does not fit with the end user's wishes.

In the past, the weather forecast was delivered through traditional methods as newspapers, radio and TV. Nevertheless, nowadays the Internet has taken this role in several aspects. Almost, all the governmental weather organizations mentioned in this chapter have a web site in which they publish -in different quantities and formats-, the information collected and extracted from their meteorological networks. Although traditional media still report the daily forecast, the tendency points to the Internet as the future mainstream channel of this information.

In addition, other commercial web sites offer this information partially free of charge. This practice caused the appearance of several sites offering API services to fetch weather data, giving the possibility to the developers to get some storage data to perform some operations. Due to this API availability, some organizations non related directly with the weather data collection workflow, have published some web sites that are exposing data fetched from different APIs and providing a different range of alternatives to the users.

[5]This data is collected by individuals that have installed a specify software in their computers to send the data to Meteoclimatic servers.



Helsinki: no warnings. See all warnings »

Local weather observations

Observation station: Helsinki railway station square

Recent weather observation: 1 Sep 2011 17:20 local time

Temperature	17.5 °C	Humidity	66%
Dew point	11.1 °C		

Observation data for the last 2 days

More observations available from the Testbed-project web site »

Overview on Finland

1.9. 17:00

Figure 2.4: FMI website [23] spreading local weather observations.

The author could not find any API offering the capability to connect directly to the weather instruments to fetch RAW data streams; all the APIs available are offering pre-processed data.

2.2 Summary

In this chapter we have given general background information in order to make the scope of the thesis more familiar, in terms of which organizations are in charge of the weather collection and the structure and collaboration between them. Also, it has been analyzed how different organizations of the same field coexist.

We discussed how the same activity is performed in different layers, being involved in the process from official organizations to individuals. Some schemas have been presented, giving a global vision about how the weather data workflow works.

We know now that there is even a global organization named WMO. This organization is only dictating some guidelines to perform the measurements. The next chapter introduces a general overview of a weather instrument, to understand how it works, its technologies and limitations.

In the next chapter some concepts and scenarios are explained to understand how a weather instrument works, the technologies that are conforming it, and giving us a global vision of the technologies to have in consideration when we are implementing a protocol for a weather instrument.

Chapter 3

Infrastructure for the weather data

History is full of attempts to understand the weather. From the very beginning, humans have been focusing their attention in the weather, putting a lot of effort trying to understand and predict it. The first treatise concerning weather observations was *Meteorologica*, written by Aristotle (340 B.C.). Despite of this, *"the birth of meteorology as a genuine natural science did not take place until the invention of weather instruments, such as the thermometer at the end of the sixteenth century, the barometer (for measuring air pressure) in 1643, and the hygrometer (for measuring humidity) in the late 1700s"*[1]. It was with the invention of the telegraph, in 1843, when the weather observations started to be useful owing to the capability to transmit the weather reports to different locations. Since this time elapsed, the industry has been developing and improving the weather instruments to achieve better measurements. Furthermore, the networks for weather data collection have been maturing. This chapter introduces the technology that is composing a modern weather instrument, its role in the weather's collection infrastructure and shows us some concepts to understand the conflicts of this setup exposed in chapter four.

3.1 A meteorological instrument

The purpose of a weather instrument is to measure a particular phenomenon under certain conditions, to collect some data that can be processed to obtain some conclusions (in terms of understanding and predictability). The success of the prediction and understanding comes supported by the accuracy that these instruments can provide. The industry has been creating new instruments based on new techniques discovered in Physics, to measure the phenomena; in addition, the advance of the digital technology, is providing to the physicians a great scenario in which physical principles can be combined easily with digital technology, producing as result modern instruments with the ability to transform the result of these physical principles in digital data.

Despite their size and appearance, the weather instruments are complex artifacts. The materials used to build them are a combination between plastic and metal, this combination provides the necessary robustness to place the weather instruments at isolated places with all kind of degradation conditions. Furthermore, these instruments must have a low power consumption in order to fit the requirements of their locations. That forces the manufacturers to use more tiny and efficient technologies for measuring the phenomena without sacrificing energy and accuracy.

It is not possible to discuss all these instruments in this thesis. For this reason the following subsections of this chapter are focused on automatic weather stations(AWSes). The WMO defines an AWS as: *meteorological station at which observations are made and transmitted automatically*[46], at the same time this concept comes with other nuances as Automated Weather Observing System (AWOS) and Automated Surface Observing System (ASOS): *a combined system of instruments, interfaces and processing and transmission units is usually called an automated weather observing system AWOS or automated surface observing system ASOS. It has become common practice to refer to such a system as an AWS.*

The focus on the AWSes is supported by the popularity of these weather stations as main tools to measure the weather. **The author considers more useful to focus on this technology because a wide range of AWSes is available for the end-non professional user**; meaning this that is possible to experiment with a new protocol using this scenario without affecting the current setups used for scientific purposes. In addition, later migration of the protocol to professional instruments should not be difficult because the manufacturers are using mostly the same technologies in the data transmission interfaces for both brands (professional and end-user).

3.1.1 Industrial design

Depending of the type of phenomenon to measure, the physical principle needed will require an instrument with certain sizes, materials and lifetime. It is rarely possible to measure multiple phenomena with the same instrument, this fact causes the creation of instruments focused only on one phenomenon [1] and even in only one specify and tiny part of it.

The industrial design of an instrument is one of the keys for the success of the observations; the ability to put available the required technical conditions to perform the measurement through a digital interface, reside on it. To avoid conflicts in the study of the phenomenon, the materials should be chosen very carefully based on a complex equation between: robustness, durability, impact, impact assessment, etc. Furthermore, the shapes and sizes depend on the environment in which the

[1]We refer here to high-tech and professional instruments for scientific purposes. It is possible to find several sensors giving an output for different phenomena in one instrument. However, this is not common in the instruments used for scientist observations; at the same time this configuration should be considered as a weather station not as an "individual" weather instrument.

instrument is going to be placed and the requirements needed for the physical principle used.

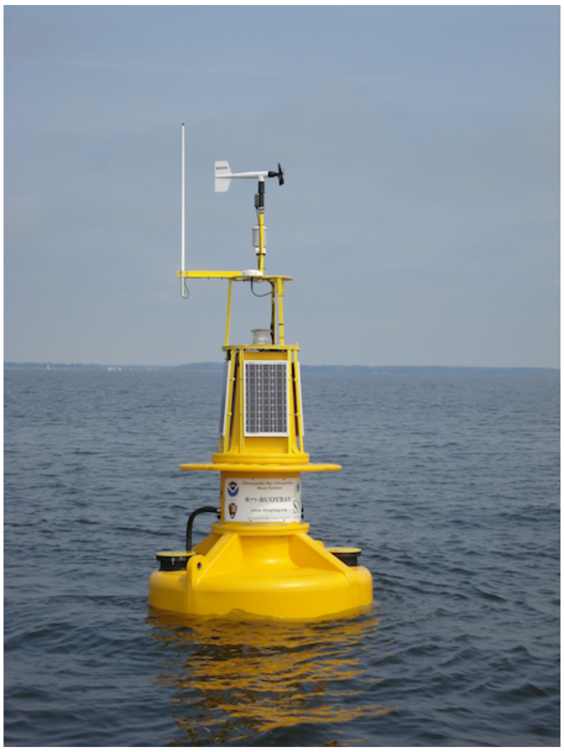


Figure 3.1: NOAA weather buoy [32], example of a complex an robustness AWS.

We can find in the market dozens of instruments for the same purpose, using in some cases the same principles to measure the phenomenon and even with some strong differences concerning the industrial design. Though, the instruments from different manufacturers have similar dimensions and they are build with similar materials, there is non available standard concerning all these characteristics, only some general guidelines are provided by the WMO[44] suggesting dimensions and sizes for some instruments, an example of this recommendation is the following:

Wind-measuring systems can be designed in many different ways; [...] The first system consists of an anemometer with a response length of 5 m, a pulse generator that generates pulses at a frequency proportional to the rotation rate of the anemometer (preferably several pulses per rotation), a counting device that counts the pulses at intervals of 0.25 s, and a microprocessor that computes averages and standard deviation over 10 min intervals on the basis of 0.25 s samples.[46]

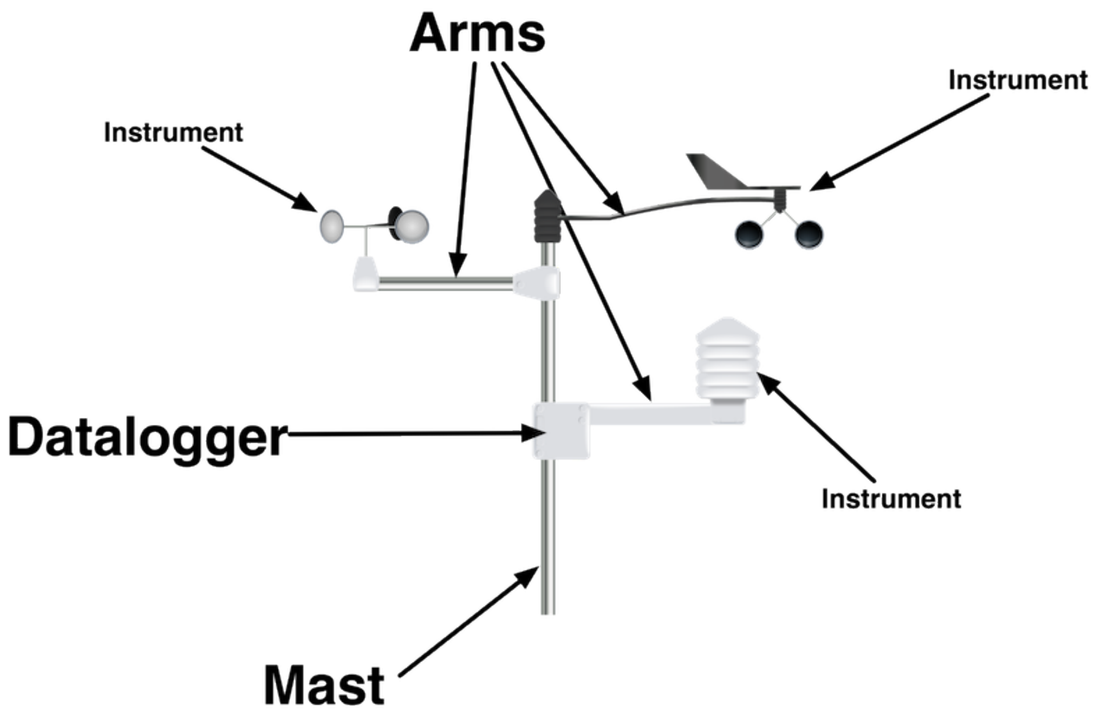


Figure 3.2: Generic AWS with different instruments and materials combination.

The figure 3.2 shows a generic schema in which we can see different combinations of materials as plastic and metal, at the same time the instruments are placed in different heights due to technical requirements for the techniques used to perform the measurements.

Most of the instruments available at the market are the result of the coordination between the requirements requested by the physicists and the possibilities that the technology developed by the industry. Notwithstanding, the instruments industry and their industrial design, is something really big and complex and it is out of the scope of the thesis. Furthermore, we need to be conscious about the industrial design of the instruments, because it is strong-linked to the electronics that they can house, conditioning this the digital interfaces for data transmission that we can install in them.

3.1.2 Electronics and data handling

The electronics of a weather instrument are barely different irrespective of the phenomenon to measure. The industry is producing a wide range of instruments with a complete different set of sensors. Nevertheless, as embedded systems, all these instruments have a common need to conform these type of systems. The

WMO gives again some general guidelines with respect to electronics and weather instruments. The following paragraphs summarize them.

CPU

As other electronic device in charge of process data, an AWS has a Central processing unit (CPU) running at clock frequency of a few Megahertz (MHz). This CPU is microprocessor based with 8-bit wide.[2] Despite the low bit wide of these microprocessors, an AWS does not needed more calculation power because the amount of data generated by the sensors will be rarely up of 1 Kilobyte (kB), meaning this that frequencies oscillating between 8-33 MHz will fit perfectly in the requirements to process the data.

Volatile Memory

Often 32-64 kB is the maximum amount of volatile memory available on an AWS, it makes the instrument non capable to keep too much data on a Random-access memory (RAM) at all. Forcing to the manufacturers to design the instruments with fast and reliable mass storages, ready to transfer the data from the volatile memory to the persistent storage.

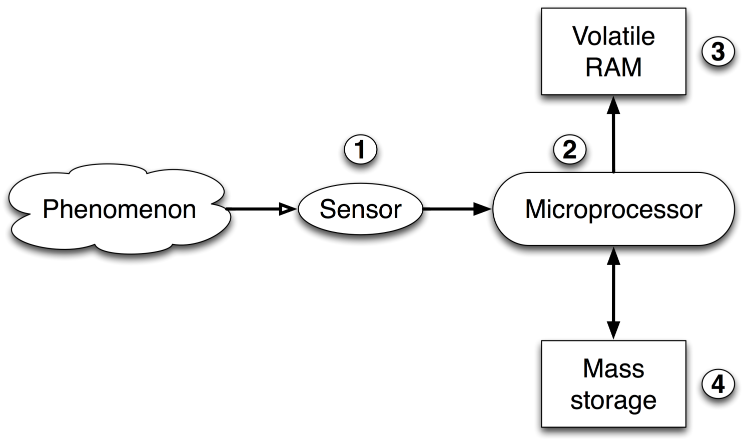


Figure 3.3: Abstracted electronic schema of an AWS reading data from one sensor.

The figure **??** shows the workflow data of an abstract sensor. In the first step the sensor generates the data from the phenomenon, based on the observation of some physical principle; the data acquired is processed by the microprocessor in the the second step, placing the data on the volatile memory. When the data is placed on RAM the in / out (IO) operations start, transferring the data from the volatile memory to the mass storage (persistent memory). According to the

[2]Nowadays some manufacturers are introducing progressively new microprocessors using 32-bit wide.

Guide to Meteorological Instruments and Methods of Observation [46] published by the WMO, it is highly recommended to equip the AWS with a battery backup dedicated to the volatile memory to avoid data loss due to some power fails. This non common feature in generic computers can be an advantage to have in mind when a protocol is implemented, because it enables the possibility to have some methodology in the protocol to recover the session after one power failure.

Mass storage

Typically, an AWS, will have mass storage device to save the data collected from the sensors. The storage of data in the AWS has been changing in the last years due to the continuously decreasing price of flash memories. It is common to find very different architectures in terms of data storage in the AWS.

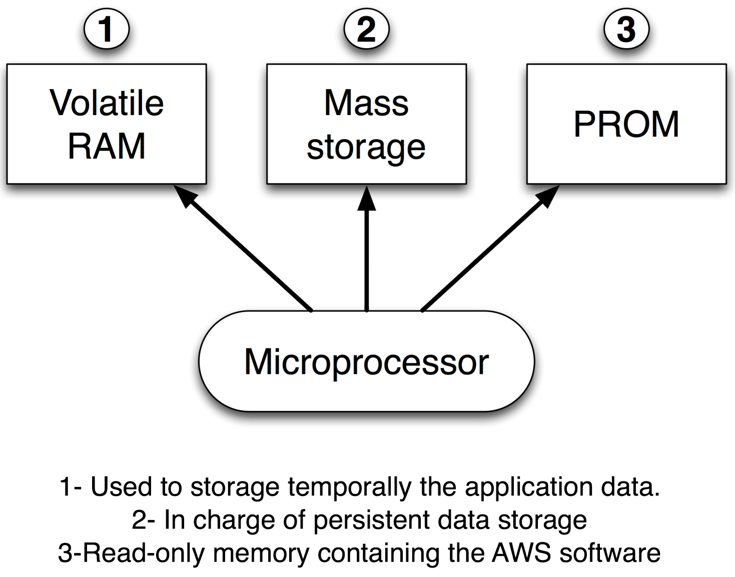

1
2
3
Volatile RAM
Mass storage
PROM
Microprocessor
1- Used to storage temporally the application data.
2- In charge of persistent data storage
3-Read-only memory containing the AWS software

Figure 3.4: Types of storages available in an AWS.

The number of sensors and the frequency in which the information is transferred to the data centers, determines the size of available memory in an AWS. Based on the market, the mainstream option in terms of memory size for mass storage is around 1 Megabyte (MB), that space is more than enough to save thousands of samples in case that the AWS has not send the data to the collecting point.

Sensors

The sensors are the digital interfaces that make an AWS different from other embedded devices. As explained in section 1.1, a sensor is a digital interface using some physical principle to measure a particular phenomenon. Their principles, implementation and complexity are out of the scope of this thesis. Even so, we need to consider the sampling frequency of them because they are involved in the frequency in which the data is produced.

The sampling frequency of the sensor depends on the data required to understand the phenomenon. A big range of sampling frequencies are used to measure different phenomena. Nevertheless, the author is not assuming this frequencies as a need for the protocol.

A correct behavior of the sensors requires a high-accurate calibration of them. The manufacturers have been developing several methodologies and mechanisms to calibrate the instruments and verify their correct behavior. These calibrations are not considered as part of the problem statement of this thesis because they are unrelated to the methods of the data transmission.

Digital interfaces

As mentioned in the section 1.2, an AWS is equipped with at least one peripheral device to provide data interaction. These interfaces offer the possibility to configure the AWS and transfer data from it. The type of device is a serial communication physical interface, and depending on the type and vendor of the instrument, it will be one the following[3]:

- RS-232
- RS-422
- RS-485
- USB

These four types are well-known in the industry. They are available in almost all the modern computers, however the relation of them with this thesis is focus mainly in the bandwidth that they offer. The table 3.1 shows a comparison between these physical digital interfaces and their bandwidth.

Standard	**Bandwidth**	**Bytes/s**	**kB**
TIA/EIA-232-F[2]	116 kilobits (kbit)/s	14848	14.5 kB
TIA/EIA-422-F[3]	200 kbit/s	25600	25 kB
TIA/EIA-485-F[4]	35 Megabits (Mbits)/s	4587520	4.375 MB
USB[12][4]	1.5 Mbits/s	196680	192 kB

Table 3.1: Comparison between standards and bandwidth offered.

Even so, as the table 3.1 shows, the minimum bandwidth provided by theses interfaces (RS-232) should be enough. As described in the sensors section, the

[3]Other types of interfaces can be found in the instruments. However, the industry stablished —with non-written agreement— the use of the mentioned interfaces as mainstream.

total amount of data generated by the sensors of one AWS rarely exceeds 1kB; fitting perfectly this in the bandwidth offered by the RS-232.

Due to the constant renovation in digital interfaces that the industry does, we do not consider other old interfaces in the analysis, assuming that the protocol will work with instruments manufactured in the last 10 years[5].

Although the interfaces are not conditioning our protocol implementation, it is necessary to highlight that most of the vendors offer the possibility to re-wire the AWS to make it work with different physical interfaces.

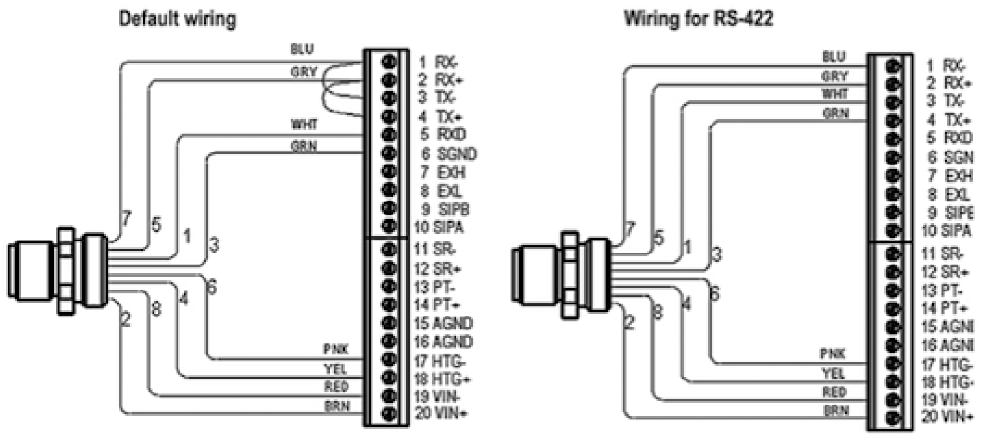


Figure 3.5: Wiring schema showing how to re-wire the AWS to use RS-422.

Datalogger

The datalogger is one the most critical parts of an AWS. It is in charge of the data logging produced by the sensors and deliver by the operating system. Its main task is to keep track of the data collected by the AWS. This component plays an important role in the implementation of the protocol, because of the data of the protocol must be originated in this part.

Depending on the architecture of the AWS, the datalogger can be an external embedded system with serial communication capabilities, able to send data through a network and with multi-station capability[6]. Small AWSes can have datalogging capabilities, keeping the data in a persistent memory for a short period. To have the datalogger implemented internally implies increasing the complexity of the AWS, converting it in a more complex embedded system with features as data delivery through a network, long-term data storage, etc.

[5]Those should be equipped with the interfaces mentioned in the Table 3.1.

[6]Some dataloggers are able to track and to operate several AWS at the same time.

Often, the architecture chosen for AWSes is an external device connected through the physical interface. These devices are equipped with some kind of connectivity such as Global System for Mobile Communications (GSM), General Packet Radio Service (GPRS) or Universal Mobile Telecommunications System (UMTS) modems, using them to deliver the data to the collection point.

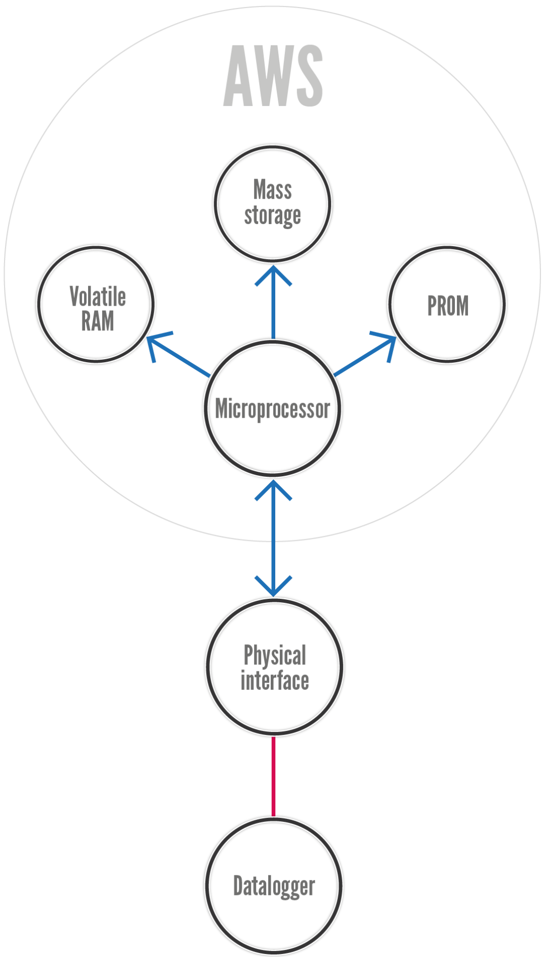


Figure 3.6: Location of the datalogger in an AWS.

3.1.3 Software

As it is common in the embedded systems, an AWS has a tiny internal software. The programming languages used to develop this software have no relevance in this topic. We assume that the internal operating system of the AWS will offer us the data collected from the sensors, moreover of some set of options to configure and calibrate the AWS.

We need to differentiate between the software embedded in the AWS and the software at the end of the peripheral device.

AWS's Operating System

The operating system installed in an AWS resides in a Programmable Read-Only Memory (PROM). Its architecture is based in a real-time clock implemented on the mother board of the AWS. The OS provides a limited set of options to interact with the AWS, most of these options are focused in data acquisition, calibration and hardware configuration. This software is in charge of the formatted data of the AWS, in other words, it gets the data from the sensors, applies the necessary formulas to extract a meaningful result and formats the data in one of the following serial communication protocols[7]:

- RAW ASCII
- SDI-12
- NMEA-0183

After the data is formatted, it is transmitted through the peripheral device to the the datalogger.

External software used for datalogging / data distribution

As explained in 3.1.2, an AWS needs a datalogger device to track the data collected from the sensors. Irrespective of the type of datalogger, at the end of it, we will find some computer in charge of the data manipulation and storage. The software installed on the computers can be really differently implemented and designed depending of the vendor, but its main task is to understand the data format chosen by the vendor to transmit information and take use of it.

The market offer concerning software for AWSes is too big, even some companies not related with the manufacturing of the instruments, are releasing software for datalogging purposes. It is common that the AWS is provided from the factory with its own set of software, nevertheless due to the serial communications protocols used by the AWS, is simple to implement a software that interprets and takes advantage of the data format chosen by the vendor to implement new capabilities.

[7]We need to distinguish between the data format used to communicate with the interface (ASCII, NMEA-0183, etc) and the format in which the data is formatted, this is explained the section 4.2.

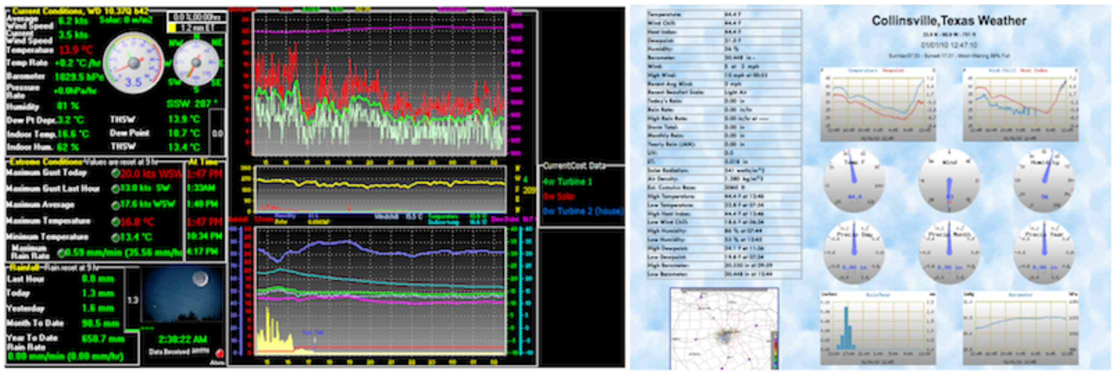


Figure 3.7: Screenshots of some popular desktop applications for AWS.

3.1.4 Networking

As mentioned in the datalogger subsection, the connectivity capabilities in an AWS resides on it. The industry offers multiple options to provide connectivity in an AWS, nevertheless, most of these options are limited for bandwidth, energy and geographical limitations. It is possible to find AWSes directly connected to a computer via USB, providing this the connectivity, or we can find an isolated AWS in the middle of a mountain connected through a radio-link to the closest place.

The common technologies to provide connectivity to an AWS are:

- GSM
- GPRS
- UMTS

In places with better geographical location and energy availability, it is possible to find the following technologies offering connectivity:

- Ethernet
- USB
- 802.11b/g

Whatever the connectivity on the AWS is, the common pattern is that this connectivity is reliable but offers a rather low bandwidth.

3.2 Meteorological data networks

The previous section gave a general overview of AWSes, the relation between them and this thesis, is how they behave in terms of networking communication, which kind of topologies are used and in which points this communication can be improved.

To understand the workflow of the data in terms of weather data collection, we should see an AWS as an individual node without interaction with other nodes, except the collection point.

The collection point is the place in which different data from different AWS is received. It is not mandatory that this collection point is the end of the weather data workflow, for instance it is possible to find an intermediate collection point that has been stablished for geographical reasons to improve the connectivity[8]. Even so, we consider the collection point, the place in which the data has been received and it is ready to be processed.

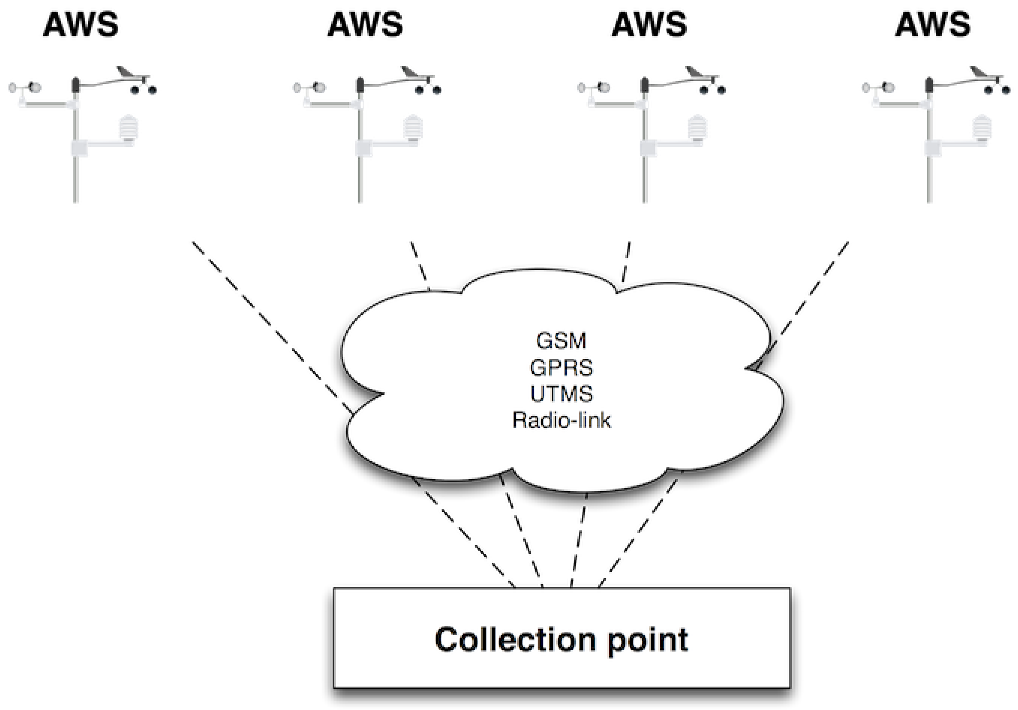


Figure 3.8: AWSes connectivity schema.

When the data arrives at the collection point, different mechanisms get activated to process it. As described in section 1.2, rarely, the data received comes from the same brand of instruments, meaning this that the data will be received in different formats and different time frequencies; this fact forces to implement these mechanisms to homogenize the data and make it understandable on the collection point. The collection point is the hop in which to have a standard protocol to communicate with the AWS will have a bigger benefit, because it is in this hop in

[8]Some AWS are located at inaccessible places, sometimes this implies to establish a collection point close to them to avoid issues such as lack of connectivity (GSM, GPRS, UMTS).

which the most effort is made, it in terms of data parsing, power calculation and data homogenization.

3.2.1 Common architectures

The definition of star topology fits in the methodology used to collect data from different AWSes. The nodes have a strong dependency with the collection point, without it, an AWS will have a high limited time to save data before it is fetched manually. Furthermore, the meteorological networks are not following the pure definition of star topology because different nodes are transmitting data with different connectivity technologies. Nevertheless, seems the nodes are not interacting between them, the network is not affected by bandwidth limitations. This topology is chosen by weather organizations based in the geographical limitations. However, the possibility to interconnect AWSes between them has not been study deeply. The assumption for this is that the utility of the data is based on the availability of it, for this reason the data delivered with big delays is not considered at all in the weather data collection workflow. Interconnect the nodes of the meteorological networks it not feasible with the current technology at all for different factors such as bandwidth, geographical locations or absence of a common protocol.

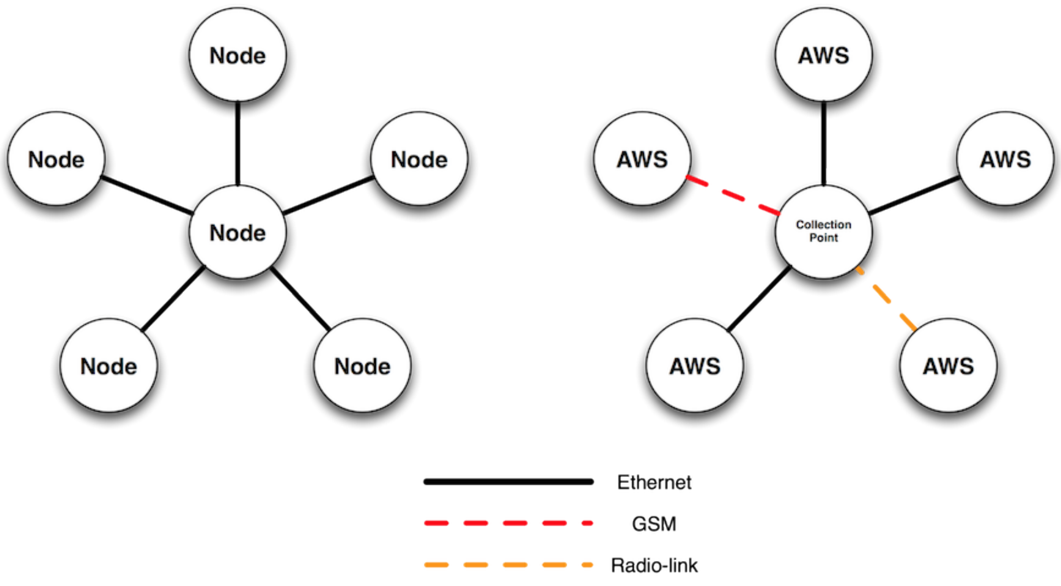


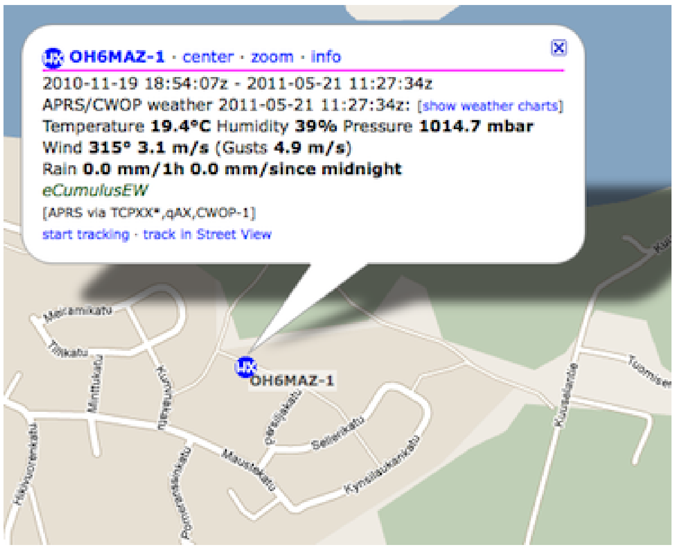
Figure 3.9: Comparison between pure star-topology against star-topology and the connectivity technologies used in AWSes.

Not only star-topology is used in the meteorological networks, the combination of different instruments can end in different topologies depending of the datalogger configuration. For instance, it is possible to have some local network of sensors connected to a datalogger that is part of a star topology, commonly, this topology will be a combination between bus-topology and star-topology. These combinations will not affect a common protocol in anyway, due to its implementation should happen on the datalogger's side, not mattering the combination of topologies behind it.

APRS

APRS is using unnumbered Link Access Protocol for Amateur Packet Radio (AX.25) frames[43]. AX.25 is a data link layer protocol without too many capabilities in terms of bandwidth's offer, error correction and data integrity. Though it is used in some weather stations to spread the data, **it is not a good choice because it is not warranting a constant visibility and connection of the node.**

The AWS using the APRS technology are spreading the data based radio technologies. It is allowing to any node with a radio equipment to receive the information produced in the weather station. Furthermore this topology does not offer any warranty in data delivery because it does not use the collection point model.

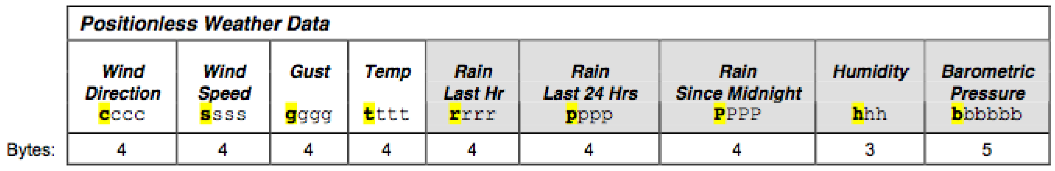


OH6MAZ-1 · center · zoom · info
2010-11-19 18:54:07z - 2011-05-21 11:27:34z
APRS/CWOP weather 2011-05-21 11:27:34z: [show weather charts]
Temperature **19.4°C** Humidity **39%** Pressure **1014.7 mbar**
Wind **315° 3.1 m/s** (Gusts **4.9 m/s**)
Rain **0.0 mm/1h 0.0 mm/since midnight**
eCumulusEW
[APRS via TCPXX*,qAX,CWOP-1]
start tracking · track in Street View

Figure 3.10: Example of an AWS using APRS at Helsinki area[9].

APRS has gained popularity inside the radio amateur community and programs as CWOP due to the simplicity and technical requirements that it implies. The *Weather Station Siting, Performance, and Data Quality Guide*[25] explains how to setup an AWS to get integrated in the CWOP using APRS.

[9]Source: http://aprs.fi



	Positionless Weather Data								
	Wind Direction cccc	Wind Speed ssss	Gust gggg	Temp tttt	Rain Last Hr rrrr	Rain Last 24 Hrs pppp	Rain Since Midnight PPPP	Humidity hhh	Barometric Pressure bbbbbb
Bytes:	4	4	4	4	4	4	4	3	5

Figure 3.11: Weather data message using APRS [24].

Nevertheless, APRS is not used in scientific installations. Although it is not possible to re-implement APRS to adapt it to OpenWeather, it will be possible to use the same data format as it used in OpenWeather under AX.25. Thus, it will offer compatibility between applications using OpenWeather. To provide this capability, will involve modifying the way in which APRS is used, one way to do it can be to send the same data beacon with different formats: standard APRS messages for weather reports and after it a data message based in OpenWeather format.

Even with these incompatibilities the data provided by the APRS data message can be transformed to OpenWeather's data format in a middle point having to modify the APRS protocol.

The author assumes that the AWSes will behave as nodes with connectivity to a common point, being able to interact between them, through the collection point or point to point.

3.2.2 Data distribution

Data distribution is the ultimate's reason for weather data collection. We can identify at least fours levels of different data in the process for weather data collection.

- **RAW data**, produced in the sensors' instruments
- **Network data**, used in the transmission from the instruments to the collection point
- **Operational data**, result of the scientific's practices
- **Informational data**, mainly focus in the general public (forecasts, climate reports, etc)

After the data is collected and processed, the conclusions made by the scientists must be spread to inform the society. It is necessary to highlight that only a few conclusions get to the general public, some of them are known as forecast or climate reports. Most of the data processed is not useful for non scientists, because the complexity or amount of information on it. At the end of the work flow we have the data in two categories, the data that will be minimized to make it understandable to a general public, known as **informational data** [10] and the data that must be shared between different international and local governmental organizations, known as **operational data**.

As part of the problem statement, the data distribution is one of the big efforts that these organizations need to do to make the data that they collect understandable. In 2002 the WMO started a standardization process to create a metadata standard to fix part of this problem, however nowadays this standardization process is still on progress without any draft available[45]. A standard protocol to communicate with the AWS will help the development of a common data format between organizations because all of them will be fetching the data with the same methods and mechanisms.

3.3 Summary

We have now introduced the elements and process involved in measuring and collecting weather data and the technologies related with them. Some topics have been explained to provide a general understanding of how an AWS works.

We have highlighted the limitations the AWSes, concerning data storage and CPU calculation; at the same time the maximum bandwidth available for the digital interfaces has been analyzed. The role of the datalogger has been exposed and its implications of it in the implementation of OpenWeather's format.

In addition, the connectivity technologies available in an AWS have been enumerated, analyzing the bandwidth offered and concluding that only the interruption of the connection and not the bandwidth's offer can be an issue.

Finally, the topologies used in the meteorological networks haven analyzed briefly, clarifying that the AWS are behaving as nodes without interaction between them, only sending data to a common point named "collection point" (the node that interacts with all the AWSes). The APRS protocol and its topology have been explained, taking in consideration the possibility to be compatible with the implementation of OpenWeather.

The next chapter describes the technical issues related with the data transmission in the AWSes.

[10] An example of this is the weather forecast shown every day in newspapers, TV, radio, etc.

Chapter 4

State of the art in the weather data transmission

The previous chapters we have introduced a general overview of the basics needed to understand how weather data are collected and how a weather instrument is designed to undertake its function. Even though the purpose of this thesis is to analyze the issues found in the weather data transmission and to provide an alternative to fix these problems. Nowadays, the way in which a weather instrument is transmitting the data can be classified as generic, because the methodologies used in this task have not been optimized thinking in the data implied in the process. This practice limits the possibility to acquire data without the implementation of intermediary hops in which the data is parsed and converted to a useful data format. This results in an unnecessary investment of CPU cycles, delays in the data delivery, incompatibility between difference brand of instruments, and at the end causing the investment of more resources and effort to exchange data between organizations. This chapter analyzes the technical points that are causing this issues in the weather data transmission.

4.1 The evolution of the digital interfaces in a weather instrument

As mentioned in chapter three, the meteorology did not advance until the invention of the telegraph. The value of the weather data resides in the ability to combine it with other sources to get some conclusions to make predictions. Nevertheless, this combination involves having the possibility to transmit this data fast and far enough. The telegraph brought this possibility, and with this new chance scientists had the opportunity to understand concepts as wind flow and storm movement[1] among others. During the 19th and 20th century the industry has been developing new improvements in the instruments manufactured; all of these improvements come supported for the new methods found by the physics to measure the phenomena, and the conversion of them to digital instruments.

In 1969 the RS-232-C standard was published; this interface has been the mainstream technology used in the weather instruments for more than thirty years; only in the last decade some updates have been introduced in the industry, migrating to new standards as ANSI/EIA/TIA-232-F[2], ANSI/EIA/TIA-422-F[3], ANSI/EIA/TIA-485-F[4] or USB.

As far as we can judge this slow transition in as of the digital interfaces used in a weather instrument come supported for the fact of the wide use of RS-232-C in different fields of the industry, at the same time these interfaces fit perfectly in the needs of the weather data transmission: enough bandwidth, low cost and they are an international standard. If some updates have been introduced in the industry of the weather's instruments, they come supported by the need to adapt these interfaces to the hardware ports available at the moderns computers, seldom by the requirement of more bandwidth[1].

It is an observable fact that the industry performs some updates in the technology to make it compatible with the moderns computers despite that the is not needed in terms of data delivery. Moreover, the new standards are offering more capabilities a part of more bandwidth, for example, technologies as USB, bring the opportunity to plug an AWS to a computer and have it working without previous configurations as bit-rate, parity, etc.[2]

These interfaces provided by the industry are generic as in other technologies, not mattering the type of data transmitted through them; a well-known process of standardization has been performed to develop these interfaces. Though does not exist any standard specifying which type of interface should provide an AWS, the WMO recognizes the universality of the interfaces mentioned, and establishes them as requirement for the AWSes performing official measurements for governmental organizations[46]. Based on this we assume that a protocol implemented in an AWS must work under these technologies; because these interfaces are generic, they have not any requirement for the data transmitted, giving complete freedom to us to implement any protocol over them.

As mentioned in section 3.1.2, the bandwidth offered for the different interfaces available in a weather instrument, are offering even more bandwidth than the amount of data that an AWS's CPU can process. Hence, a weather instrument has not limitations (concerning bandwidth) in the data interfaces that would prevent the possibility to implement a protocol to afford the needs of the data delivery.

Based on this retrospective we assume that the digital interfaces provided by the

[1]In some big AWSes in which have been placed many sensors and complex instruments, exists the possibility to need a bigger bandwidth, even so this is a specific case out of the mainstream setups.

[2]Interfaces based in ANSI/EIA/TIA-232-F, ANSI/EIA/TIA-422-F, ANSI/EIA/TIA-485-F require to adapt the software to certain bit-rates, flow controls and other parameters.

industry are well know and tested standards, providing mechanisms to achieve the goal of the data transmission. However, as it is explained in section 4.2 no weather data transmission protocol has been defined for them. We identified this as **the first deficiency in the weather data transmission because of the potential offered by these digital interfaces is not used in the weather instruments**.

4.2 The absence of a protocol

The goal of the Internet Engineering Task Force (IETF)[26] is to make the Internet work better. One of its multiple task implies to take care about the standardization process of the new Internet standards. A protocol is considered as standard when the IETF publishes a memorandum[3], specifying all the aspects of the protocol and assigning a number in the STD series of it[7].

A research performed by the author in the RFCs available at IETF's website [26][4], looking for the following terms: "weather", "meteorology", "weather station", "atmosphere", "weather data", gave as result the following number of mentions. Only **9** RFCs do direct or indirect mention to the weather data.

The first RFC mentioning a protocol related with the weather data is the RFC 765 [38] File Transfer Protocol (FTP):

3.4.2. BLOCK MODE The file is transmitted as a series of data blocks preceded by one or more header bytes. The header bytes contain a count field, and descriptor code. The count field indicates the total length of the data block in bytes, thus marking the beginning of the next data block (there are no filler bits). The descriptor code defines: last block in the file (EOF) last block in the record (EOR), restart marker (see the Section on Error Recovery and Restart) or suspect data (i.e., the data being transferred is suspected of errors and is not reliable). This last code is NOT intended for error control within FTP. It is motivated by the desire of sites exchanging certain types of data (e.g., seismic or ***weather data****) to send and receive all the data despite local errors (such as "magnetic tape read errors"), but to indicate in the transmission that certain portions are suspect). Record structures are allowed in this mode, and any representation type may be used.*

Nevertheless, this reference of weather data is just an example (as the other references) that disappeared in later updates of the File Transfer Protocol (FTP).

The industry has focused its effort in improving the measure methodologies, the robustness of the instruments or other features as power consumption or life-time. Thus, the methodologies utilized to transmit weather data have been developed independently by the vendors, choosing their own data formats and techniques.

[3]This memorandums are named as Request for Comments (RFC) for historical reasons.

[4]The searched has been performed over all the content of the RFC published: ftp://ftp.rfc-editor.org/in-notes/tar/RFC-all.tar.gz . Retrieved: 28-03-2011.

Nevertheless, the WMO initialized different programs as Global Observing System (GOS), Global Telecommunication System and WMO Information System (GTS) , Global Data-processing and Forecasting System (GDPFS) [44] among others, in which the weather data exchange is a key-component of the systems to archive the goals of these programs. In addition, as mentioned in the section 3.2.2 the WMO started a process of standardization 9 years ago.

Even assuming that the industry focused its attention on prioritizing measurements methods and product quality, the technologies related to the weather data transmission are outdated. The proof of this is that only a few governmental organizations have access to real-time information [5] collected from the AWSes[6], at the same time programs as CWOP still depend of technologies such as FTP or APRS, that they do not contemplate scenarios in which scalability, data on demand or real-time data is needed. Finally, as a real example, the SMEAR project[40] is experimenting the issues of not having a standard protocol for the AWS, producing as result the implementation of intermediary points to parse and normalize the data, incompatibility between different sources of data from the same phenomenon collected with different instruments and scalability of the system among others.

Based in these facts, we can say that during the last 40 years the industry unattended the communication's side of the AWS, adapting the instruments to be capable to use protocols as FTP to transmit the data from the AWS to the collection point; focalizing the effort transmiting the data not mattering at all the technologies used or if they are or not optimized for that purpose. This practice gave as result multiple data formats implemented by the vendors without any common agreement, creating a huge incompatibility between the instruments and several bottlenecks in the data transmissions.

The following subsections expose some data format used by the vendors to archive the data transmission and analyze why these data formats are causing bottlenecks.

Data formats used by the vendors

As mentioned in previous chapters, the format in which the data is produced by AWS is formatted is up to the vendors. Nowadays the only standards used or involved in this process is ASCII as character-encoding scheme or NMEA-0183. Depending on the digital interface different control characters can be used, for instance is a common practice to generate one line of data follow by the carriage return (CR) or carriage return followed by line feed (CR+LF)[7].

Depending the AWS's brand the data's format is completely different from other brands and vendors. In most of the cases the data format is implemented based

[5]All of these instruments are generating by default real-time data.

[6]Note that these organizations can have this capability due to they invest a big effort in to develop custom systems for their weather instrument's setup.

[7]CR hexadecimal value: 0x0D. LF hexadecimal value: 0x0A. CR+LF: hexadecimal value 0x0D 0x0A.

```
>"BARDATA"<LF>
«LF><CR>"OK"<LF><CR>
<"BAR 29775"<LF><CR>
<"ELEVATION 27"<LF><CR>
<"DEW POINT 56"<LF><CR>
<"VIRTUAL TEMP 63"<LF><CR>
<"C 29"<LF><CR> <"R 1001"<LF><CR>
<"BARCAL 0"<LF><CR> <"GAIN 1533"<LF><CR>
<"OFFSET 18110"<LF><CR>
```

Table 4.1: Example of data format used in a specific AWS to communicate the barometric pressure.

in the vendor's wishes. These wishes can be supported by technical reasons or not. Some vendors used acronyms to refer the data values returned by the sensors, others use the whole word to refer the phenomenon; not mattering the technique used in the data format, is a fact that they do not exist any compatibility of formats between vendors.

```
0r2,Ta=10.6C,Tp=10.8C,Ua=74.6P,Pa=1006.0HKHK
```

Table 4.2: Another example of data format used in a specific AWS to communicate different data as temperature or barometric pressure.

A part of these big differences between the formats used in the digital interfaces, is needed to highlight that also the field's value used in CSV or TSV files producted by the AWS are unique and incompatible between vendors. Thus, two levels of incompatibility exist, first the original data is delivered in a custom formatted untill the software's side. In the software's side this data is converted to a CSV or TSV format with the custom fields chosen by the vendors; this causes that even having the final data in a standard format as CSV or TSV, the order of the fields and their denomination will be different, forcing to the scientist to add an extra layer to the workflow to normalize this data and make it ready to be combined.

Data formats used by governmental organizations

Despite the fact that vendors used privative and non standard formats for the data, the WMO has defined some specific data representation for certain users. An example of this is the Meteorological Service For International Air Navigation (METAR) format. Approved by the International Civil Aviation Organization (ICAO), this format is the only one considered as official to communicate weather forecasting to the aviation and at the same it is widely use for other purposes as general weather forecasting.

Phenomenon	**METAR's acronym**
cumulonimbus clouds	CB
thunderstorm	TS
moderate or severe turbulence	MOD TURB, SEV TURB
wind shear	WS
hail	GR

Table 4.3: Some acronyms used in METAR format [36].

However, this format has not relationship with the formats used by the vendors. Only a few AWSes have the ability to product the METAR format by default. The AWS doing this are only focus in product data useful for the aviation, wasting the opportunity to provide the data in other formats for different use.

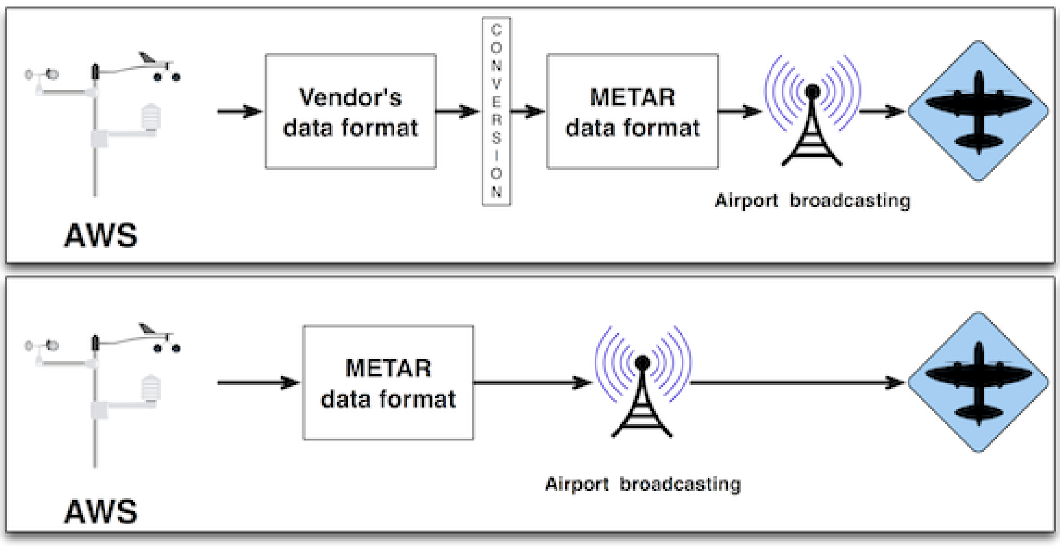


Figure 4.1: Weather data workflow, normal AWS VS METAR's AWS.

METAR format is just an example of the multiple data formats invented for a specific purpose. The point to highlight is that often the weather data can be represented in a complete different format compare with the original format used for it. Nevertheless, the optimization of the data format until the point in which it is transformed marks a big difference in terms of data manipulation.

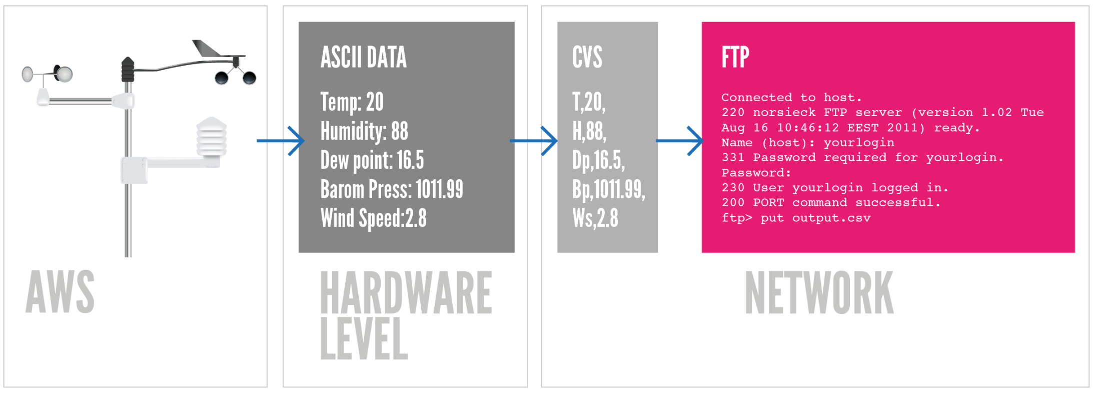
With the current technology the weather data arrives in different formats and with difference times frequencies, forcing to implement customized and particular mechanisms to transform this data to the format required. The complexity of this task resides in the requirement de facto requested by the AWSes: they need intermediary points to convert the data because by default the data provided is useless for the required result.

In conclusion, it does not matter if the vendors provide a well known documented data format of their instruments. Because the observation of the weather is performed with different instruments, the data must be normalized to make it understandable. Thus, at the end of the data workflow (when we take data from different sources and instruments), an intermediary layer to translate the vendor's data format to a common format is required.

Mainstream architecture used for the weather data transmission

To understand where are located the bottlenecks in the weather data transmissions is needed to understand the current architecture used by the vendors to archive this goal. As explained in section 3.1.2, an AWS is an embedded system collecting information produced by the sensors attached to it. As embedded system, it has small capabilities to perform big CPU calculations, massive data storage or data delivery, however moderns systems are pretty balanced in terms of hardware and software to archive this goal. Although the AWS have been optimized to collect and delivery the data, the protocols used for it are generic an non-specific. As explained in previous chapters the quality of the weather predictions reside in the ability to collect and process the atmosphere data with efficiency, reliability and fast delivery.

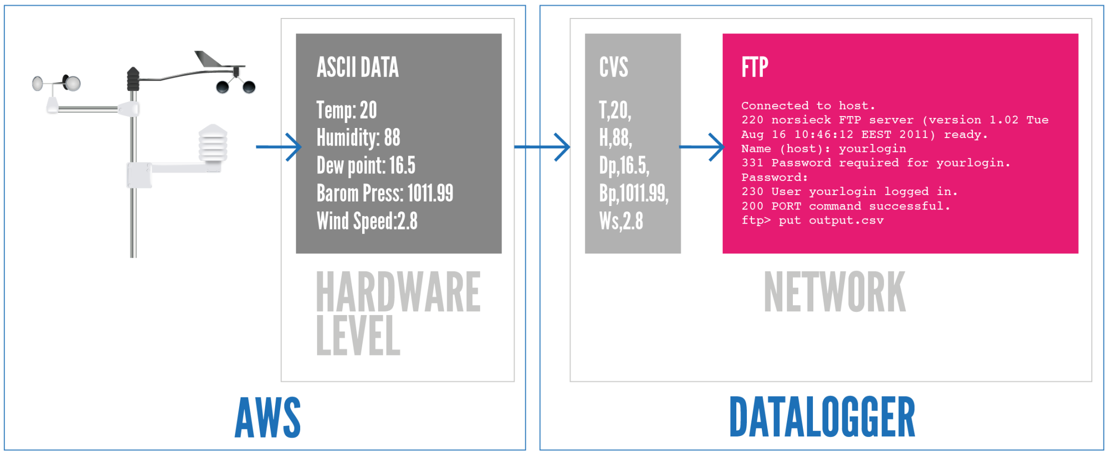
Despite of this, the methodologies for network communications are not optimized for this purpose. The following figure shows how the data is delivery.



```
Connected to host.
220 norsieck FTP server (version 1.02 Tue
Aug 16 10:46:12 EEST 2011) ready.
Name (host): yourlogin
331 Password required for yourlogin.
Password:
230 User yourlogin logged in.
200 PORT command successful.
ftp> put output.csv
```

Figure 4.2: Example of an AWS transmitting weather data.

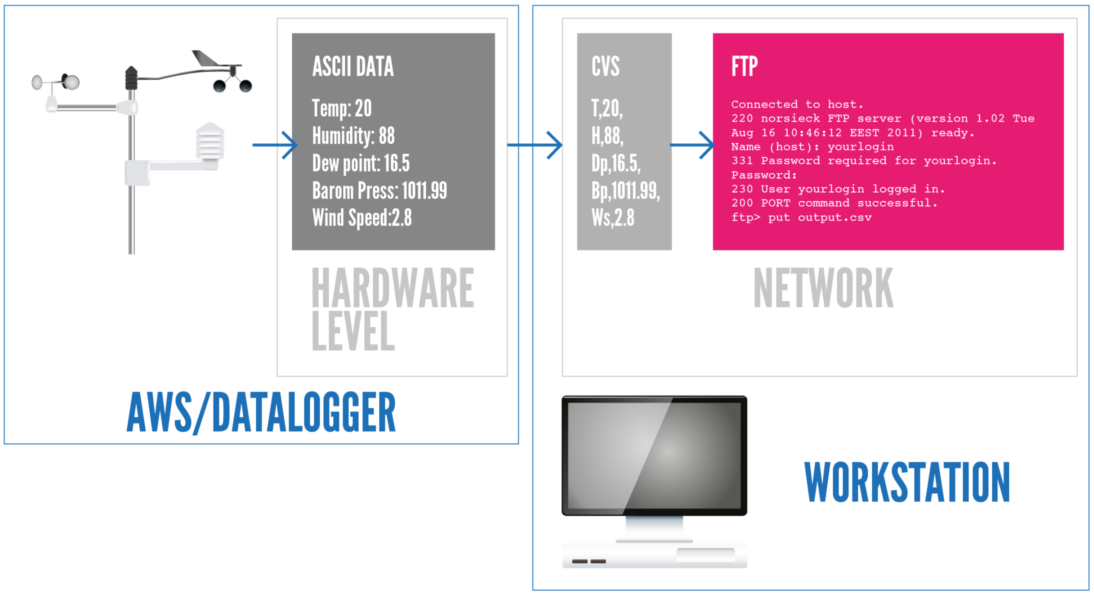
In the figure 4.2 we can appreciate an example of the methodology used to transmit the weather data. In the hardware's level the data is delivered through a digital interface as explained in section 3.1.2, using some custom vendor's data format, commonly based in abbreviations as "Tmp (Temperature)", "Bp (Barometric Pressure)" "Ws (Wind Speed)", among others. These abbreviations are understood by the software. Depending of the AWS's setup this process can happen all together between the AWS and the datalogger:



```
Connected to host.
220 norsieck FTP server (version 1.02 Tue
Aug 16 10:46:12 EEST 2011) ready.
Name (host): yourlogin
331 Password required for yourlogin.
Password:
230 User yourlogin logged in.
200 PORT command successful.
ftp> put output.csv
```

Figure 4.3: Example of a AWS and a datalogger transmitting weather data.

If the AWS / datalogger has not network capabilities, a third entity can enter in the workflow. This entity is commonly a modern computer with the peripheral devices needed to interact with the AWS. The computer takes the role of the weather data transmission, due to the possibilities that it offers, one computer can manage several AWSes at the same time. Nevertheless, it does not introduce new protocols to send the data, it stills using protocols as FTP or in some setups just shared folders using Server Message Block (SMB):



```
Connected to host.
220 norsieck FTP server (version 1.02 Tue
Aug 16 10:46:12 EEST 2011) ready.
Name (host): yourlogin
331 Password required for yourlogin.
Password:
230 User yourlogin logged in.
200 PORT command successful.
ftp> put output.csv
```

Figure 4.4: Workstation taking the role of the weather data transmission.

FTP, the mainstream protocol in the weather data transmission

Disregarding the setup used to send the data to the collection point, the protocol used will be generic and in most of the cases based in FTP. Although FTP has the capability to operate under stream mode [39], the author could not find any vendor offering the capability to deliver the data through stream FTP connections. Even being this possible, it will involve to use the image mode (commonly known as binary mode, thus, involving byte ordering choices) to transmit the data, however this choice will subject the data transmission to problems with the endianness[8].

This setup can fill the requirements to delivery weather data collected over different time frequencies, however, it can not offer real-time capabilities, because the FTP is not designed for this purpose. The author identifies **the use of FTP as a deficiency in the weather data transmission** [9], the reasons for this are based in the fact that the protocol is designed to provide network capabilities to delivery data streams based in files. Notwithstanding, the AWSes are producing data streams based in real-time data; **the use of FTP involves an intermediary step to convert these data streams to files, to continue after this sending theses these files to the collection point**. Even though to track this data in files is needed for storage and backup reasons, the data streams generated in real-time by the AWS are not used at all to send them directly to the collection point. In addition, the use of this methodology is forcing extra IO operations required by the FTP, that are not required in other protocols in which the data transfer does not involve the use of files.

Thus, it is not available any protocol taking advantage of all the capabilities offered by the AWSes and its sensors, instead generic protocols as FTP or SMB have been chosen to transmit data. These protocols are widely and accepted as the mainstream solutions for data transmission available on the weather instruments.

4.3 The missing standard

One of the important factors of an implemented protocol, is to know how is going to be represented the data transmitted at the end of the transmission. This helps to design the best representation required by the data; for instance a protocol

[8] *"Endianness describes how multi-byte data is represented by a computer system and is dictated by the CPU architecture of the system. Unfortunately not all computer systems are designed with the same Endian- architecture. The difference in Endian-architecture is an issue when software or data is shared between computer systems. An analysis of the computer system and its interfaces will determine the requirements of the Endian implementation of the software."[15].*

[9] All the AWS checked by the author are offering the data delivery based in ASCII files using the FTP ASCII mode and sending the data using the FTP block mode. Though is possible to find some AWS using different methodologies as HTTP get methods or email delivery, the FTP choice is mainstream overall the industry.

implementing real-time capabilities should be focus in fast data delivery and data integrity, among others.

In addition, to know the final representation of the data helps to implement a protocol optimized for the data that is transporting, this gives as result a better software for the protocol, besides it provides the capability to implement different protocols giving the same data result[10].

Nevertheless, the weather data has certain particularities; the WMO defines a set of methods to perform different measurements, notwithstanding theses methods are changing based in the advance of the physics, and these changes are causing an instability concerning what is the best way to measure a phenomenon, thus the data representation can get affected easily. Furthermore, the correlation between phenomena generates certain scenarios in which the data results can change completely if a new method is found to measure the phenomenon. This fact determines to which point we can have or not standards for these particular data. The WMO defines which system of units must be used to represent the data for scientific purposes, in addition several guidelines are provided by the WMO to perform the measurements under standard procedures. However, these guidelines are not enough to specify the final format of the data.

The WMO started a process of standardization in **2002**, the goal is to create a data format to fit the requirements of the GOS, in other words to provide a common basement to represent the data of the weather's observations. This is an arduous task, not only for the amount of data that is needed to manage, also for the big a mount of different phenomena in the atmosphere that are producing different data and their particularities. It is expected that in some point, the WMO will publish a standard for weather's metadata representation, nevertheless, after 9 years this process still under development.

The absence of a standard for weather data representation is one of the key-issues of the current situation. Without knowing how must be formatted the data at the end of the collection workflow, is understandable that vendors ended implementing their own formats without compatibility.

This is an open issue that unfortunately can not be treated in this thesis. The author recognizes that the implementation of a protocol to transmit the weather data without to know the final format of the data is a risky but an interesting feature. In chapter eight, an exposition of the solution chose (a software library to normalize the data) is explained.

We identify the absence of a common format for data representation as one of the major technical deficiencies in the weather data transmission. In addition, the absence of a common data format in the collection point as well, forces to convert the weather data multiple times to the final format. OpenWeather considers this issue and provides some mechanisms to implement smoothly and

[10] A good example of this are the peer to peer networks, in which the protocol's designers know that at end of the process the data must be a file.

mostly transparent the conversion from OpenWeather's format to a future data standard.

4.4 Data transmission and Automatic Weather Stations

As embedded systems the AWS have more limitations that moderns computers, not having capabilities to perform complex CPU operations or to manipulate a considerable amount of data. Most of the modern AWSes offer the possibility to interact with them in a small scale. Commonly, this interact is focused in three tasks:

- AWS configuration
- Sensor's calibration
- Data retrieval

Even so in most of the cases the AWSes behave as "broadcasters" of weather data. The tasks of configuration and sensor's calibration are performed only a few times in the instrument, happening this at the beginning of the AWS's installation and in some periodical calibrations during the life-time of the instrument; both operations are performed in most of the cases through command's line parameters or some GUI developed for this purposed. As it was explained in section 3.2.2, the data transmission with an AWS is performed through digital interfaces based in serial communications standards, it means that at the end all the data transmitted and received in an AWS goes through some data format implemented by the vendor that provides a set of custom instructions.

>"BAUD 9600"<LF>
«LF><CR>"OK"<LF><CR>

Table 4.4: Example of command configuring the baud rate of the digital interface in an AWS.

Even if this practice is something understandable[11], an exception should be made in the data retrieval operation.

Most of the AWSes offer the possibility to retrieve particular data if a specific command is sent to them. Again the method to obtain this data is up to the vendor, not being compatible these instructions between vendors, and even sometimes even not between the products of the same manufacturer.

[11]The author recognize that to have a proprietary set of instructions can be a method to keep some industrial's secret of the instruments, however this practice difficulties the implementation of standard methodologies to interact with multiple the instrument.

The mechanisms to retrieve data from the AWSes **are critical** in order to implement a protocol with real-time capabilities. We need to differentiate two use cases on an AWS. The first use case involves the data broadcasting that the AWS is performing by default if it is configured as "automatic mode"[12]. The AWS just send the data through the digital interface in the time frequency configured, for this case is not required interaction with the AWS; to read the data from the digital interface is enough to use it in the protocol. Nevertheless the second use case involves the retrieval of particular data. One example of this is a user interested in to know the average of temperature recorded by the AWS in the last week. This data is not sent by default because it is not part of the information collected in real-time for the AWS, to get the data the user must send a command asking for it to the AWS:

Command: aR2<cr><lf>
Response: 0R2,Ta=23.6C,Ua=14.2P,Pa=1026.6H<cr><lf>

Table 4.5: Example of command asking for PTH data.

This second use case introduces much more complexity. If a particular data not send by default is needed, the interaction with the AWS is mandatory, however, to interact with it implies to do it using the methodology specify by the vendor. To implement a protocol that takes this use case in consideration involves to implement a command-translator between the AWS and the protocol implementation. **We identify this issue as another technical deficiency in the weather transmission in order to enable the capability to retrieve specific data on demand.**

4.5 Summary

In this chapter we described the state of art in the weather data transmission. We have been analyzing the different interfaces available in an AWS, focusing on their bandwidth, and based in the bit-rate that they offer, concluding that the AWS are not taking advantages of all the capabilities offered by the digital interfaces. This fact is enough reason to claim that the AWSes are capable to manage more amount of data that the current quantities that they do.

Data formats used by the vendors and data format requested for the governmental organizations have been compared; finding that is not any relation between the original format used in the AWSs and the final format in which the weather data is represented, being this one of the reasons that forces the implementation of intermediary points to translate the data to different data formats.

The absence of a protocol dedicated to the weather data transmission has been studied; the use of the FTP has been explained and the limitations that it can

[12]This is the default configuration used in almost all the scenarios.

involve to transmit data in real-time have been analyzed. We conclude that FTP is chose by the industry as non-optimal solution that fix partially the issue of the weather data transmission. In addition, the key issues of FTP has been exposed in order to implement a system that use this protocol to delivery data in real-time.

We analyzed the implications of a missing standard to represent the weather data, concluding that without a consensus of the international community about how the weather data should be represented, is really complex to implement a protocol to fit all the requirements needed.

Despite the absence of a protocol and the use of multiple protocols and data formats, the industry and weather organizations are using these methodologies to acquire weather data in their weather data networks. Although projects such as GOS or GDPFS, are looking for technologies to optimize and standardize the weather data transmission, the current status of weather data acquisition is based on the methodologies that the industry provided without previous agreement. These methodologies have been accepted by the weather organizations as the standards for the weather data transmission, achieving until today their purpose.

At the end of the chapter we exposed how to retrieve particular data from the AWS involves user interaction, adding complexity to the data workflow and requiring an intermediary step to translate the data requests to the native format used in the AWS to retrieve the data. We identify this as an impediment in order to implement a protocol that provides data on demand.

The next chapter explains in which consists OpenWeather, its architecture and how it can fix the issues explained in this chapter.

Chapter 5

Introduction to OpenWeather

The previous chapter summarized the issues found by the author in the protocols used for weather data. It has been analyzed how the weather instruments use protocols as FTP or SMB to transmit data. Nevertheless, these protocols are not designed to be used in a scenario in which the data is generated based on real-time inputs. In addition, the current methodologies provided by the industry, are not efficient enough to interact with the AWS without additional effort in performing data normalization or data delivery. This chapter gives a general overview of OpenWeather, the protocol developed by the author, in order to provide a solution to problems that weather instruments encounter during data transmission.

5.1 Overview and goals

OpenWeather is an application layer protocol based on TCP/Internet Protocol (IP). It assumes a reliable transport layer (TCP), in order to achieve a successful data delivery, based on such mechanisms as error detection, flow control, congestion control, etc.

The protocol is built assuming three principles:

- Every AWS is considered to be a node
- A node accepts incoming sessions from peering hosts and initiates outgoing sessions to peering hosts as well
- An AWS must have the capability to provide and to request services from other nodes.

These principles are supported by assumptions that an AWS is an embedded system with networking capabilities, able to interact via TCP/IP to deliver the data produced by its sensors. The sensors' output are considered to be services offered by the AWS (node) to other nodes.

In addition, the star topology explained in section 3.1.4, disappears to give way to a decentralized topology based on a peer to peer architecture.

OpenWeather provides the capability to dispense a unique collection point. Instead, all nodes can be collection point and at the same time to be part of other collections points. In addition, the protocol offers a service oriented model (SOA), to provide an easy way to interact with the nodes and retrieve or send data to them.

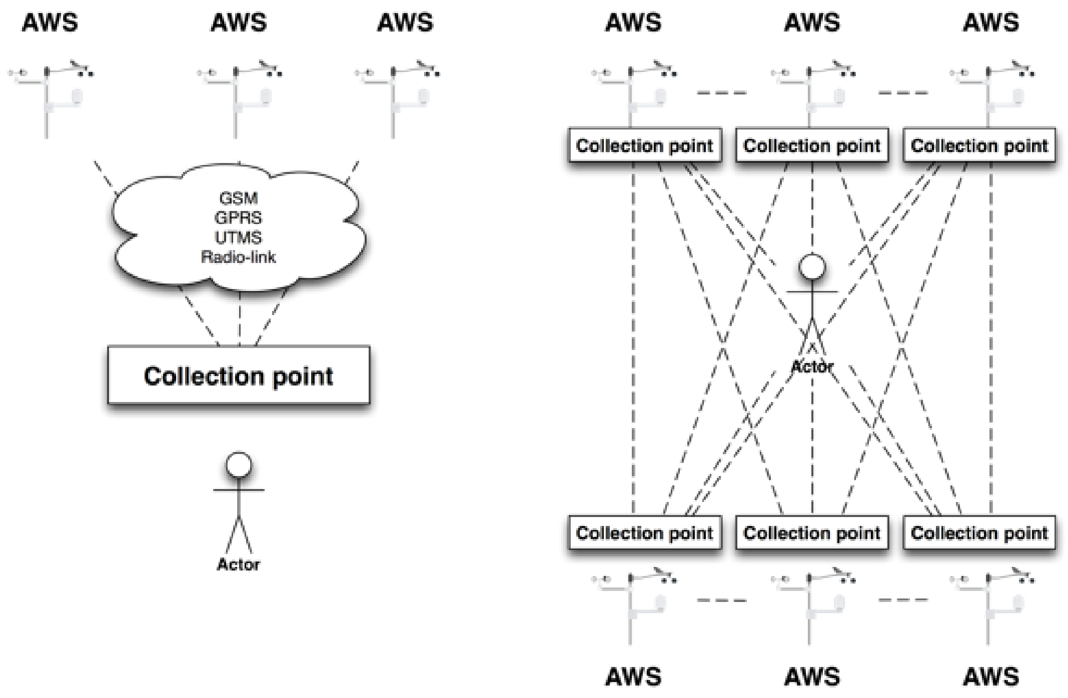
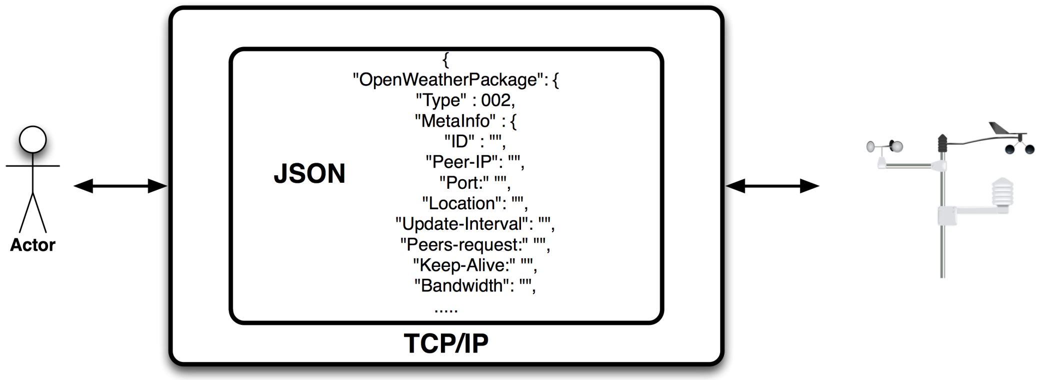


Figure 5.1: Comparison of the currently centralized architecture provided by the industry against OpenWeather architecture.

From the perspective of portability and data delivery, the protocol has been designed to avoid problems with the endianness and data normalization; to achieve this goal, JSON[16] has been chosen as data interchange format between nodes.

JSON allows OpenWeather to use data streams based on parsable objects, facilitating the data manipulation and normalizing the data to one common format. Additionally, JSON is well supported by several libraries[17], bringing the possibility to easily create applications based on OpenWeather format.



```
{
"OpenWeatherPackage": {
"Type" : 002,
"MetaInfo" : {
"ID" : "",
"Peer-IP": "",
"Port:" "",
"Location": "",
"Update-Interval": "",
"Peers-request:" "",
"Keep-Alive:" "",
"Bandwidth": "",
.....
```

Figure 5.2: Example of a OpenWeather's JSON object inside of data message.

5.1.1 Improvements in the current technology

OpenWeather provides a new paradigm for weather data collection. Based on a Peer to peer (P2P) architecture, it allows the users to interact between multiple nodes, retrieving and sending information inside of the network independently of the brand's instruments used. At the same time, it brings the possibility to combine the real-time data streams obtained from the nodes, providing a stack to build applications using multiple data sources without requiring extra resources on the data manipulation.

In addition, the protocol is designed to be extensible, adaptable to new types of data, while maintaining compatibility with future formats. Furthermore, the service oriented model (SOA) of the nodes, allows the users to develop applications that only want to obtain some specific data from a particular service.

Finally, the protocol brings new opportunities to be operated under distributed models and to provide implementational basis for future standards of the weather data categorization. Because the data interchange format is text-based and human-readable, it provides the capability to combine the protocol with database applications without the need to develop extra APIs, facilitating even more possibilities to take advantage of the data.

5.1.2 The role of OpenWeather and data spreading

OpenWeather is designed to fix deficiencies in weather data transmission, while helping with the tasks of spreading data to the end users. Though most of the phenomena require scientific analysis to make the data understandable, some phenomena as atmospheric temperature, pressure or wind speed, are simple enough and known to be spread across them directly to the end users without the need of additional processing. OpenWeather allows to connect to an AWS[1], to retrieve this type of data in real-time and —host to host— based, not needing more than a computer with software supporting the OpenWeather protocol and network connectivity.

In addition, the technologies used in OpenWeather can facilitate the creation of new APIs for web services oriented on weather's forecasts. Some websites offer the possibility for calling APIs to obtain weather data. However, these API calls are completely different between websites, which leads with extra development time of web applications which utilizes different web resources for data extraction. This problem can be easily handled with OpenWeather, creating standard API calls according to the protocol specification. This enables the use of such encapsulated protocols methods as HTTP for creating for an intermediary bridge between the web application and the end nodes.

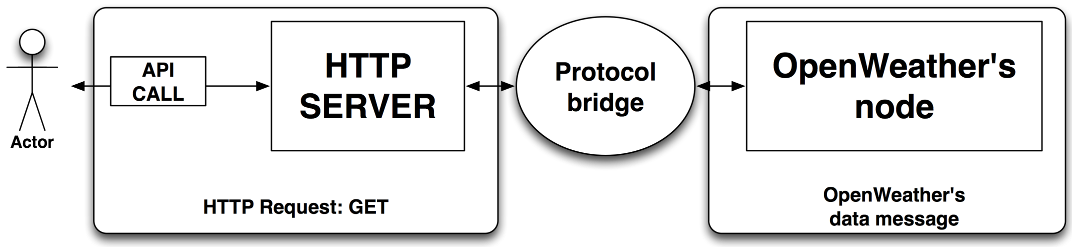


Figure 5.3: Example of an API call through HTTP and OpenWeather.

5.1.3 Contribution to the current methodologies for weather data acquistion

Even if OpenWeather is a proof of concept of an adapted protocol for AWS, it proves how the problems exposed in chapter four can be resolved. The feasibility of migration of scientific installations for production, will be deemed feasible as the principles applied in OpenWeather, just adopting the P2P architecture or the use of a human-readable lightweight format as JSON, it will be enough to observe improvements in data delivery and acquisition. In chapter seven is analyzed the results of use OpenWeather.

As it was mentioned in chapter four, the WMO has several worldwide projects, such as GOS, in which different weather organizations around the world are involved in the process of creation of future basis for weather data processing. As described on WMO's website[44], one of the purposes of the project is: *'The coordinated system of methods and facilities for making meteorological and other environmental observations on a global scale in support of all WMO Programmes"*. OpenWeather,

[1]Through a intermediary layer implemented through software.

as scalable and extensible protocol, can proven useful in certain areas of projects as GOS or SMEAR[40], concerning data availability.

5.1.4 Impact on weather instrument industry

As it was analyzed in chapter four, the industry has not started the process of standardization for their instruments. Despite the issues that this practice causes, OpenWeather aims to be the first solution that tries to fix the absence of such protocol and at the same time provides a basis to be adapted for the future data standard format, providing better archiving mechanisms for a more efficient exchange of weather data.

Furthermore, the P2P architecture brings such a new industry paradigm, allowing to develop new products in which real-time data retrieval will be put to use.

5.2 Basic functionality of OpenWeather

Considering any AWS a node, the implementation of OpenWeather should be done inside of the AWS's software itself. Nevertheless, the author **can not implement a fully functional prototype, because it is not available any open source / libre software version of AWS's OS**. Instead, an intermediary layer has been created for the evaluation setup, to normalize data from vendor format into OpenWeather format. [2].

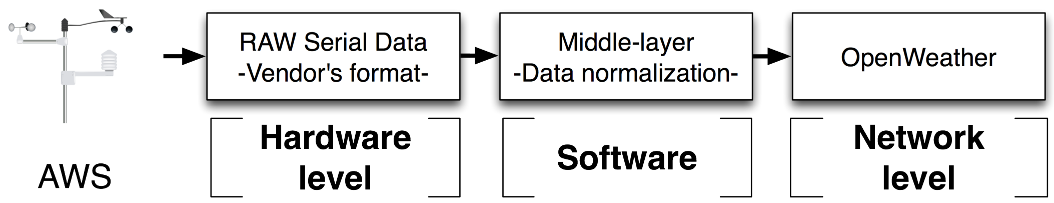


Figure 5.4: Middle-layer for data normalization.

This layer provides the conversion from native vendor format explained in section 4.2, to an operational format in which OpenWeather can work. When the data is pulled through a digital interface, the middle-layer recognizes the vendor format and converts it according with OpenWeather requirements.

This middle-layer is located between the hardware and the network level, giving as a result formatted data ready to be used in the protocol. With the introduction of

[2]The removal of this layer depends on cooperation between vendors in order to implement a protocol inside of the AWS's OS.

this layer, **the steps mentioned in previous chapters[3] disappear**. The data normalization occurs only once at a time, instead of multiple times along the data workflow.

Original sender AWS data:0r2,Ta=10.6C,Tp=10.8C,Ua=74.6P,Pa=1006.0HKHK

OpenWeather's format:

```
"Data" : {
            "PTU" : {
                "Air-Temperature" : "23.6",
                "Relative-Humidity" : "14.2",
                "Air-Pressure":  "1026.6"
            }
```

Table 5.1: Comparison of one vendor format against OpenWeather JSON format.

When data is normalized by this intermediary layer, the AWS is ready to operate inside of OpenWeather network. This intermediary layer will not be needed if the vendors establish a process of standardization.

5.2.1 Peer to Peer Architecture

As mentioned in section 5.1, OpenWeather is designed based on a P2P architecture. The RFC 5694 (Peer-to-Peer (P2P) Architecture: Definition, Taxonomies, Examples, and Applicability)[8], defines a P2P system as the following:

[...] We consider a system to be P2P if the elements that form the system share their resources in order to provide the service the system has been designed to provide. The elements in the system both provide services to other elements and request services from other elements. [...]

OpenWeather is according with the definition established by the RFC 5694 [8]. The protocol is thought to share the resources available in an AWS and at the same time request services from others. In order to function properly the OpenWeather network requires a minimum activity that must be performed by the nodes (as peers's list exchange).

Note that user itself is considered to be a node. **It is not necessary to have an AWS in order to be considered a node**. A node is part of OpenWeather network, interacting with other nodes, sending and retrieving data, while time offering services to them[4].

An OpenWeather node possesses the following properties:

[3]Concerning data parsing.

[4]Thus, a user without an AWS can interact with other nodes offering for example peer list exchange.

- A node has a **unique ID** within OpenWeather's network
- The geographical location of a node **is essential to its connection in order** to OpenWeather's network
- A node of the OpenWeather network can require the use of Network address translation (NAT)[21] [5]

Opposed to other P2P networks, OpenWeather does not use the P2P architecture to archive a better performance transmitting big amounts of data[6]; the justification of use of P2P architecture in OpenWeather is based on the distribution of the nodes and for better interaction with them. The centralized model, fails to utilize its ability to use weather data from different collections points without a pre-normalizing data. In addition, the P2P architecture enables scaling of the network as well, as giving the advantage of not being restricted by the limitations of a central node.

5.2.2 Service Oriented Architecture in nodes

As explained in section 3.1.2, an AWS produces real-time data collected by its sensors. At the same time some AWS are able to store specific data in persistent memory such as averages figures, daily reports, etc. These features provide two data use cases for OpenWeather:

- Data becomes available in real-time
- Data can be retrieved on demand without the need to be real-time specific

OpenWeather handles these use cases providing an extra layer based on SOA. In order to achieve this, OpenWeather provides a mechanism to discover which services being available in a particular node, being possible after the initialization of the session, to interact with these services.

[5]As described in RFC 5128 (State of Peer-to-Peer (P2P) Communication across Network Address Translators (NATs)[41], will be recommendable to implement the TCP/UDP Hole Punching technique in OpenWeather's software, in order to avoid peer connectivity issues.

[6]In fact, as explained in section 3.1.2, the amount of data generated by a node is insignificantly small.

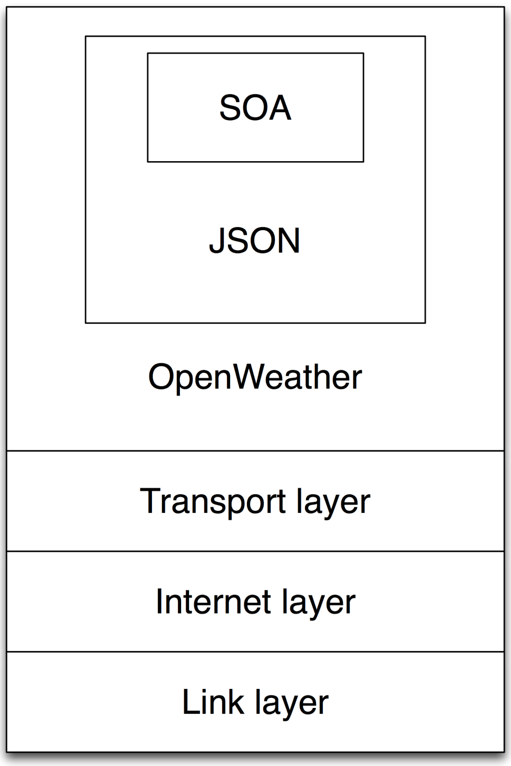


Figure 5.5: OpenWeather stack over TCP/IP.

The fundamental reasons of choice of SOA for OpenWeather, is to facilitate the accessibility of the data. A user can be both interested in receiving only real-time data or in to retrieving a particular chunk of data. To provide this capability, the protocol must be SOA oriented, in order to alleviate data access through these services.

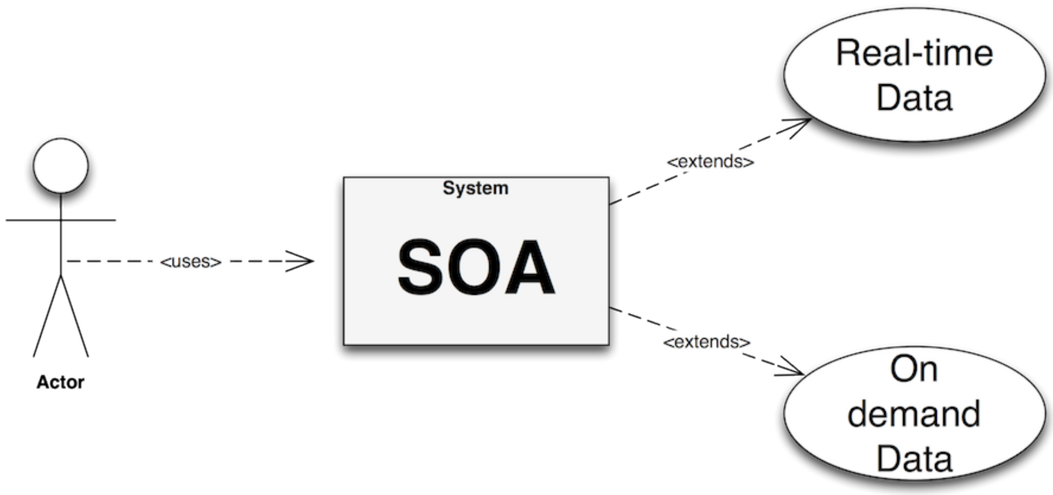


Figure 5.6: Uses cases available in OpenWeather via SOA.

Real-time data messages flow is considered to be as a continuos service offered by the AWS via OpenWeather. Additionally, the possibility to retrieve saved data in the AWS exits. Both real-time data and data on demand, is sent and retrieved through OpenWeather data message system, using JSON. Thus, OpenWeather offers the same possibilities as the common methodologies currently used by the vendors explained in chapter four, moreover the chance to get real-time data through a reliable and efficient way.

5.3 Summary

In this chapter we gave an introduction of OpenWeather, highlighting the general guidelines applied in its design. We exposed some of the principles used in OpenWeather nodes we considered some possible examples of future applications using OpenWeather.

We introduced the areas in which OpenWeather can have a contribution or impact. Projects as GOS or SMEAR[40] seeking for new technologies for data acquisition, could get a positive use of OpenWeather concepts.

In addition, the basic functionality of the protocol, such as its architectural principles or software model implementation have been introduced as well.

Chapter 6

Protocol specification

In this chapter, the OpenWeather protocol specifications are explained.

6.1 Definitions

The following subsections summarized the role of the elements involved in the protocol. Some of the definitions are widely used in other protocols.

OpenWeather network

The nodes used in OpenWeather protocol conform to OpenWeather network standards. Inside of this network a node is able to interact with other nodes, requesting and delivering services to other nodes. These services are oriented to provide weather data. Because OpenWeather is based in a P2P architecture, its topology is decentralized. This topology makes the nodes independent of central nodes in order to interact between them.

OpenWeather node

A node is an active or passive element connected to OpenWeather network. One node can offer none to multiple services. An element is considered a node when it has a working implementation of OpenWeather protocol and is connected to the network.

Peer

Every node is considered a peer of OpenWeather network. All nodes in OpenWeather network are able to be clients and servers at the same time. This is establishing the basis of the P2P architecture used in OpenWeather. A peer must be able to offer services to others peers, however it is not mandatory to offer a service[1] in order to be connected to OpenWeather's network.

Weather data

The purpose of OpenWeather is to create a network in which the data exchange comes from the weather data sources. To obtain this data the nodes can be connected to an AWS or other system of weather data collection. OpenWeather does not differentiate between the original source of the instrument's brand, because data normalization[2] is required in order to make the data network available.

6.2 Architecture

As it is mentioned in section 5.2.1, the architecture used in OpenWeather matches the requirements mentioned in the RFC 5694 [8], with OpenWeather containing nodes offering and requesting services between them.

The technical reasons why a P2P architecture is a better network solution for a topology as define by default by the AWS, are supported in the following points:

- An AWS is an individual entity being part of a bigger network that does not need a centralized model except for data processing.
- The process executed over the weather data in order to extract meaningful conclusions does not posses a technical requirement to be linked to the network layer.
- The collection point model forces the node to depend exclusively on one node in the network, adding unnecessary risks to the data flow.
- The common architecture used in the weather data flow, is forced by the legacy of the protocols used within it.

The P2P architecture is chosen by OpenWeather because it brings autonomy and robustness to the nodes. In addition, it provides the network the capability to scale and to share resources without single dependencies. Moreover, the geographical situation of the nodes, is suitable for developing models in which the nodes can collaborate to distribute the data. Finally, the P2P architecture provides the capability to retrieve data directly from the node, without going through a common point that can be collapsed or not available.

[1]Doing reference here to high-level services related with the data delivery, de facto, a peer is always offering a minimum amount of services integrated within the protocol, needed to interact in the network.

[2]As it was explained in chapter 3 and chapter 4, this step is required because it is not possible to modify AWS's OS without the vendor's collaboration.

6.2.1 Standards used for data units

OpenWeather does not provide the weather date measurement units. The protocol is designed to deliver weather data formatted according to the data units specified in International Standard Organization (ISO) 80000 [27] family and the *Guide to Meteorological Instruments and Methods of Observation* [46].

Table 6.1 provides the data units used in the prototype:

Data field	**Data unit**	**Acronym**
Air-Temperature	Celsius	C
Relative-Humidity	Percentage	% RH
Air-Pressure	Hectopascals	hPa
Wind direction	Degress	degrees
Wind speed	Meters per second	$\frac{m}{s}$
Rain accumulation	Millimeters	mm
Rain duration	Seconds	s
Rain intensity	Millimeters per hour	$\frac{mm}{h}$
Rain peak	Millimeters per hour	$\frac{mm}{h}$
Hail accumulation	Hits per square centimeter	$\frac{Hits}{cm^2}$
Hail duration	Seconds	s
Hail intensity	Hits per square centimeter per hour	$\frac{Hits}{cm^2h}$
Hail peak	Hits per square centimeter per hour	$\frac{Hits}{cm^2h}$

Table 6.1: Data units implicit on the data fields.

Since the data units have a known standard, the author considers that it is not necessary to increase data messages sizes and data fields, but only to provide the data units. Instead, it is more pragmatic and efficient to assume that weather data will be supplied with appropriate data units. It is necessary to highlight that despite the absence of network protocol for weather data, the vendors maintain a strict control of data units used in the AWS, facilitating this the implementation of OpenWeather across vendors.

6.2.2 Nodes

A node connected to a OpenWeather network behaves as a deterministic finite automaton, not executing without a clear definition operations or a definite result. All the operations performed by the nodes are identified by codes placed in the MetaInfo data field. Any data message delivered in OpenWeather protocol contains all information[3] required to identify the type of operation to be performed by the software when the data message is received / delivered.

[3]Through the protocol code.

Any data requested or delivered by a node using OpenWeather is based on a request and a confirmation of it. With this mechanism the nodes are notified of status of the operations of execution in the application layer are successful or not. This same mechanism is implemented in protocols as HTTP[22] in order to control the status of retrieval and delivery operations.

A node is able to interact with multiple nodes, being only limited by the bandwidth and system resources availability. OpenWeather does not define a minimum or maximum of connections needed, however a node requires a $>=$**1** number of peers on its internal list in order to interact with OpenWeather network.

Automatic Weather Stations as individual nodes

The section 3.1.2 explains how the AWS are categorized as embedded systems. By the definition, an embedded system has certain limitations in data processing and data delivery. Nevertheless the AWS are still able to do some networking operations and data processing when the size of them is small. OpenWeather has been designed to work around these limitations.

Taking this as a basis, OpenWeather transforms the centralized model currently used by the industry, to a decentralized model taking advantage of a P2P architecture. OpenWeather considers every AWS as a node using SOA. Because the AWS are under constant connection and deliver data to collection points, the only modification needed in the equipment is to change the network protocols used to deliver this data[4].

Instead of using an architecture in which the AWS plainly sends the data over the network without any further interaction, OpenWeather provides the mechanisms to convert the AWS to an entity able to respond to the data requests made by the user in real-time. Although all of this process can be handled through the centralized model, the independence of nodes from the collection point is mandatory in order to achieve scalability and data accessibility. For instance, a user located outside of a specific network of AWS, can not access the data produced by them without the need to go through the collection point[5], this use case avoids any possibility to combine data in real-time from different AWS in different geographical locations, restricting any possibility to interact directly with the AWS.

Enabling the AWS to behave as a nodes, the protocol provides the basis to take advantage of the real-time data and at the same time fix the issues exposed in chapter four. Though this thesis sets an ambitious goal: the transition from a centralized model to a decentralized model, it has to be noted that the industry has been using the same technologies for decades, not taking advantage of the improvements made in the networking technologies, concerning data delivery and

[4]An adaption of the AWS's OS will be required in order to integrate the OpenWeather's stack inside of the AWS.

[5]If the AWS work but the collection point is down, the data will not be accessible.

acquisition. The decentralized models have a proven successful track, offering scalability and robustness.

As any other network protocol, OpenWeather has a defined set of operations. These operations provide the core principles to deliver and retrieve data from nodes. However, these principles do not need to contain the whole data flow.

Super-nodes

OpenWeather refers to super-nodes to those nodes with static IP / hostname, which are always available to exchange peer lists. Unlike other P2P applications, an OpenWeather super-node does not have any other extra property, except its bandwidth availability [6] and an updated list of peers, to deliver to the other nodes. The role of a super-node is to be always available and to provide updated peer lists to those nodes without one. This is enough to guarantee that the nodes will be able to connect to it if they can not find other nodes available.

Peer list calculation algorithm

One of the biggest challenges of the P2P architecture is to identify which peers are superior to others. This issue is mostly found in those architectures in which the purpose of the network is to transfer data based on user reputation[7]. Since all nodes are consider peers containing unique data, OpenWeather does not make distinction amount them.

Even so, for practical reasons, it is necessary to develop an algorithm to calculate which peers are better than others in terms of connectivity and bandwidth availability, to provide a list to the nodes to guarantee the connection to OpenWeather network.

The author considers that due to the nature of the data and the main factor of its importance is availability. Thus, the algorithm shall be a node bandwidth, network latency and geographical location.

Bandwidth and latency are two obvious and common used factors in other P2P architectures. However in this case is important to note that most of these nodes are going to have better network visibility with nodes are located in proximity. The geographical location of the node, available in the MetaInfo data field through the "Location" data field, can be used to calculate the closest peers.

The algorithm to calculate the best peers to keep on the internal list, is too a vast and complex topic to be analyzed in this thesis. In the prototype created, the author used random peers in order to verify the protocol specifications. It is

[6]It must be higher than average so that it may process higher network traffic.

[7]Meaning the amount of data shared and uploaded to other peers.

necessary to highlight that the peer list calculation must be analyzed deeply in order to implement OpenWeather in production scenario.

Node identification

In section 5.2.1 is mentioned that a node **has a unique ID**. This ID is used to identify the node and at the same time by the user/software to recognize which node is currently active. The value of this ID is based on SHA-256[34]. Nevertheless, the length of it and its alphanumeric composition make it really difficult to remember the node ID, even when using some mnemonic techniques. However, it can be easily fixed with a proper algorithm, based on a standardized AWS system for identification and use of the RFC 3986 *Uniform Resource Identifier (URI): Generic Syntax* [6].

As example, the CWOP uses different parameters[31] to identify the AWS; some of them are:

- Block number 2 digits representing the WMO-assigned block
- Station number 3 digits representing the WMO-assigned station
- Place name: common name of station location
- Country name: country name is ISO short English form

The block number refers to the geographical region[8] of the AWS, and the station number is assigned base on *the nearest 10 degree meridian which is numerically lower than the station longitude*[31]. The place name and country name are values assigned based on the geographical location of the AWS. Although CWOP also provides the latitude and the longitude, their introduction in Uniform Resource Locator (URL) generation, will cause greater complexity.

02;974;EFHK;Helsinki-Vantaa;;Finland;6;60-19N;024-58E;60-19N;024-58E;51;56;P

Table 6.2: Example of CWOP's AWS identification.

The table 6.2 shows all data used by CWOP to identify an AWS, the the following syntax is used to generate the URL:

owp://Country Name/Place Name/Block number + Station number

Table 6.3: ID partially based on CWOP notation.

[8]Extracted from station index numbers database, CWOP Meteorological Station Location Information [31].

Based on the data used in the table 6.2 the output will be:

owp://finland/helsinki-vantaa/02974

Table 6.4: ID's partially based in CWOP's identification system.

The scheme is denominated as **owp** (OpenWeather Protocol), the authority field is used for country name, the absolute path is based on the place name and the station number assigned by the WMO. This combination is enough to guarantee the uniqueness of the node accessed through the URL.

The value of the ID used in the OpenWeather data message will be the resulting hash of the data "02;974;Helsinki-Vantaa;;Finland" generated with SHA-256.

```
{
    "OpenWeatherMessage": {
            ...
"ID" :"a88a9b6b4c0381e0509ce36cadb5fd06e5446ab23881020b9f212db24b16ee75",
            ...
},
```

Table 6.5: IDs based in the SHA-256 result of the CWOP notation.

6.3 Protocol operations

The protocol allows the following operations:

- Session establishment
- Service discovery
- Real-time data retrieval
- Data on demand

Note that all of these operations have an implicit internal functional workflow, based on the requests and retrievals and their results. The following sections analyze the functioning of these operations.

6.3.1 Session establishment - Peer handshake

The first operation needed for OpenWeather is session establishment. The elements involved in this operation can go from **2..n** nodes. Thus, a node can execute the operation to establish session with multiple nodes at the same time, nevertheless, the session establishment is always an isolated process between two nodes.

These nodes must offer the basic services integrated in the protocol, as peer exchange information or peers-list exchange. The session establishment between nodes is denominated **peer handshake**. At this point the nodes exchange their information in order to identify each other, sending a data message with the parameters mentioned in section 6.2.4. This operation is categorized as an internal protocol requirement, using the code **100** as type of data message.

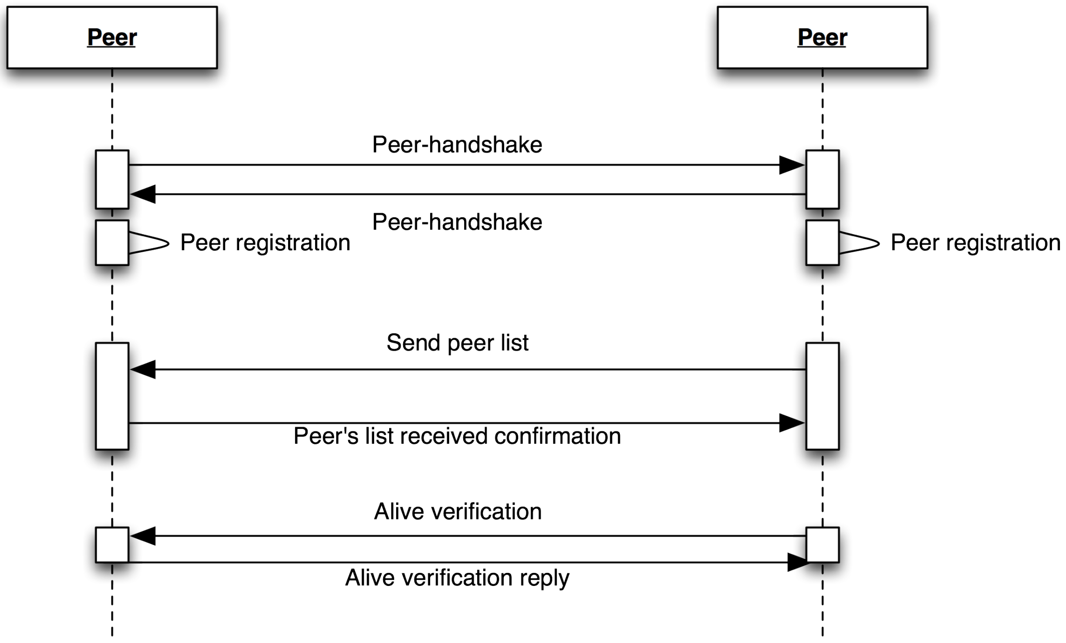


Figure 6.1: Session establishment sequence diagram.

When the nodes establish a session, two operations are performed

- Peers-list exchange.
- Alive verification.

The first operation —peers-list exchange— is performed in order to verify if the nodes can update their internal list of peers available.

The second operation performed is alive verification. The peers send a data message after the exchange of the peer list, in order to verify that the nodes are ready to request weather data[9]. If the alive verification is not successful, the node executing it will close the TCP connection with the node that is not responding to it.

[9]Note that this check is realized to ensure the availability of the node twice.

6.3.2 Service discovery

OpenWeather assumes that when two nodes establish a session, the purpose of it is to exchange certain data, even if it is just for protocol requirements. As it is explained in section 5.2.2, OpenWeather is designed according to a service oriented architecture SOA. All data sent or receive by a node goes through services provided in the OpenWeather software implementation.

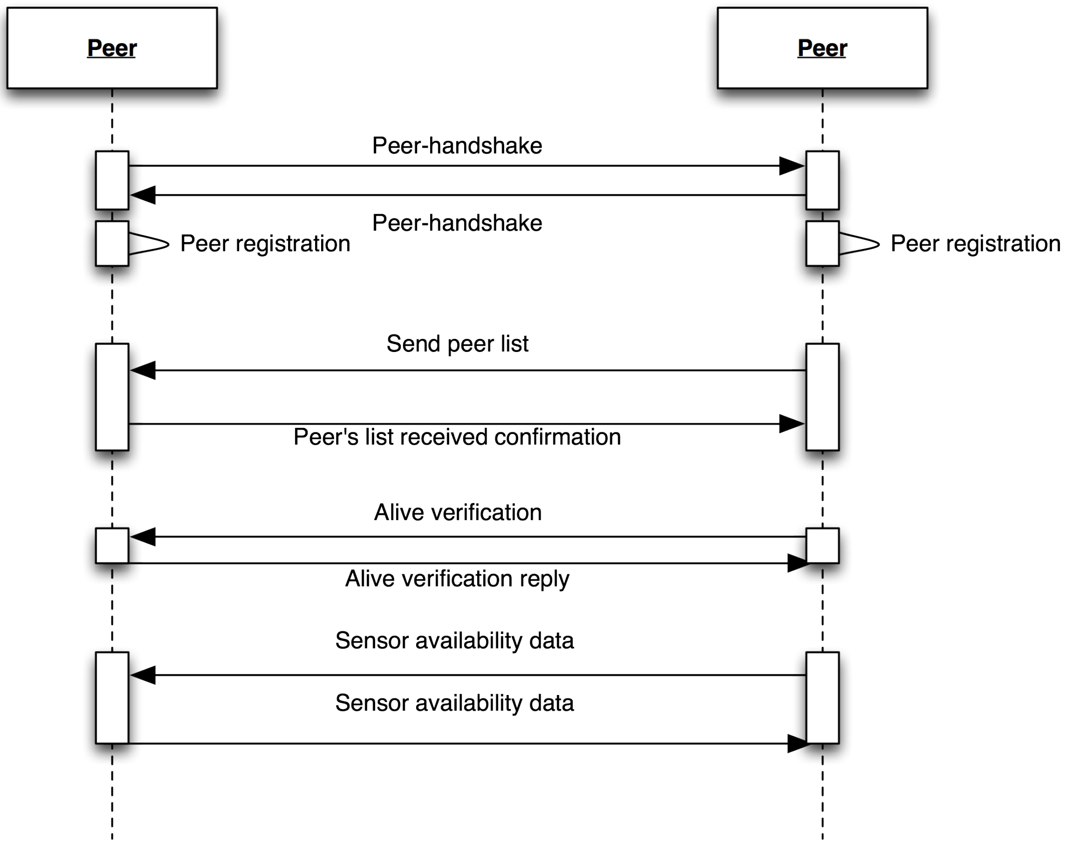


Figure 6.2: Service discovery sequence diagram.

The nodes involved in the session must exchange the type of data messages, in order to be aware of services available to the nodes. Note that this operation informs the nodes which sensors are available to other nodes and which kind of weather data can be retrieved from them. After the nodes communicate through the services available, other operations as real-time data retrieval or data on demand, can be performed.

6.3.3 Real-time data retrieval

When the nodes establish the session and service discovery operations is performed successfully, they are consider to be ready to send and receive weather data between them. As it is explained in section 5.2.2, the data can be real-time data or data on demand. In case of real time data, the node requesting it, must send a type of data message with code **200**, immediately after, the other node involved in the session, must start to delivery real-time data messages.

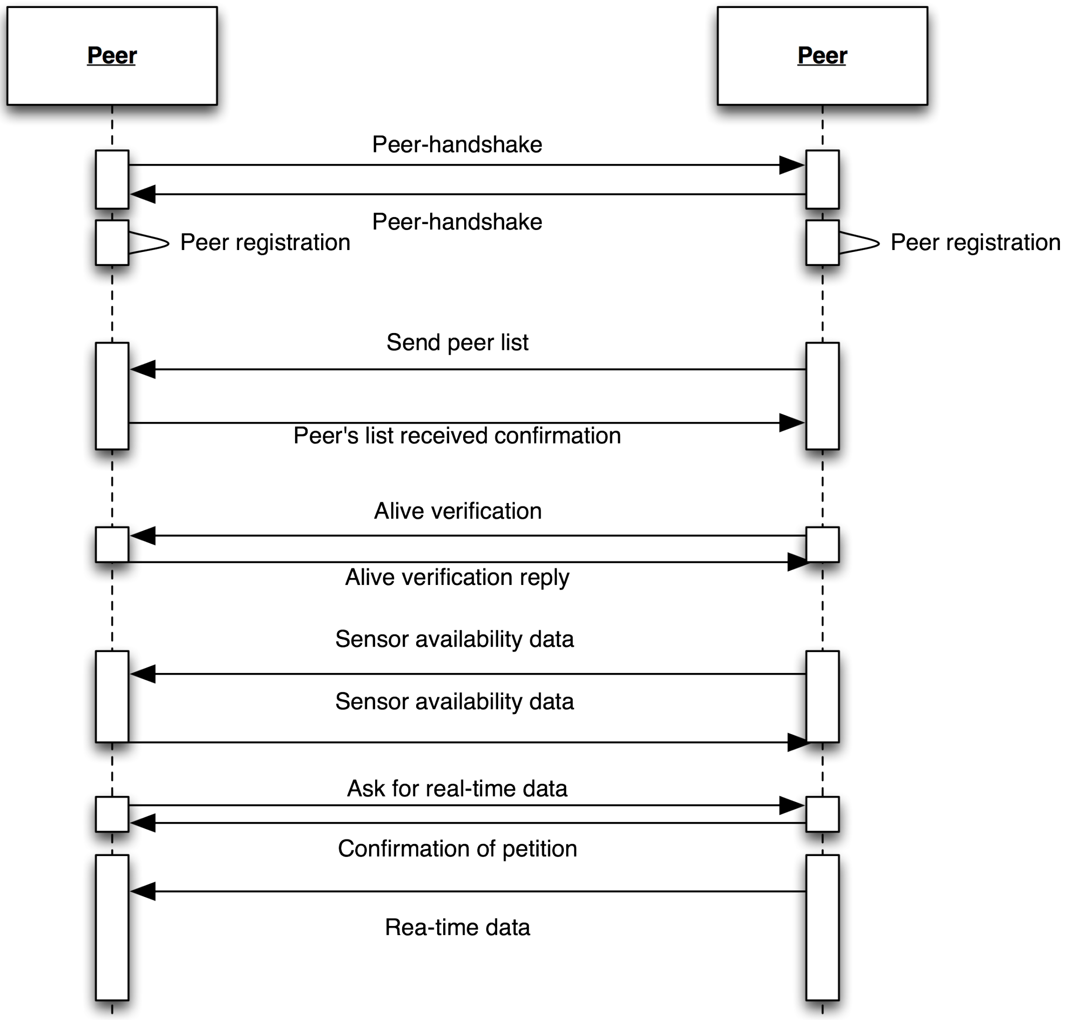


Figure 6.3: Real-time data sequence diagram.

The real-time data will be deliver until the node requesting it decides to stop the data stream[10]. This data stream provides the real-time data generated in the remote AWS. As in any other network solution the delay that the nodes can experience can affect the delivery of the data. Nevertheless, all the data messages are

[10]The data streams can be interrupted by other exceptions as connectivity or software issues.

timestamped when the data was assembled within them. Because this timestamp is available, it will be feasible to implement an algorithm on the software side, applying a correction factor to the timestamp based on the latency of the nodes, to fix this issue.

6.3.4 Data on demand

Apart from the the real-time data, a user can request data on demand. When a user requests data on demand, it creates individual requests with a specific timestamp. Based on these requests, the remote node will deliver an individual data message timestamped with the date and time provided, the requests and the weather data on that time.

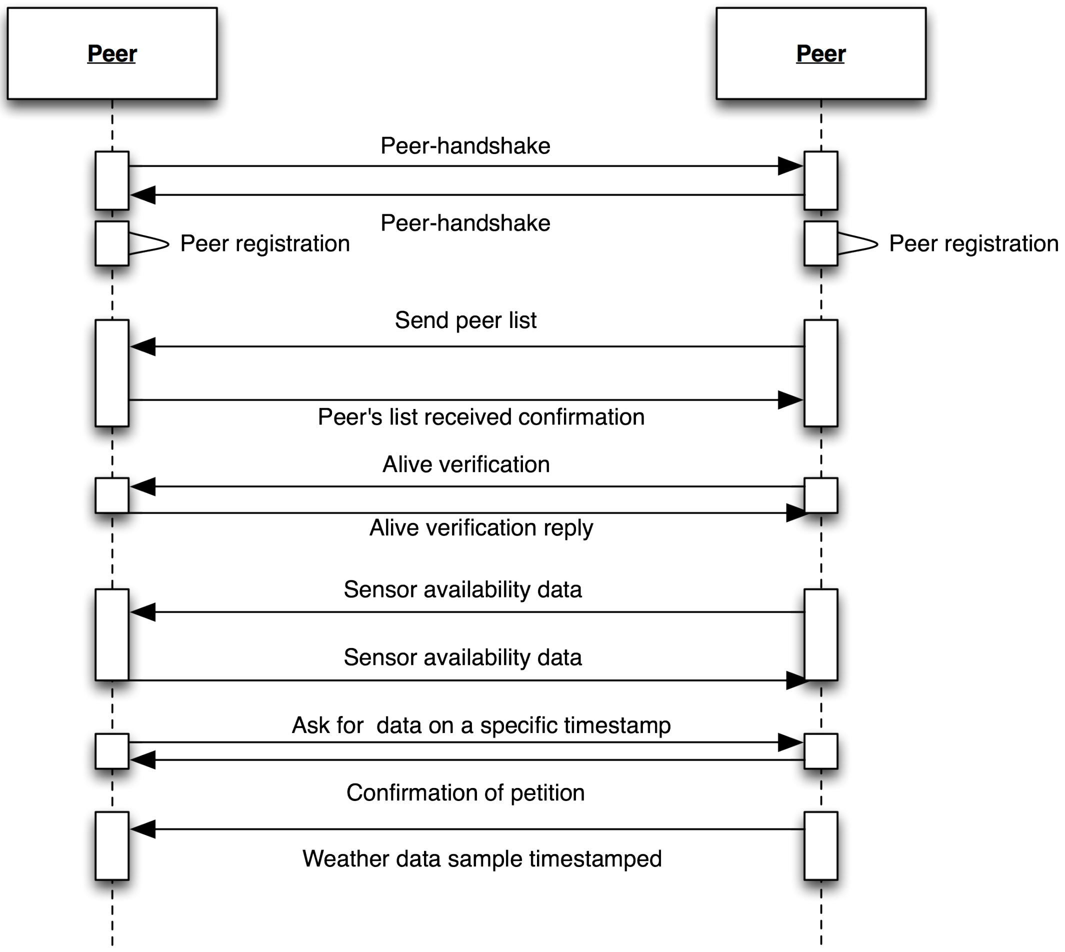


Figure 6.4: On demand data sequence diagram.

OpenWeather does not support the capability to request a range of dates or times on protocol level, meaning that it is not possible to retrieve isolated weather data samples form the node during a certain period of time. Instead, it is possible to implement on the software side the functionality to process a group of data messages with a certain timestamp. The justification of this limitation is based on the bandwidth availability in an AWS. In contrast with individual weather data samples, a range of them can have a considerable size and this can cause significant obstacles for the AWS: heavy CPU load, bandwidth consumption, etc.

6.4 Data messages

A data message refers to the data transmitted using the OpenWeather protocol. A data message can contain multiple informational values, referring to weather data or data needed for protocol maintenance.

All data fields contained in an OpenWeather data message are considered to be encapsulated data represented through JSON objects using Universal Character Set - Transformation Format (UTF)-8[13][47] as character encoding. According with the RFC 4627[16], the definition of an JSON's object is:

[...] An object is an unordered collection of zero or more name/value pairs, where a name is a string and a value is a string, number, boolean, null, object, or array.[...]

Therefore, any data field contained in an OpenWeather data message, is an individual or group of JSON objects or values. These objects are optimized according to the data that they contain. For instance, some data fields are JSON objects containing other objects at the same time. The data optimization made in the protocol using these data structures, allows data encapsulation which makes enables a fast data the data processing from the network to software levels.

All OpenWeather data messages are formatted using JSON syntax. Type of data contained in the data message is insignificant as it is structured in one JSON object composed for different sub-objects. These objects are represented as data fields in terms of networking architecture.

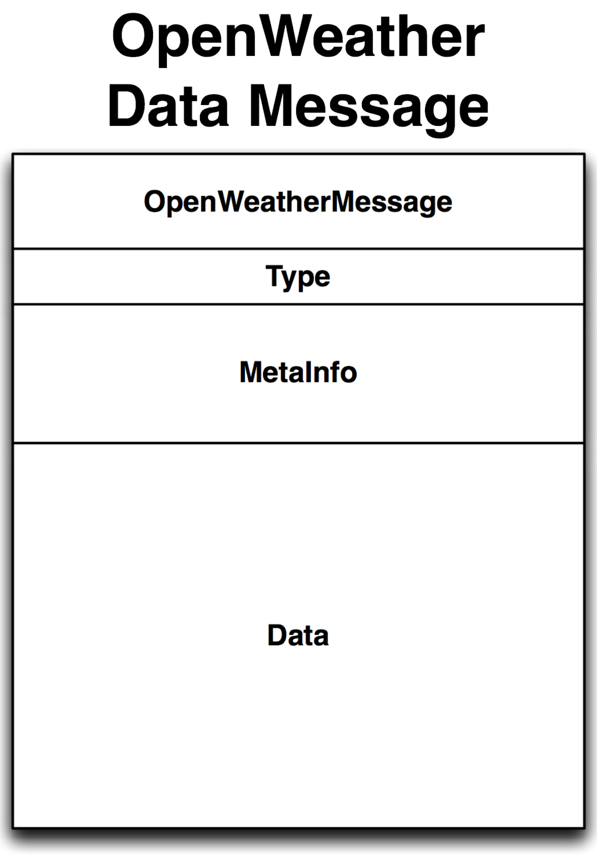


Figure 6.5: OpenWeather data message structure.

The parent object is denominated **OpenWeatherMessage**; this object **is present in all the data messages** inside of OpenWeather network. This parent contains two sub-objects; the MetaInfo object and the Data object or Info object. The MetaInfo object is a data field acting as **the header of the data message in OpenWeather protocol**[11]. Furthermore an OpenWeather data message contains the Data object or the Info object. The Data object is a data field containing all data related to the weather data that the data message transports. The Info object contains the information used internally by the OpenWeather protocol.

6.4.1 Header

OpenWeather uses a fixed[12] header data field in all the data messages, in order to guarantee its functioning. The function of this header is to provide all necessary data parameters needed by the OpenWeather protocol in every data message.

[11]This object is added an individual data field named "Type", explained in the next section.
[12]In terms of data fields provided.

Though it requires some data repetition, its insignificant size of this header, compensates the disadvantages of its repetition during transmission.

Table 6.4.1 shows the fields contained in the header:

```
{
    "OpenWeatherMessage": {
        "Type" : "",
        "MetaInfo" : {
            "ID" : "",
            "Peer-IP": "",
            "Port": "",
            "Location": "",
            "Update-Interval": "",
            "Peers-request":"",
            "Keep-Alive":"",
            "Bandwidth": "",
            "Timestamp" : "",
            "Version" : "",
        },
```

Table 6.6: Header field (Header object) in a data message of OpenWeather.

As the table 6.4.1 exposes all data messages start with the term ”OpenWeather-Message”, building JSON parent object of the data message. Any data contained within the data message will belong to this parent object. Although this hierarchy does not impact the data message size, it provides significant assistance to the post processing of the data on the software side. This design is inspired by the same concepts use in eXtensible Markup Language (XML) and XML Schemas [14], concerning the metadata fields. Nevertheless, OpenWeather does not providing any extra fields for metadata definition, meaning that the software utilizing OpenWeather, should recognize the expected format beforehand[13]. With this practice speed up and simplifies the parsing compare to XML.

6.4.2 Types of data messages

The second field contained in an OpenWeather's message is denominated **Type**. This field indicates which type of data is located within a data message through a numerical code and if it is related with weather data, peers exchange, protocol itself, etc.

Depending on the type of data message it will be in one of the following categories:

[13]XML allows data type provision in the data itself. However, this practice increases the size of the data considerably.

- Data messages for protocol maintenance only.
- Data messages use to transport weather data only.
    - Real-time data.
    - Data on demand.

6.4.3 Protocol codes

The "Type" field can contain a numerical value from **1..n**. This numerical value is known as the protocol code associated with the type of messages. The codes used are divided in categories and subcategories:

- Codes assigned to data messages used for protocol maintenance.
    - Protocol codes (From: 1..1xx)
        - Requests
        - Retrievals
        - Status
            - Success
            - Error
- Codes assigned to data messages for weather data exchange between peers:
    - Peer codes
        - Requests
            - Real-time data: 200
            - Data on demand: 201
        - Retrievals
            - Real-time data: 300
            - Data on demand: 301
        - Status
            - Success: 500..599
            - Error: 600..699

The numerical value is used by the software in order to recognize the data processing procedure.

All the protocol codes used in the prototype are available **in the appendix**.

6.4.4 MetaInfo data field

The MetaInfo data field (MetaInfo JSON object) defines fixed data fields transmitted in every data message. The purpose of these fields is to provide all information needed, in order to identify the peer's ID, its geographical location, IP address, among other data. The use of this data throughout all data messages makes allows for easier implementation and extensibility of the P2P architecture, as it enables the software to be aware properties and status of a specific peer at all times.

The MetaInfo field contains the following data fields:

- Bandwidth
- ID
- Keep-Alive
- Location
- Peer-IP
- Peers-Request
- Port
- Timestamp
- Update-Interval
- Version

The MetaInfo data field is structure as a JSON object containing an array of elements[14]. These elements are the fields mentioned above. Every element does reference to an specific parameter needed by OpenWeather protocol.

```
{
"OpenWeatherMessage": {
        "Type" : 1,
        "MetaInfo" : {
              ARRAY DATA ELEMENTS
        },
},
```

Table 6.7: MetaInfo field in a data message of OpenWeather protocol.

[14]**Note:** in the following figures the expression "ARRAY DATA ELEMENTS" is used to refer the MetaInfo data fields.

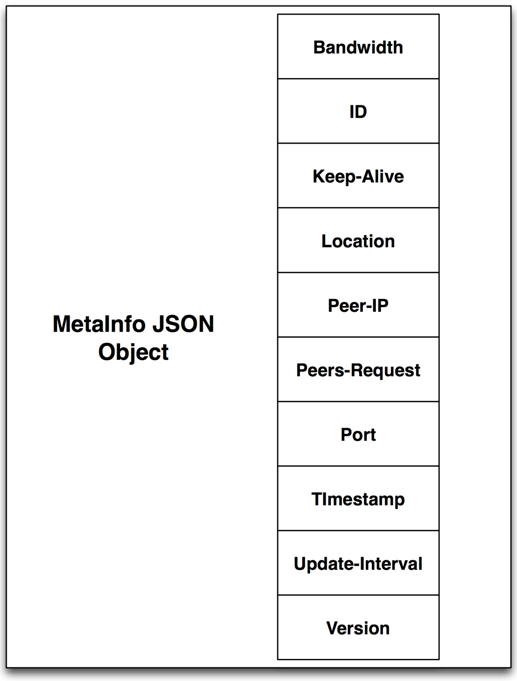


Figure 6.6: OpenWeather MetaInfo data field with data array elements.

Bandwidth

As any other network oriented software, the amount of bandwidth is a critical factor in its proper functionality. Most software solutions using P2P architecture offer a dedicated section to control the bandwidth parameters. OpenWeather informs others nodes of the amount of bandwidth that a node has available while giving full control of the amount of bandwidth and connections and remote connections allow. As opposed to mainstream solutions, in which the node is only controls the amount of connections and bandwidth locally, the bandwidth control in OpenWeather can be managed both locally and remotely. To achieve this, the field "Bandwidth" is provided in every data message, informing the nodes what is the capacity of the node whereby are operating.

```
{
    "OpenWeatherMessage": {
            ...
            "Bandwidth" : "4", // Correspondency 1 Megabit/s
            ...
},
```

Table 6.8: Bandwidth field in a data message of OpenWeather

The user must provide this parameter to configure its node. Due to the a big amount of possibilities for bandwidth quality, this data field contains a numeric value that should be translated by the software to bits per second. Nevertheless, if the user considers that its bandwidth does not fit in the categories provided, it is possible to provide an integer number that will be translated by the software to bits per seconds. Thus, if the "Bandwidth" data field contains a numeric value higher than 6, the value will be translated for the software to bits per second. This feature allows the user to use a custom parameter.

Numeric value	**Bandwidth equivalency**
0	56 kbit/s
1	128 kbit/s
2	256 kbit/s
3	512 kbit/s
4	1 Mbits/s
5	10 Mbits/s
6	100 Mbits/s

Table 6.9: Bandwidths equivalency in Bandwidth data field.

ID

As explained in sections 5.2.1, every peer has an unique ID throughout OpenWeather's network. In fact, its properties make it theoretically unique in the world.The ID is generated based in the AWS identification. The ID data field is thought to be representation of the AWS, such representation is the result of the hash applied over some identification system for AWSes[15].

If the AWS is not part of some identification system, its ID can be generated randomly by the software, however is highly recommended to provide an ID assigned for some organization as the CWOP or NOAA.[16].

```
{
    "OpenWeatherMessage": {
             ...
"ID" :"4f9a67e8496d69b8707858576ec12b8aa3fa5519c23a79ea071dc7dbc0c9b2e3",
             ...
},
```

Table 6.10: ID's field in a data message of OpenWeather protocol.

[15]Several weather organizations provide this identification.

[16]In the evaluation setup, the author uses a random ID.

Keep-Alive

Due to possible node connection instability, it is necessary to implement a mechanism to identify the current connection status with a specific node is, on the application layer level. OpenWeather implements the field "Keep-Alive". This field provides the amount of time that the software must wait until the connection is close.

```
{
    "OpenWeatherMessage": {
            ...
            "Keep-Alive" : "120000",
            ...
},
```

Table 6.11: Keep-Alive field in a data messages of OpenWeather protocol.

When a node stops sending data to other node/s, the connection will be closed when the sum of the timestamps of the last data messages received and the "Keep-Alive" value, is less than the current date and time.

The protocol assumes that if the node is not delivering data, is not useful to keep a connection with it. The same principle is applied in a number of network oriented software solutions. The value of this field is expressed in milliseconds, and by the default has a timeout of **120000 milliseconds** (2 minutes). Though possible, the customization this parameter is not recommended, as it assumes responsibility between nodes when necessary.

Location

The "Location" field does reference to the geographical coordinates of the node, expressed in the Universal Transverse Mercator (UTM) system. This data field has two different functions:

- Identify the geographical location of the node.[17]
- Provide identificational information to other peers, does providing them with the updated information which store in the node's internal list. [18]

[17]Mandatory due to the nature of the data.
[18]This is explained deeper in section 7.2.

```
{
    "OpenWeatherMessage": {
            ...
            "Location" : "4597807 269999 30T",
            ...
},
```

Table 6.12: Location field in a data messages of OpenWeather protocol.

This field should be filled manually by the user. It is highly recommended to provide this parameter with as much accuracy as possible.

Peer's IP address & port

The MetaInfo's field contains two fields dedicated to TCP/IP:

- Peer-IP
- Port

The field "Peer-IP" contains the **public** IP address assigned to the computer's network interface that is running the software supporting OpenWeather's protocol.This field can be an IP address using 32-bit number (IPv4) or 128 bit number (IPv6). The introduction of this field is based on the requirement of the protocol to possess an updated address of the peer in order to able to connect to it. Though the field is labeled as "Peer-IP" not necessarily must be the numeric address. It is possible to implement the OpenWeather protocol to use hostname resolution based on Dynamic Name Server (DNS) requests[19], with a few modifications on the software's side.

The field "Port" contains the port used in the TCP to establish a connection with the peer. The default TCP port number is **62535**[20][21]nevertheless any port can be used inside of TCP's range always that it does not conflict with other ports.

[19]However as it is implicit in the use of DNS, it will be required to have the hostname of the peers recorded in the name servers.

[20]Port number choose according with the range of ports available for dynamic and/or private use published by IANA[5].

[21]We assume fixed ports and port forwarding techniques for this. The functioning of OpenWeather behind firewalls or/and NAT is out of the scope of this thesis.

```
{
    "OpenWeatherMessage": {
            ...
            "Peer-IP" : "140.186.70.148",
            "Port": "62535",
            ...
},
```

Table 6.13: Peer-IP & Port fields in a data message of OpenWeather protocol.

Both fields, "Peer-IP" and "Port", are present in others P2P architectures[22], the reason for this is that these fields facilitate a significant part of the software implementation and the network functionality. Adding these fields to all data messages, enables the software to keep the peer list updated and working between nodes and at the same time it facilitates the protocol session establishment.

Peers-Requested

The "Peers-Requested" field provides the number of peers that the node requests to other nodes in order to fill its internal list of peers. In a P2P architecture it is critical to keep an updated list of peers to guarantee successful delivery of the data throughout the network. By default this field is set to **20**, with a possible range of **1..100**.

```
{
    "OpenWeatherMessage": {
            ...
            "Peers-Requested" : "20",
            ...
},
```

Table 6.14: Peers-Requested field in a data messages of OpenWeather protocol.

Timestamp

As explained in chapter two, the success of weather prediction depends on different factors. One of the most important variables are the geographical location and the time and date, in which the weather data samples were collected. OpenWeather provides the field "Timestamp" to supply a solution for this condition. Every data message contains the timestamp in which the data was assembled. This provides

[22]Often denominated with different terms and syntax.

a feasible mechanism to know when the weather data sample by the data message received was collected.

The data format used by OpenWeather protocol follows the RFC 3339 (Date and Time on the Internet: Timestamps)[28] and it follows the guidelines established by the ISO 8601:2004[20] as well. All data messages are timestamped using the Coordinated Universal Time (UTC).[23]

```
{
    "OpenWeatherMessage": {
            ...
            "Timestamp" : "2011-05-29T12:10:23Z",
            ...
},
```

Table 6.15: Timestamp field in a data message of OpenWeather.

Note that OpenWeather protocol does not use the timestamp value for any purpose related with protocol operations. This Timestamp field is provided in order to fit the requirements of the weather data. Because the weather data requires precise stamping of the time in which it was acquire, this field is introduced. In addition, as in other real-time data systems, it is recommended to sync the time of the node using protocol such as Network Time Protocol (NTP), to guarantee the quality of the data. Such synchronization must be managed independently of OpenWeather.

Update-Interval

The "Update-Interval" field contains the time value, expressed in **milliseconds**, that other peers should wait before to requesting protocol information. This field can be used to manage data availability provided absence of network congestion.

```
{
    "OpenWeatherMessage": {
            ...
            "Update-Interval" : "120000",
            ...
},
```

Table 6.16: Update-Interval field in a data messages of OpenWeather protocol.

[23]The conversion to the original timezone of the data message can be managed through software.

By default this field is set to **120000 milliseconds** (2 minutes), however this parameter that can be customize by the user.

Protocol versioning

Following the same principles as HTTP and other protocols, OpenWeather uses **<major>.<minor>** numbering scheme to indicate the versions of the protocol. The versioning is indicated in the "Version" field of the data header, adding the term ”OpenWeather” and the character ’/’ before the numbering.

```
{
    "OpenWeatherMessage": {
            ...
            "Version" : "OpenWeather/1.0",
            ...
},
```

Table 6.17: Version field in a data message of OpenWeather.

MetaInfo data field summary

The table 6.18 shows the structure of the MetaInfo data field with all array elements already filled in with data:

```
{
    "OpenWeatherMessage": {
        "Type" : 1,
        "MetaInfo" : {
"ID" :"4f9a67e8496d69b8707858576ec12b8aa3fa5519c23a79ea071dc7dbc0c9b2e3",
            "Peer-IP" : "140.186.70.148",
            "Port": "62535",
            "Location" : "4597807 269999 30T",
            "Update-Interval" : "120000",
            "Peers-Request" : "20",
            "Keep-Alive" : "120000",
            "Bandwidth" : "4",
            "Timestamp" : "2011-05-29T12:10:23Z",
            "Version" : "OpenWeather/1.0",
            },
            ...
}
```

Table 6.18: MetaInfo data field (MetaInfo object) in a data message of OpenWeather.

All the OpenWeather data messages will contain a header as the shown in the table 6.18, fill in with the particular data of the node.

6.4.5 Data field

As part of the MetaInfo data field (MetaInfo object), OpenWeather data messages can contain a field named Data (Data object). This data field is a JSON object composed from different sub-objects. The values or sub-objects having this object as a parent, are dedicated to transport weather data.

The Data field is necessary in order to complement the MetaInfo data field. The MetaInfo data field only provides information about the node itself. The data field contains the data that the node retrieves or request from others nodes. The type of data available in this data field can be:

- Real-time weather data
- Data requested/delivery in demand (non real-time)

```
{
"OpenWeatherMessage": {
        "Type" : 1,
        "MetaInfo" : {
              ARRAY DATA ELEMENTS
        },
        "Data" : {
                       ARRAY DATA  OBJECTS
        }
},
```

Table 6.19: Data field in a data message of OpenWeather protocol.

All phenomena data transmitted in OpenWeather uses the data units, specify in the *Guide to Meteorological Instruments and Methods of Observation*[46], published by the WMO[44]. The author assumes that the protocol must follow this standard, because it is adopted by the major number of countries[24]. Though some countries still keep local units for measurements, OpenWeather protocol does not take in consideration these use cases, nevertheless the implementation of the conversion between units, can easily be done on the software side.

All values or sub-objects containing information about weather data **will always have the Data object as a parent..** The following sections explain how these different types of data are assembled in OpenWeather.

[24]Exceptions: United States, Liberia and Myanmar (Burma).

Real-time data messages

The section 6.4.4 introduced the persistent data provided in every data message of OpenWeather. However, this data is provided in order to guarantee the protocol's functioning. A part of the header field, the data messages can contain weather data. This section explains how a real-time message is assembled. Note that the prototype used in the experimental setup **only supports** data extracted from the following phenomena:

- **PTU -Pressure, Temperature, Humidity**
  - Air temperature
  - Relative humidity
  - Air pressure
- **Wind**
  - Direction (minimum, average, maximum)
  - Speed (minimum, average, maximum)
- **Precipitation**
  - Rain (accumulation, duration, intensity, peak)
  - Hail (accumulation, duration, intensity, peak)

These data have been chosen because it is available in most of the AWS of semi-professional / end-user range. In addition, the data used in OpenWeather provides a functional prototype adapted to this thesis. The author highlights that none of these data fields (concerning weather data) are used claiming them to be a standard or a suggestion of it. As mentioned in section 4.3, only a process of standardization can provide the correct data fields to use. Nevertheless, the use of these weather data fields is enough to develop a prototype.

Note that some data objects contain values in the data fields such as "minimum", "maximum" or "accumulation" among others, that are representing data collected in time intervals. Depending of the phenomenon these time intervals can be completely different. The recommend intervals of measurement are described in the *"Guide to Meteorological Instruments and Methods of Observation"* [46], and theoretically they must be always the same independently of the brand of the weather instrument used.

Pressure, temperature and humidity data

The PTU, are the most common data available in an AWS, due to the close relation between the phenomena and the ease of its acquirability. Any modern AWS will is equipped with necessary sensors to measure these phenomena.

The AWSes collect this data in real-time, transforming the raw input data from the sensors into digital data. The workflow of this data is described in section 3.1.4. As other data in OpenWeather, it will be normalized in the layer implemented between the hardware layer and OpenWeather[25].

0r2,Ta=18.7C,Ua=77.4P,Pa=1002.1H

Table 6.20: PTU real-time data in the raw format used by the AWS.

When the PTU data is transformed to OpenWeather's format, it has the following format:

```
{
"OpenWeatherMessage": {
        "Type" : 1,
        "MetaInfo" : {
              ARRAY DATA ELEMENTS
        },
        "Data" : {
            "PTU" : {
                "Air-Temperature" : "",
                "Relative-Humidity" : "",
                "Air-Pressure":  ""
            },
},
```

Table 6.21: PTU data field in a data message of OpenWeather protocol.

The three data fields contained in the Data object are:

- **Air-Temperature**: expressed in degree Celsius (°C)
- **Relative-Humidity**: expressed in percentage in base of relative humidity
- **Air-Pressure**: expressed in Hectopascals (hPa)

These data fields are encapsulated as an array of data elements inside of the JSON object PTU. The table 6.22 shows an example of the PTU object filled with real-time data:

[25]Explained in section 5.2.

```
         "Data" : {
             "PTU" : {
                 "Air-Temperature" : "20.0", // Celsius: ºC
                 "Relative-Humidity" : "59.5", // %RH
                 "Air-Pressure":  "1002.1" // Hectopascals: hPa
             },
```

Table 6.22: PTU data field with real-time data in a data message of OpenWeather protocol.

The frequency of reporting this data will depend on the configuration of the AWS. Most of AWS offer a time interval between **1 second** and **3 seconds**, to generate this data.

Wind data

The wind is other of the most popular phenomena to measure in AWSes. The wind data contains two sub-objects: direction and speed.

```
{
"OpenWeatherMessage": {
        "Type" : 1,
        "MetaInfo" : {
              ARRAY DATA ELEMENTS
        },
        "Data" : {
            ...
          "WIND" : {
                "Direction" : {
                    "min" : "",
                    "ave" : "",
                    "max" : ""
                },
                "Speed" : {
                    "min" : "",
                    "ave" : "",
                    "max" : ""
                }
            },
```

Table 6.23: Wind data field in a data message of OpenWeather protocol.

At the same time these two objects are composed by three array data elements:

- **Direction**
  - Minimum (min): expressed in degrees
  - Maximum (max): expressed in degrees
  - Average (avg): expressed in degrees
- **Speed**
  - Minimum (min): expressed in meters per second $\frac{m}{s}$
  - Maximum (max): expressed in meters per second $\frac{m}{s}$
  - Average (avg): expressed in meters per second $\frac{m}{s}$

```
{
"OpenWeatherMessage": {
        "Type" : 1,
        "MetaInfo" : {
              ARRAY DATA ELEMENTS
        },
        "Data" : {
            ...
          "WIND" : {
                "Direction" : {
                    "min" : "217", // Degrees
                    "ave" : "217",// Degrees
                    "max" : "218"// Degrees
                },
                "Speed" : {
                    "min" : "4.2",// m/s
                    "ave" : "4.2",// m/s
                    "max" : "4.5"// m/s
                }
            },
```

Table 6.24: Wind data field with real-time in a data message of OpenWeather protocol.

Precipitation data

Precipitation is the last phenomena that typically all the AWSes measure. Inside of the concept of precipitations encompasses two different classes, rain and hail. Thus, the precipitation is structure two sub-objects containing an array of four data elements.

```
{
"OpenWeatherMessage": {
        "Type" : 1,
        "MetaInfo" : {
              ARRAY DATA ELEMENTS
        },
        "Data" : {
            ...
       "PRECIPITATION" : {
                "Rain" : {
                    "accumulation" : ""
                    "duration" : "",
                    "intensity" : "",
                    "peak" : ""
                },
                "Hail" : {
                    "accumulation" : "",
                    "duration" : "",
                    "intensity" : "",
                    "peak" : ""
                }
            }
```

Table 6.25: Precipitation data field in a data message of OpenWeather protocol.

Both of them are measured with the same data fields:

- **Rain**
  - Accumulation (accumulation): expressed in millimeters
  - Duration (duration): expressed in seconds
  - Intensity (intensity): expressed in millimeters per hour
  - Peak (peak): expressed in millimeters per hour
- **Hail**
  - Accumulation (accumulation): expressed in hits per cm^2
  - Duration (duration): expressed in seconds
  - Intensity (intensity): expressed in hits per cm^2
  - Peak (peak): expressed in hits per cm^2

Compared with the PTU or wind, precipitation may be absent in the current weather. It means that the measurement of these phenomena will happen only when it is present. Despite this, OpenWeather always delivers the precipitation data field in the real-time data messages[26].

```
{
"OpenWeatherMessage": {
        "Type" : 1,
        "MetaInfo" : {
              ARRAY DATA ELEMENTS
        },
        "Data" : {
            ...
       "PRECIPITATION" : {
                "Rain" : {
                    "accumulation" : "12" // mm
                    "duration" : "34", // seconds
                    "intensity" : "12", // mm/h
                    "peak" : "9" // mm/h
                },
                "Hail" : {
                    "accumulation" : "2", //hits/cm^2
                    "duration" : "78", //seconds
                    "intensity" : "1", // hits/cm^2h
                    "peak" : "1" // hits/cm^2h
                }
            }
```

Table 6.26: Precipitation data field with real-time in a data message of OpenWeather protocol.

Data field overview

The table 6.27 indicates the Data field structure with all objects and their array elements filled with data:

[26]A zero value is assigned to the data fields when the phenomena are not present.

```
{
"OpenWeatherMessage": {
        "Type" : 1,
        "MetaInfo" : {
            ARRAY DATA ELEMENTS
        },
        "Data" : {
            "PTU" : {
                "Air-Temperature" : "20.0", // Celsius: C
                "Relative-Humidity" : "59.5", // %RH
                "Air-Pressure": "1002.1" // Hectopascals: hPa
            },
            "WIND" : {
                "Direction" : {
                    "min" : "217", // Degrees
                    "ave" : "217",// Degrees
                    "max" : "218"// Degrees
                },
                "Speed" : {
                    "min" : "4.2",// m/s
                    "ave" : "4.2",// m/s
                    "max" : "4.5"// m/s
                }
            },
            "PRECIPITATION" : {
                "Rain" : {
                    "accumulation" : "12" // mm
                    "duration" : "34", // seconds
                    "intensity" : "12", // mm/h
                    "peak" : "9" // mm/h
                },
                "Hail" : {
                    "accumulation" : "2", //hits/cm^2
                    "duration" : "78", //seconds
                    "intensity" : "1", // hits/cm^2h
                    "peak" : "1" // hits/cm^2h
                }
            }
        }
    }
}
```

Table 6.27: Real-time data message of OpenWeather protocol.

Data on demand

As highlighted in section 5.2.2, OpenWeather possesses the capability to transport data on demand (not being the data generated in real-time)[27]. In this use case, this data is only delivered by the nodes when the user requests it. To achieve this operation, OpenWeather uses the object's hierarchy, to know which kind of data the user is requesting. The protocol encodes such data in OpenWeather data header, after that it is interpreted by the software to localize the data requested from the AWS.

Note that the levels of hierarchy can be as deep as it is required. Nevertheless, the prototype only offers the possibility to retrieve the same data as in real-time.[28]

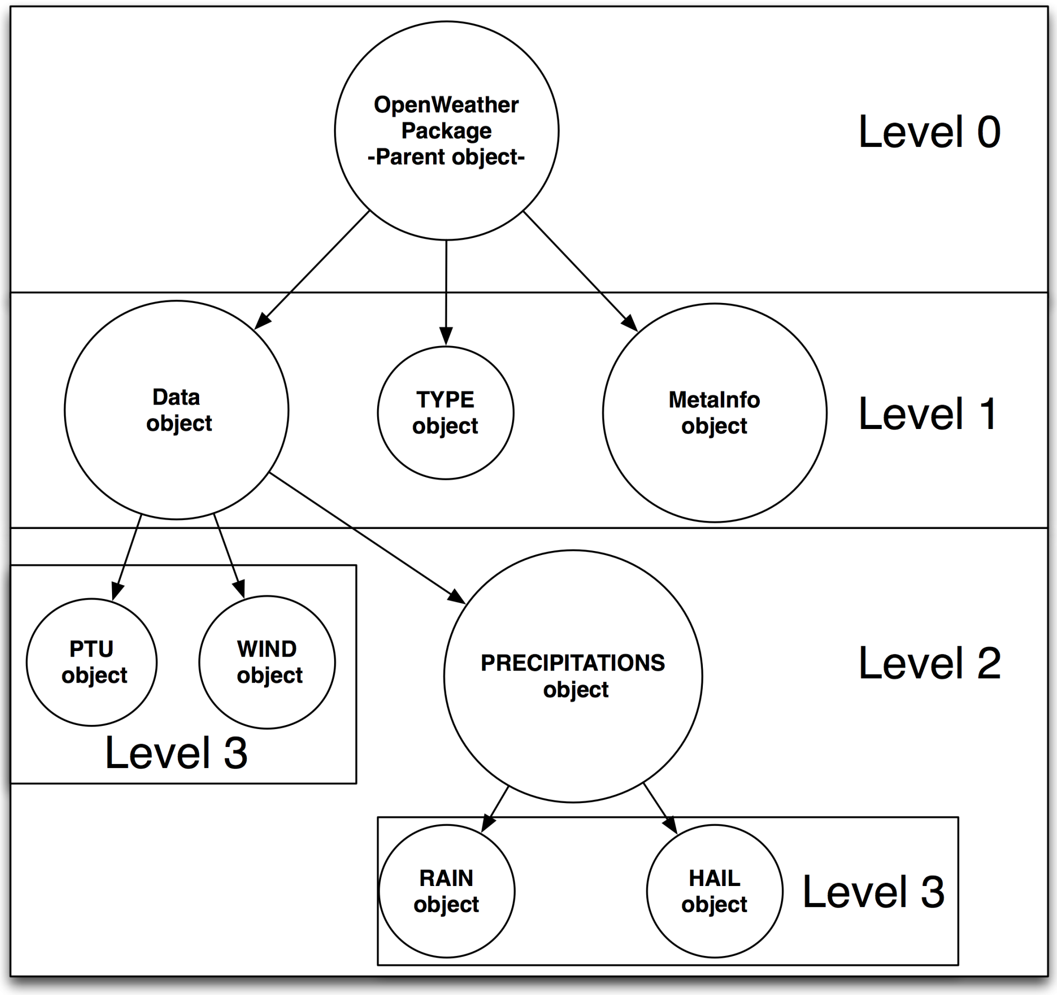


Figure 6.7: OpenWeather's MetaInfo data field with the data array elements.

[27]This data can be stored in the AWS itself.

[28]Mark with a different timestamp inside of anAWS or datalogger.

Through the different levels established in the object's hierarchy, it is easy to find the information that the user expects.

As explained in section 6.4.2, OpenWeather uses numerical codes to identify the types of data messages. In this case the data on demand must be requested by a user (node), thus the protocol's code will be **201**[29].

The data message will contain a JSON object containing an array of data elements. The data field is named "Retrieve", it contains the data requested, indicated by the letter 'D' as a variable to be reference for data objects requested (PTU, wind or precipitation). In addition a timestamp[30] is added to the request in order to specify in which sample is interested the user.[31]

```
{
"OpenWeatherMessage": {
        "Type" : 201,
        "MetaInfo" : {
            ARRAY DATA ELEMENTS
        },
        "Data" : {
            "Retrive" : {
                ["D":"PTU","D":"WIND","D":"PRECIPITATION"],
                "Timestamp": "2011-05-29T12:10:23Z"
            }
        }
    }
}
```

Table 6.28: Real-time data message of OpenWeather protocol.

This request will return the PTU, wind and precipitations recorded in the timestamp provided. The next data message received by the node in response of this will have exactly the same format as a real-time data message, **except the code and the timestamp in the header**; they will provide referencing to the response for the data on demand in the date and time specified.

6.4.6 Internal protocol data

As any other P2P architecture, OpenWeather needs a certain amount of internal data to keep working. Commonly, this data is focused in peer's information

[29]Review the protocol codes reference.

[30]This variable follows exactly the same standards used in the Timestamp field used in the MetaInfo data field.

[31]It is possible to change this field value in order to adapt it to request samples from a range of time.

as hostnames and ports used by the nodes. OpenWeather uses a mechanism to exchange list of peers between nodes, to guarantee the well-functioning of OpenWeather network. The information provided in these data messages can have different purposes. The author reserves this type of data for future implementations, nevertheless the protocol has been implemented to be able to transfer list of peers and keep updated the nodes with them.

The data messages used for this purpose are categorized as protocol dedicated, as explained in section 6.4.2 these data messages can be requests, retrievals or status information.

Opposed to weather data messages, the internal data messages do not have a Data object, but instead are composed by an Info object. This info object contains the data fields referencing the information required by the protocol.

The type of data message —code **101**—, notifies to the node that it must return a list of peers. Because this message also contains the MetaInfo object, the receiver is inform of all the information necessary to deliver the best peer list to the node in the same requests.

In the case of a list of peers, the Info object will contain a list of variables composed by an array of data elements with the IP address of the nodes, the port and the bandwidth available in it:

```
{
"OpenWeatherMessage": {
        "Type" : 101,
        "MetaInfo" : {
            ARRAY DATA ELEMENTS
        },
        "Info": {
"Peer-ID" : ["Peer-IP":"226.134.231.73","Port": "62535","Bandwidth":"2"],
"Peer-ID" : ["Peer-IP":"116.234.231.13","Port": "62535","Bandwidth":"1"],
"Peer-ID" : ["Peer-IP":"186.214.211.53","Port": "62535","Bandwidth":"5"],
"Peer-ID" : ["Peer-IP":"182.124.221.23","Port": "62535","Bandwidth":"6"],
"Peer-ID" : ["Peer-IP":"190.144.231.13","Port": "62535","Bandwidth":"1"]
        }
}
```

Table 6.29: Peer's list exchange in OpenWeather protocol.

The table 6.29 shows the response of the data message, providing a list of peers. Note that the "Peer-ID" will contain the unique ID of the peers. After the requester gets this data message, the software should update the internal list of peers with the new data and to deliver a status data message to the node that provides the list of peers to confirm the correct retrieval of the data.

Services availability

OpenWeather offers a mechanism to know which services are available in an AWS. A node requesting data from these services, must send a data message with code **102**, to obtain a response with the services remotely available in the node.

```
{
"OpenWeatherMessage": {
        "Type" : 102,
        "MetaInfo" : {
            ARRAY DATA ELEMENTS
        }
    }
}
```

Table 6.30: Services list availability request.

After this data message is received by the remote node, it will reply with another data message, providing the list of the services:

```
{
"OpenWeatherMessage": {
        "Type" : 101,
        "MetaInfo" : {
            ARRAY DATA ELEMENTS
        },
        "Info": {
            "Services" : { "PTU":"RO","WIND": "RO","PRECIPITATION":"RO"}
        }
}
```

Table 6.31: Peer's list exchange in OpenWeather protocol.

One array of data is delivered in the reply:

- Services array: indicating the type of service and its availability.[32]

With this information the software knows which services can be checked on the remote node and which kind of data —real-time or on demand— can be retrieved from them.

[32]R is equal to "real time data" and O to "data on demand". Both can be present or isolated.

6.5 Protocol considerations

The following sections describe some aspects of OpenWeather related with other protocols or future features of it.

OpenWeather and other protocols

We can find dozens of protocols available, using P2P architectures and/or optimizations in the data delivery. Nevertheless, the author could not find any protocol suitable enough to fit in the characteristic required by the AWSes. Protocols as Bittorrent[10], have a proven track delivering large amount of data and scaling their networks properly. FastTrack[42] has been successful achieving similar results as Bitorrent. However, almost all the P2P protocols are oriented to transfer files or real-time data with a big size (such as video or voice streams). In addition, these protocols are designed focusing in nodes with common computational capabilities (such as desktops or small servers), not considering embedded system inside of their purpose (being difficult to handle the necessary resources to implement these protocols on an embedded system).

Other alternatives as HTTP, were considered by the author as solutions for this thesis. Nevertheless, HTTP still has a big dependency of the centralized model. At the same time, HTTP works under synchronously mode, something that will limit the real-time capabilities needed for the AWSes.

Finally, because the use of FTP (a generic protocol) is under use for weather instruments, the author considered much more interesting to research a custom solution for the AWSes.

Nevertheless, several concepts have been taken from the mentioned protocols. OpenWeather uses the same philosophy as HTTP, providing in the header of the data message all the information needed. The same approach as HTTP has been chosen to identify the type of data messages. Through protocol codes the data message is identified in a category / purpose, being simple to extend the amount of protocol operations, just creating new identifiers through the codes. Moreover, the protocol uses JSON as data format, being text-based as HTTP. Concepts such node ID, peers-requested or update-interval have been taken from protocols as Bitorrent[10] or FastTrack[42] . These properties allow OpenWeather to implement methodologies tested in other P2P networks with successful results.

Aggregation of data between nodes

As in other P2P networks, the scalability of the OpenWeather network can be an issue. Although OpenWeather does not implement an aggregation technique between the nodes, it is ready to be adapted to it. The nodes conforming to OpenWeather protocol could require the capability to request and retrieve data using indirect paths to the end node. These paths could be found using the connections already established with other nodes.

The aggregation of the data will be executed using the same data format as common on OpenWeather protocol, thus, the data messages will use JSON format plus the required fields in the data message to provide such functionality. The same operations of the protocol will be available through aggregation. In addition, the protocol will require the implementation of new operations for internal use.

We need to consider the nature of the weather data networks when we chose the aggregation technique. As it is described in section 3.1.2, the amount of bandwidth is commonly limited in an AWS. Several techniques have been developed to aggregate information from different sources having in consideration connectivity and bandwidth availability issues. These techniques are classified based in how they aggregate and route the data [35].

In case that the aggregation is required in OpenWeather, it should be a combination of gossiping and tree-based methods, in order to provide a feasible way to aggregate data between nodes. The reason for this combination is that both methodologies have one specified purpose. Gossiping techniques are focused into offer robust communications, meanwhile, tree-based techniques are focused in to have better performance transferring data. Because a weather network needs to guarantee the flow of the data and at the same time the availability of the data as soon as possible, a research combining both techniques must be performed in order to find suitable solutions for such environment.

Notwithstanding, the OpenWeather specification available in this thesis provides the capability to request and retrieve the list of peers of a remote node. The combination of this list of peers and the Keep-Alive value of them, can be used to build a tree-based structure with the nodes that have a established session. Through this tree, OpenWeather can be able to find new paths to other nodes. This will require the implementation of a internal operation of the protocol, providing the capability to make queries to other nodes, in order to build new paths. In addition, the tree-based structure will not be enough to guarantee the robustness necessary for the weather data transmission. Hence, it will be required to find the compatibility of this technique with gossiping methodologies, implementing an algorithm inside the protocol that periodically and randomly tries to update the table of nodes available, and the paths of them.

Finally, we need to highlight that the aggregation of data is a complicated area, not being possible to treat it in this thesis.

Compatibility with centralized models

In chapters two and three we introduced the different techniques and topologies used by weather organizations to acquire weather data. The centralized model was explained, showing how the nodes have a strong dependency of one common point. This setup is the current solution chosen by weather organizations, and almost all big weather data networks are builded based on such infrastructure. 3 Thus, it is needed to consider the compatibility of OpenWeather with this topology. Although OpenWeather is designed to have the AWSes as independent nodes, infrastructures using the centralized model, can provide a node doing a bridge between the collection point and OpenWeather network. It will be required to develop the methods to retrieve the data from the subnet of the collection point. As it was mentioned in section 5.1.2 and the example of HTTP and OpenWeather, it is possible to encapsulate data to other protocols with the proper adaption.

Because every weather organization has their own setups and methodologies, an independent study will be required in order to design a bridge from the collection point models to the P2P architecture of OpenWeather.

6.6 Summary

In this chapter the core architecture of OpenWeather was presented. The definitions establish by the protocol have been explained. The roles of nodes and their identification is presented to justify how OpenWeather can be adapted for future use in a different system for AWS identification.

We have explained the architecture of OpenWeather. Justifying the use of the AWSes as indivudal nodes conforming a P2P network. The protocol functionality is analyzed, explaining how the different operations perform. The main characteristics of OpenWeather have been exposed.

The structure used in OpenWeather data messages has been analyzed, explaining how JSON is used as syntax to encapsulate the data. In addition, the application of object hierarchy on data has been explained. All data fields, which compose data messages were defined technically.

The protocol codes and their categories have been described, justifying their numeration and purposes.

The differences between real-time data messages, data messages on demand and internal data messages, have been justified, putting attention in how the different data messages have a common structure and use. Finally an example of all the types of data messages implemented in the protocol are explained, providing enough information to implement a functional prototype of it.

Chapter 7

Experimental evaluation setup

In this chapter, the experimental setup used to implemented the proof of concept of the OpenWeather protocol is explained. A generic AWS has been setup to test the protocol with real-time data. The software architecture implementing the functionality of the protocol is introduced as well. The purpose of this chapter is to introduce the general guidelines followed by the implementation of a prototype of OpenWeather protocol and to analyze the tests cases performed using it.

7.1 Scenario

The AWS utilized in the experimental setup is the model WXT520, manufactured by Vaisala Oyj. Along with other AWSes sharing these characteristics, it is able to measure the following phenomena:

- Liquid Precipitation
  - Hail
  - Rain
- Relative Humidity
- Wind
  - Direction
  - Speed
- Air Temperature
- Barometric Pressure

The geographical location of the AWS is **N 60º 11' 15.6" E 024º 50' 14.8"**[1]. The AWS has been connected to a computer, in which the software developed to implement OpenWeather protocol is installed. The AWS has been configured following the manufacturer suggestions, emulating a normal installation environment. The digital interface configured in the AWS is a RS-232 port, offering a maximum amount of bit rate of 116 kbit/s.

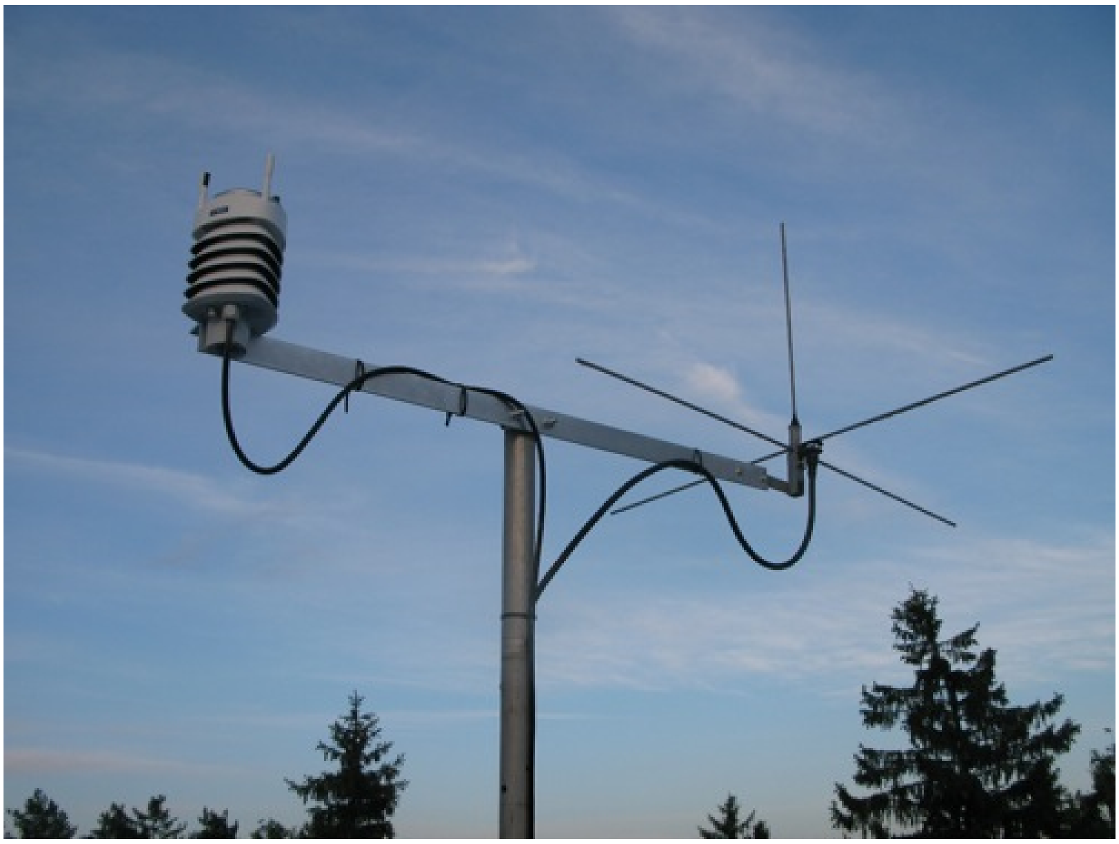


Figure 7.1: AWS installed to simulate a real scenario.

The AWS is plugged in continually 24 hours and installed on a mast of 2 meters length. The RS-232 port provides the data acquired in the AWS to computer a that operates an implementation of OpenWeather protocol.

Thus, the AWS used to implement the protocol has not been modified to adapt it to OpenWeather, all the adaptions realized have been made through a software implementation. This fact allows the verification of the adaptability of the protocol to the current technology without no major modifications to the AWS.

[1]UTM: Zone: 35 Easting: 380076 Northing: 6674276. Municipality of Otaniemi, Espoo, Finland.

Evaluation setup

The evaluation setup consists four nodes. All of them run a copy of the prototype, thus acting as nodes. Nevertheless, only one node is connected to a functional AWS, the other three simulate the weather data input[2].

The table 7.1 shows nodes specifications:

CPU	Memory	Network connection	Operating system	Hostname
2.4GHz	4GB	100Mbps	GNU/Linux	Node 1
2.2GHz	1GB	100Mbps	GNU/Linux	Node 2
900GHz	1GB	128Kbps	GNU/Linux	Node 3
1GHz	1GB	56Kbps	GNU/Linux	Node 4

Table 7.1: Hardware and OS specifications of the evaluation setup.

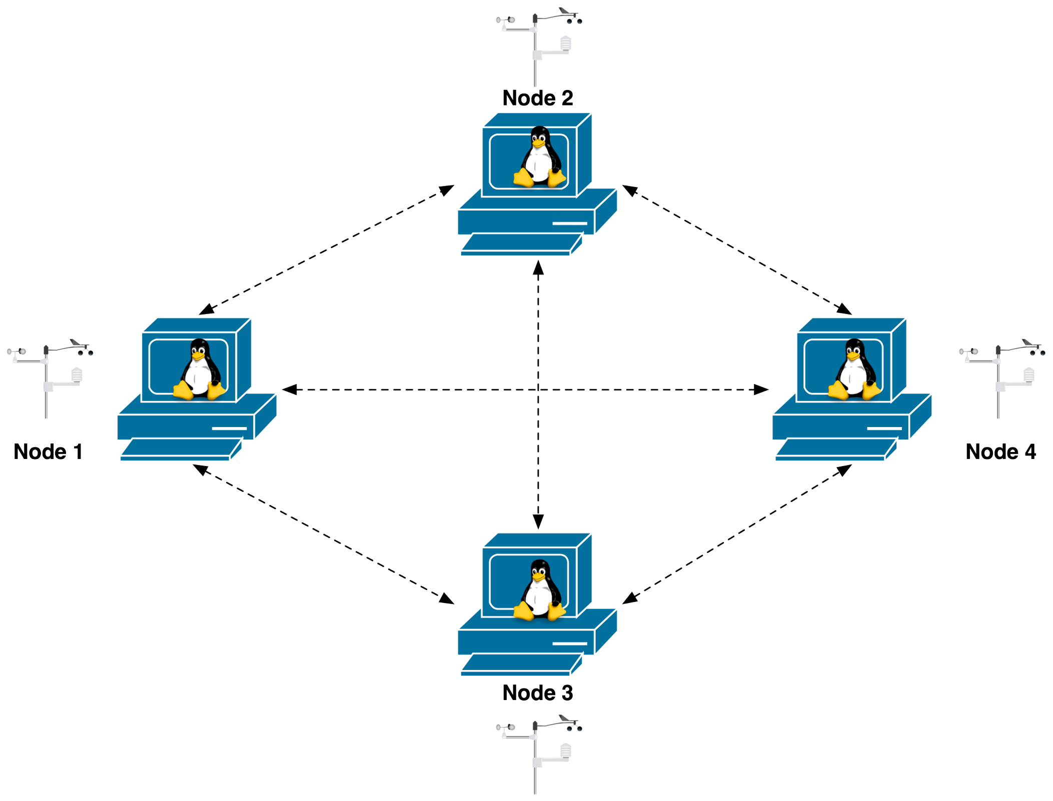


Figure 7.2: Network topology used in the evaluation setup.

All the nodes posses network visibility among them, with maximum network latency less than **75 milliseconds**. The bandwidth in node number two and four

[2]Generated randomly based on the same patterns as a normal AWS.

has been limited (Round-trip time (RTT)) to 128kbit/s and 56kbit/s respectively. These restrictions emulate the network limitations mentioned in chapter three.

The purpose of this setup is to create an environment that simulates common conditions experience during weather data acquisition. All the nodes use OpenWeather protocol to exchange data between them. This environment provides the necessary resources to test and verify the characteristics of the protocol, such data message size, times of response, etc.

7.2 Prototype implementation

In order to verify the feasibility and the functionality of OpenWeather, the author developed a proof of concept of the protocol, to test and verify its feasibility as alternative protocol for weather data transmission. This implementation provides the necessary data to independently evaluate the protocol.

7.2.1 Technologies used

OpenWeather is designed to have an emphasis on the data structures used in the software implementation. In addition, the object hierarchy used to structure the data makes the implementation of the protocol easier by using an object oriented programming language.

Thus, C++ has been chosen as the primary language used in the prototype. The C++ standard library is used to write the intermediary layer (in combination with some Python scripts). Because the target of this protocol can have end users which are not familiar with command-line applications, a functional GUI has been implemented. The Qt framework[33] has been chosen to implement the GUI, together with QJson[9] for the data representation.

7.2.2 Software Architecture

The prototype requires to be implemented supporting the functionality described in the P2P architecture taxonomies[8]. Thus, the nodes should posses the capability to request services, and at the same time, offer services to others nodes. This requirement conditions the node to behave as a client and server at the same time.

To realize this architecture, the concept of **local peer** is introduced. The local peer refers to the node itself; representing the AWS entity in the network; nevertheless, as described in section 5.2.1, a node without an AWS can be part of the network as well.

Because the node behaves as client and server at the same time, the software implementation is designed to maximize the utilization of the common resources between both modalities. Thus, the implementation of the classes have been done using abstract interfaces, not mattering if the data to process has been received through the client or server module.

Common implementation

The prototype implementing OpenWeather protocol has been optimized for the data structures and object hierarchy explained in chapter six. The handling of JSON data through TCP sockets is the basis of the implementation. The prototype focuses its core functionality in to take advance of the most optimal way of sockets management and data manipulation.

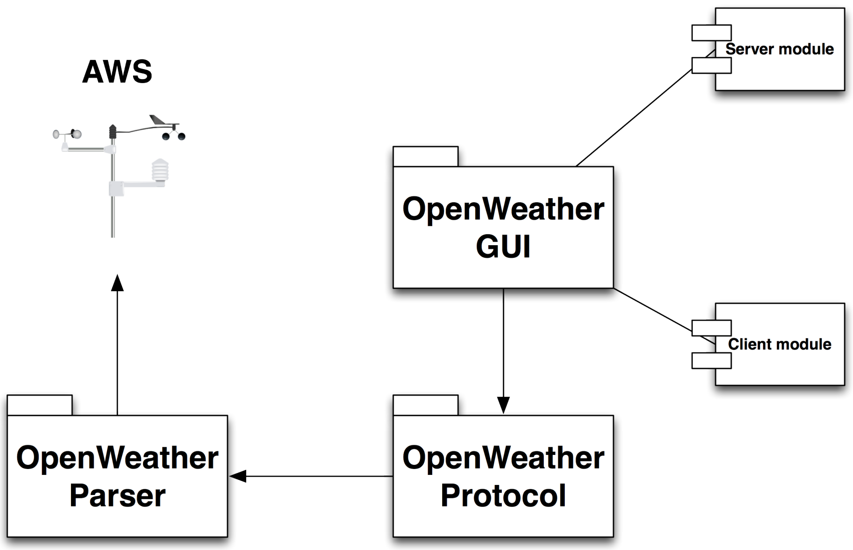


Figure 7.3: Software prototype conceived.

The prototype is structured in three parts:

- The GUI providing access to certain functionalities of OpenWeather protocol.
- The network level implementation of OpenWeather protocol.
- The intermediary middle layer adapting the WXT520 to OpenWeather protocol.

Despite this modularity in the components, everything is assembled in one application. The prototype implements the client and the server modules internally. Both modules have access to the core implementation of OpenWeather protocol, and at the same time the application is linked with the OpenWeather parser (*libopenweatherparser*).

The implementation of the protocol has been made based on the objects hierarchy explained in chapter six. Thus, the representation of the OpenWeather data only involves the transformation of JSON objects into primitive data types.

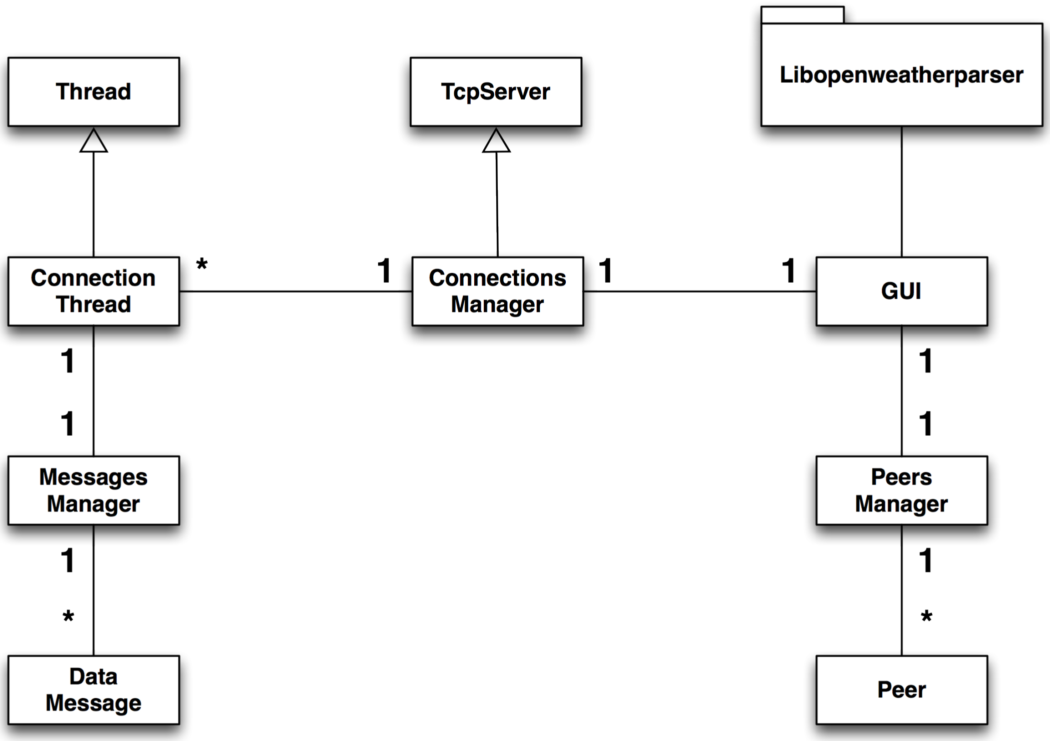


Figure 7.4: UML diagram of the prototype.

The figure 7.4 shows a general overview of the classes implemented in the prototype in order to make functional the OpenWeather protocol. All the classes developed are able to manage the data in both modes (client and server), being possible to retrieve and delivery data using the same internal software mechanisms, with complete transparency for the end user.

Client module

The software implements certain parts fully pertaining to client operations. Client operations are identified those that involve the data request to other nodes. When the software is using OpenWeather to retrieve data from other nodes, we denominate that it is working under client mode.

The client module of the software allows the following operations:

- Request session establishment - peer handshake
- Request real-time and/or data on demand
- Request the service availability in remote node/s

Server module

As requirement of the RFC 5694[8], an application implementing a P2P architecture must be able to offer services. To achieve this, the prototype implements one part that provides the server functionality. A socket listening to the TCP port used in OpenWeather is created when the prototype software is executed. Thus, the software allows other peers to connect to it, providing exactly the same features that client mode is able to request. Because the OpenWeather protocol is designed to not distinguish between the nodes and the services that they offer, the implementation of the server module is nearly identical to the client mode.

The server module o allows the following operations:

- Session establishment
- Delivery of real-time and/or on data on demand
- Delivery of the service available on the local node

GUI

The graphical interface aims to provide the possibility to use the protocol[3]. The GUI allows fully utilization of the AWS data interface to check the data received, to connect to OpenWeather, and to perform the operations described in the chapters six (connect to other peers, delivery real-time data samples or retrieve the services available in the remote peers).

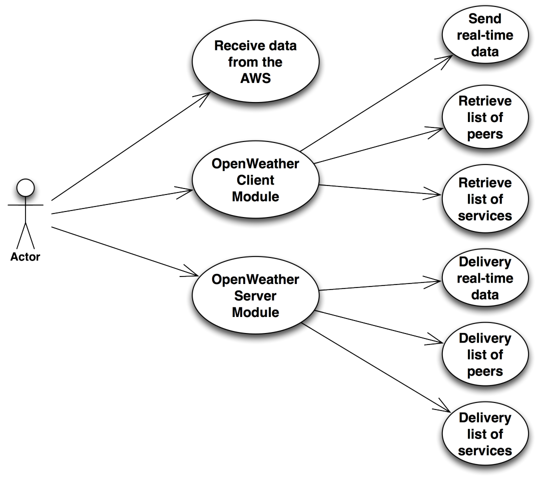

Actor
Receive data from the AWS
OpenWeather Client Module
OpenWeather Server Module
Send real-time data
Retrieve list of peers
Retrieve list of services
Delivery real-time data
Delivery list of peers
Delivery list of services

Figure 7.5: Prototype use case diagram.

The GUI has single instances of the **ConnectionsManager** and the **Messages-Manager** classes. Both classes provide the functionality required to handle peers

[3]A set of screenshots took from the GUI is available in the appendix.

and connections. In addition, the library ***libopenweatherparser***, provides the middle layer explained in section 5.2. This library is linked to the AWS, providing the RAW data collected from its digital interface, and converting it from the vendor's format to OpenWeather's format.

Connections manager

The ConnectionsManager class is in charge of handling the sockets, managing all the connections of the node. In addition, this class controls the socket used to allow remote nodes to connect to the local peer (server module).

All sockets are handled using threads, thus, all the connections are managed independently in a secondary plane, not blocking the GUI or not interfering with other connections. This implementation allows the prototype to manage multiple connections with multiple peers without performance issues.

Peers manager

The PeersManager class is in charge of the peers. The purpose of this class is to provide a control system of the peers that the node can connect to and their information; at the same time this class manages the local peer and the services that it offers to the remote peers.

This class gets updated information when a the data messages received contain data related with the peers (protocol internal traffic), for instance if some peer updates its metainformation or just confirms the receival of message.

Messages manager

The MessagesManager class handles the OpenWeather data messages. This class is able to generate data messages based on the specifications of OpenWeather protocol. Every connection containing a data message is able to access it. This class provides the core functionality of the protocol, being able to understand the protocol codes and based on them, executing the operations needed in order to achieve the expected result.

All data messages are assembled and disassembled in this class, because as OpenWeather requires JSON as its primary data format, this class provides mechanisms to generate and validate the data format of the messages.

Libopenweatherparser

This library has been developed in order to create a bridge between the AWS and the prototype. The data format used by the vendor in the AWS has been implemented in the library, creating the functionality to convert the vendor's format to the OpenWeather data format. This library is thought to normalize the data from one to multiple vendors, offering primitive data types ready to be assembled in JSON objects as output.

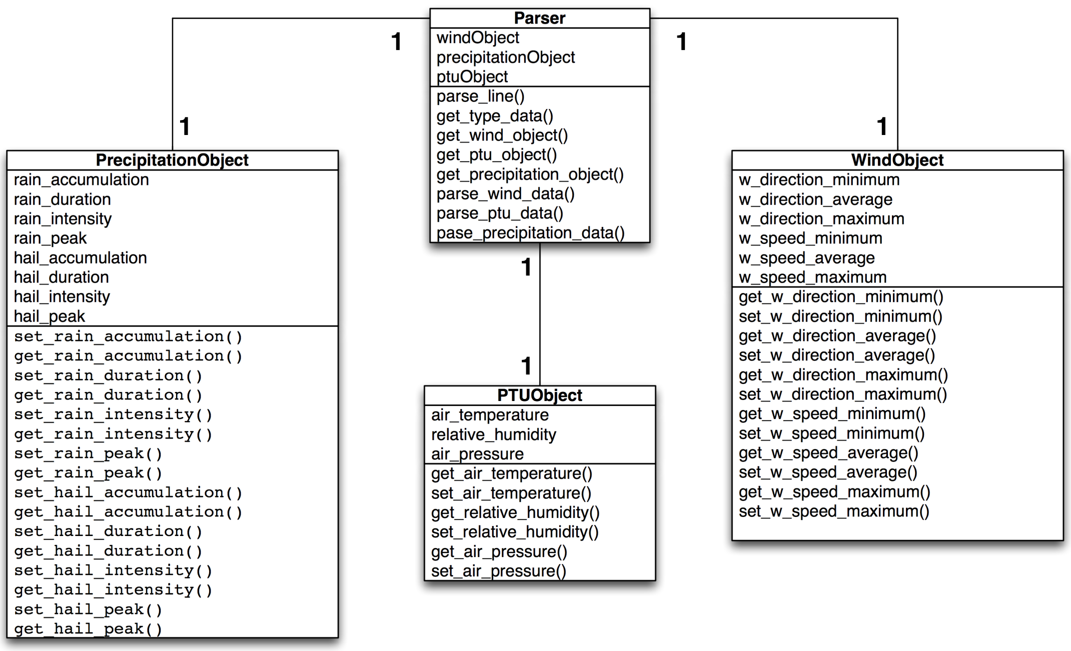


Figure 7.6: UML diagram of the library.

The library acts as an intermediary layer. Should the vendors choose to implement the OpenWeather format, the requirement of this library will be dropped. However, since the source code the AWS operating system is not available, it is not currently possible to implement the OpenWeather protocol integrated with the vendors software without their cooperation.

7.3 Testing

The prototype provides the capability to perform the operations described on chapter six. The main goal of the testing is to analyze if the implementation of the protocol achieves its purpose and the results that its generates.

The scenario used for the testing is described in the previous section. The following sections explain the utilization of different nodes used to transmit weather data using the OpenWeather protocol.

The methodologies followed to evaluate the behavior of the protocol are based in the analysis of the network traffic between nodes and the verification of the protocol operations. The tool used to capture the data messages is Wireshark[11]. This tool provides enough information to verify the operations of the protocol in the network layer.

The following protocol operations have been implemented in the prototype:

- Session establishment - peer handshake
- Service discovery
- Real-time data retrieval

Implementation considerations

Although the chapter six specifies more operations, as data on demand or peer list exchange, they have not be implemented due to their similarity in the architecture and data messages size, with the test cases executed.

The Keep-Alive functionality has not being implemented in the prototype, because this feature is just an extra check performed for OpenWeather to double assure the connectivity and the response of the node in the application layer, and it does not influence the functionality of the prototype.

All nodes have been synchronized according with date-time through NTP [30] with the **ntp1.funet.fi** server before to execute any operation. This synchronization has been performed in order to guarantee the accuracy of the measurements. Nevertheless, as it is explained in the Timestamp section, it is highly recommended to sync the clock of the nodes to guarantee the quality of the weather data.

The RAW ASCII representation of the data messages appears in different order compare to the specifications. This is due to the software re-orders the data elements by alphabetical order (always inside of the objects hierarchy).

All the data messages are keeping similar space constrains between the data elements inside the JSON object. This is causing a known additional increase of the data message size, this size can be reduced even more, suppressing theses spaces. In addition, the migration to a binary representation of the data messages using Binary-JSON (BSON)[37] should be straightforward[4].

The execution of the test has been done 50 times, extracting the average from it. The times of the sequence are including the execution of the software operations in both sides.

7.3.1 Test 1: Handshake between nodes

The purpose of this test is to validate the operation described in section 6.3.1 —**Session establishment & Peer handshake**—. In this test the peers involved

[4]Though will cause conflicts with the endianess.

are exchanging information about themselves, in order to establish the session.

The *Node 1* will send a handshake data message to the *Node 2*. This data message contains all the MetaInfo data field filled with the data of the *Node 1*, the protocol code used is **100**.

Sequence

The scenario assumes that the *Node 1* knows the IP address, TCP port and ID of the *Node 2*, because it was obtained from some list of peers received from other nodes.

The following sequence happens in the network layer:

1. *Node 1* sends a data message containing all its metainformation to the remote *Node 2*, connected through the port specified and requests session establishment.
2. *Node 2* receives a data message delivery by *Node 1*, containing all its metainformation and requesting the session establishment.
3. *Node 2* sends a data message to *Node 1*, providing all its meta-information and confirming the session establishment with the protocol code **101**.
4. The session is established between both nodes.

The following sequence happens in the software layer:

1. The button session-establishment generates the connection sequence to the node chosen (*Node 2*).
2. A thread is created, establishing a TCP connection to the chosen node. The messages manager assembles data message with all metainformation of the local node and with the protocol code **100**.
3. The data message is delivery through the socket managed by the thread.
4. The messages manager in the *Node 2* receives a data message and creates a thread to handle it.
5. The messages manager called by the thread, parses the data message and identifies its protocol code.
6. A response is generated based on the protocol code of the data message, and is deliver to *Node 2*.
7. Because the operation is the session establishment the peers manager gets executed in both sides, updating the peer information (if needed) of the peers.

Analysis

- The data session captured with Wireshark involves **7** TCP segments.
- The data message (OpenWeather) generated by ***Node 1***, has a size of **375** bytes.
- The data message (OpenWeather) generated by ***Node 2***, has a size of **375** bytes.
- The total size of the OpenWeather data message is **750** bytes.
- The total size of the sequence (TCP/IP and OpenWeather) is **1227** bytes.

The RAW ASCII representation of the data message data capture is shown in table 7.2.

Node 1
`{ "OpenWeatherMessage" : { "MetaInfo" : { "Bandwidth" : 6, "ID" : "33c11957579d1093e931bd540536b40e90339dbded8e2a2ce4e 64c480c8132bc", "Keep-Alive" : 120000, "Location" : "6672224 385565 35V", "Peer-IP" : "172.21.25.16", "Peers-Requested" : 20, "Port" : 62535, "Timestamp" : "2011-07-20T16:51:29", "Update-Interval" : 120000, "Version" : "OpenWeather/1.0" }, "Type" : 100 } }`
Node 2
`{ "OpenWeatherMessage" : { "MetaInfo" : { "Bandwidth" : 6, "ID" : "11f1cb9fb5bc57cf7905dc26c3ef045ae7b54d5ff1c7e233ff2d31be 4977bd18", "Keep-Alive" : 120000, "Location" : "6672224 385565 35V", "Peer-IP" : "172.21.25.20", "Peers-Requested" : 20, "Port" : 62535, "Timestamp" : "2011-07-20T16:51:29", "Update-Interval" : 120000, "Version" : "OpenWeather/1.0" }, "Type" : 101 } }`

Table 7.2: Data messages transmitted between *Node 1* and *Node 2*.

The TCP flow between both nodes using OpenWeather is the following:

```
| 172.21.25.16          172.21.25.20    |
|        SYN       |                    |Seq = 0 Ack = 1303623571
|(39239)  ------------------->  (62535)   |
|        SYN, ACK  |                    |Seq = 0 Ack = 1
|(39239)  <-------------------  (62535)   |
|        ACK       |                    |Seq = 1 Ack = 1
|(39239)  ------------------->  (62535)   |
|        PSH, ACK - Len: 375            |Seq = 1 Ack = 1
|(39239)  ------------------->  (62535)   |
|        ACK       |                    |Seq = 1 Ack = 376
|(39239)  <-------------------  (62535)   |
|        PSH, ACK - Len: 375            |Seq = 1 Ack = 376
|(39239)  <-------------------  (62535)   |
|        ACK       |                    |Seq = 376 Ack = 376
|(39239)  ------------------->  (62535)   |
```

Table 7.3: TCP flow sequence between *Node 1* and *Node 2*.

The time of execution of this TCP sequence is **65 milliseconds** on average.

Both nodes have delivered the data successfully, achieving the session establishment as result of the sequence.

Discussion

The measurements show that OpenWeather requires a small amount of data for the session establishment. In addition, a low response time is needed to complete the operation. It achieves the goal to provide a mechanism to establish session even with really low bandwidth availability. This small size of data can be easily handled by the memory and processor of an AWS. As the protocol specification requires, the session establishment provides all the necessary information to both nodes, to proceed requesting other data, after the peer registration happens in the software side.

7.3.2 Test 2: Service discovery

The purpose of this test is to validate the operation described in section **??** —**Service discovery**—. In this test the peers involved are exchanging information about service availability, in order to know which services could be requested.

The *Node 3* will send a service discovery data message to *Node 4*. This data message contains all the MetaInfo data field filled with the data of the *Node 3*, in addition the protocol code used is **102**.

Sequence

The scenario assumes that *Node 3* and *Node 4* have established the session, following exactly the same steps than mentioned in section 6.3.1.

The following sequence happens in the network layer:

1. *Node 3* sends a data message containing all its metainformation to the remote host of the *Node 4*, using the session already established between them.
2. *Node 4* receives a data message delivered by *Node 3*, containing all its metainformation and requesting the services available on it.
3. *Node 4* sends a data message to the *Node 3*, providing all its metainformation and delivering a data message with all the services available on it through the protocol code **103**.
4. *Node 3* receives the list of services available in the *Node 4*.

The following sequence happens in the application layer:

1. The button services discovery generates the connection sequence to the node chosen (*Node 4*).
2. The thread previously created by the session, uses the TCP connection established to the chosen node. The messages manager assembles a data message with all the metainformation of the local node and sends through connection with the protocol code **102**.
3. The data message is delivery through the socket managed by the thread.
4. The connections manager in *Node 3* receives a data message and creates a thread to handle it.
5. The messages manager is called by the thread, parses the data messages and identifies its protocol code.
6. A response is generated based on the protocol code of the data message, and is deliver to the *Node 4*.
7. Due to the operation being service discovery, the peers manager gets executed in *Node 4*, checking the services available on it and providing their information into the data message.

Analysis

- The data session captured with Wireshark involves **7** TCP segments.
- The data message (OpenWeather) generated by the ***Node 3***, has a size of **375** bytes.

- The data message (OpenWeather) generated by the ***Node 4***, has a size of **458** bytes.
- The total size of the OpenWeather data message is **833** bytes.
- The total size of the sequence (TCP/IP and OpenWeather) is **1310** bytes.

The RAW ASCII representation of the data message captured is shown table 7.4

Node 3
`{ "OpenWeatherMessage" : { "MetaInfo" : { "Bandwidth" : 1, "ID" : "654b7b521acc7549bf6854b1113d44e6433bf94a1b4caf4327e33 e9bc89b4025", "Keep-Alive" : 120000, "Location" : "6672224 385 565 35V", "Peer-IP" : "172.21.25.35", "Peers-Requested" : 20, "Port" : 62535, "Timestamp" : "2011-07-24T12:04:09", "Update-Interval" : 120000, "Version" : "OpenWeather/1.0" }, "Type" : 102 } }`
Node 4
`{ "OpenWeatherMessage" : { "Info" : { "Services" : { "PRECIPITATION" : "RO", "PTU" : "RO", "WIND" : "RO" } }, "MetaInfo" : { "Bandwidth" : 0, "ID" : "3b1f665e0d622aab7b2e71b29d966dd2a22c5d427f337585 09d4205720de9d2e", "Keep-Alive" : 120000, "Location" : "6672224 385565 35V", "Peer-IP" : "172.21.25.40", "Peers-Requested" : 20, "Port" : 62535, "Timestamp" : "2011-07-24T12:04:09", "Update-Interval" : 120000, "Version" : "OpenWeather/1.0" }, "Type" : 103 } }`

Table 7.4: Data messages transmitted between *Node 3* and *Node 4*.

The TCP flow between both nodes using OpenWeather is the following:

```
| 172.21.25.35              172.21.25.40 |
|        SYN       |                     |Seq = 0 Ack = 2259331907
|(50550)  ------------------->  (62535)   |
|        SYN, ACK  |                     |Seq = 0 Ack = 1
|(50550)  <-------------------  (62535)   |
|        ACK       |                     |Seq = 1 Ack = 1
|(50550)  ------------------->  (62535)   |
|        PSH, ACK - Len: 375             |Seq = 1 Ack = 1
|(50550)  ------------------->  (62535)   |
|        ACK       |                     |Seq = 1 Ack = 376
|(50550)  <-------------------  (62535)   |
|        PSH, ACK - Len: 458             |Seq = 1 Ack = 376
|(50550)  <-------------------  (62535)   |
|        ACK       |                     |Seq = 376 Ack = 459
|(50550)  ------------------->  (62535)   |
```

Table 7.5: TCP flow sequence between *Node 3* and *Node 4*.

The time of execution of this TCP sequence is **84 milliseconds** on average.

Both nodes have delivered the data successfully, achieving the service discovery as the result of the sequence.

Discussion

The service discover operation has bigger data message size than the session establishment. Nevertheless, this operation considered fairly small in size, and it fitting to the environment with low bandwidth available. As the session establishment, the service discovery is a common operation inside of the protocol. Its fast delivery is critical, thus, in order to provide the services available as soon as possible to the requester.

7.3.3 Test 3: Real-time data retrieval

The purpose of this test is to validate the operation described in section 7.3.3 —**Real-time data retrieval**—. In this test the peers involved are exchanging real-time weather data.

The *Node 4* will send a real-time data request to *Node 1*. This data message contains all the MetaInfo data field filled with the data of *Node 4*, in addition the protocol code used is**200**.

Sequence

The scenario assumes that *Node 4* and *Node 1*, have established the session, following exactly the same steps mentioned in section 6.3.1.

The following sequence happens in the network layer:

1. *Node 4* sends a data message containing all its meta-information to the remote host of the *Node 1*, using the session already established between them.
2. *Node 1* receives a data message delivered by *Node 4*, containing all its metainformation and requesting real-time data.
3. *Node 1* sends a data message to *Node 4*, providing all its metainformation and delivering a data message with the latest weather data available on its AWS, assigning the protocol code **201**.
4. *Node 4* receives the latest real-time data sample available in *Node 1.*

The following sequence happens in the application layer:

1. The button assign to request real-time data, generates the connection sequence to the node chosen (*Node 1*).
2. The thread previously created by the session, uses the TCP connection established to the chosen node. The messages manager assembles a data message with all the metainformation of the local node and assigning the protocol code **200**.
3. The data message is delivered through the socket managed by the thread.
4. The connections manager in *Node 1* receives a data message and creates a thread to handle it.
5. The messages manager called by the thread, parses the data messages and identifies its protocol code.
6. A response is generated based on the protocol code of the data message. Since this response involves real-time weather data, a call is made to the **libopenweatherparser**, to obtain the latest real-time data available in the AWS. After that, the data is deliver to *Node 4*.

Analysis

- The data session captured with Wireshark involves **7** TCP segments.
- The data messages (OpenWeather) generated by ***Node 4***, has a size of **375** bytes.

- The data messages (OpenWeather) generated by ***Node 1***, has a size of **814** bytes.
- The total size of the OpenWeather data messages is **1189** bytes.
- The total size of the sequence (TCP/IP and OpenWeather) is **1666** bytes.

The RAW ASCII representation of the data messages is shown in table 7.6.

Node 4

```
{ "OpenWeatherMessage" : { "MetaInfo" : { "Bandwidth" : 0,
"ID" : "3b1f665e0d622aab7b2e71b29d966dd2a22c5d427
f33758509d4205720de9d2e", "Keep-Alive" : 120000, "
Location" : "6672224 385565 35V", "Peer-IP" : "172.21.
25.40", "Peers-Requested" : 20, "Port" : 62535, "Timest
amp" : "2011-07-25T14:15:35","Update-Interval" :
120000, "Version" : "OpenWeather/1.0" },"Type" : 200 } }
```

Node 1

```
{ "OpenWeatherMessage" : { "Data" : { "PRECIPITATION" : {
 "Hail" : { "accumulation" : "0", "duration" : "0", "intensity" : "0"
, "peak" : "0" }, "Rain" : { "accumulation" : "0", "duration" : "0",
"intensity" : "0", "peak" : "0" } }, "PTU" : { "Air-Pressure" : "10
14.1", "Air-Temperature" : "19.1", "Relative-Humidity" : "69.4"
}, "WIND" : { "Direction" : { "ave" : "160", "max" : "160", "min"
: "160" }, "Speed" : { "ave" : "1.7", "max" : "1.8", "min" : "1.7"
} } }, "MetaInfo" : { "Bandwidth" : 6, "ID" :"33c11957579d10
93e931bd540536b40e90339dbded8e2a2ce4e64c480c8132
bc", "Keep-Alive" : 120000, "Location" : "6672224 385565 35V
", "Peer-IP" : "172.21.25.16", "Peers-Requested" : 20, "Port"
: 62535, "Timestamp" : "2011-07-25T14:15:35", "Update-Inter
val" : 120000, "Version" : "OpenWeather/1.0" }, "Type" : 300 }
}
```

Table 7.6: Data messages sent between *Node 3* and *Node 4*.

The TCP flow between both nodes using OpenWeather is the following:

```
| 172.21.25.20              172.21.25.40 |
|         SYN       |                    |Seq = 0 Ack = 1015394402
|(49983)  ------------------->  (62535)   |
|         SYN, ACK  |                    |Seq = 0 Ack = 1
|(49983)  <-------------------  (62535)   |
|         ACK       |                    |Seq = 1 Ack = 1
|(49983)  ------------------->  (62535)   |
|         PSH, ACK - Len: 374            |Seq = 1 Ack = 1
|(49983)  ------------------->  (62535)   |
|         ACK       |                    |Seq = 1 Ack = 375
|(49983)  <-------------------  (62535)   |
|         PSH, ACK - Len: 814            |Seq = 1 Ack = 375
|(49983)  <-------------------  (62535)   |
|         ACK       |                    |Seq = 375 Ack = 8153
|(49983)  ------------------->  (62535)   |
```

Table 7.7: TCP flow sequence between *Node 1* and *Node 2*.

The time of execution of this TCP sequence is **96 milliseconds** on average.

Both nodes have delivered the data successfully, achieving the transmission of a real-time weather sample as result of the sequence.

Discussion

Though this real-time data sample does not contain rain or hail data (both are delivered with a 0 value), we can observe how the PTU and the wind data (together with the MetaInfo data field) are up to **1.5** kB. Even with this data size, it will fit in the memory available in an AWS described in section 3.1.2. Assuming that an AWS has between 32-64 kB of volatile memory, and taking half of its memory for internal use of AWS operating system, there is still enough memory to handle real-time data samples using the OpenWeather protocol.

7.4 Summary

In this chapter the scenario and software architecture used to evaluate OpenWeather has been introduced. We tested three different use cases of the protocol with the prototype developed.

In all the use cases executed, the protocol is taking advantage of its properties and achieving a successful result.

Though the prototype implements the partial functionality of the OpenWeather protocol, it shows how the P2P can be implemented in applications oriented to weather transmission. In addition, the small sizes of the data messages and the robustness of the data transmission offered by TCP, provide enough confidence to confirm that the protocol can be implemented in the environments with low bandwidth availability.

Our goal was to verify a feasible implementation of the OpenWeather protocol and verify its functionality with a real scenario. Both purposes have been achieved.

Finally, the use of a real scenario and the integration of the prototype with it, proves how a modern AWS can be adapted to OpenWeather protocol with a few modifications through a software adaption. This fact supports that the current technology can be adapted to new methodologies to transmit the weather data, without a modification in the electronics or industrial design of the AWS.

Chapter 8

Conclusions

In this thesis we exposed the basis of weather observation, how different organizations around the world are collecting and studying enormous amounts data of different phenomena. From the very beginning the industry has been building really complex instruments to measure these phenomena. Many people, from individuals to scientists, are spending their time and resources to part take in the worldwide observation of weather. It is a fact that we need to understand the weather in order to better understand our planet and implicitly, to increase our quality of life.

We have analyzed how the instruments used for such purposes and their limits restrict our knowledge expansion. We described how the industry has been improving these instruments in many different ways. Areas such as the industrial design of the instruments or their internal electronics, have been experiencing tremendous improvements during the last decades, thus allowing the industry to offer weather measure instruments of strong robustness and high accuracy.

Based on the study of these instruments and the scientific discussion of those using them, such the SMEAR project[40], we have come to a conclusion that methods used in them can be improved significantly concerning real-time weather data transmission.

Through the analysis of the different architectures used to collect the weather data, we found several points related to technologies used on network level that need to be changed in order to achieve a successful delivery of real-time data.

We explained how the industry have been introducing new digital interfaces in order to adapt the AWS to the new standards. Nevertheless, although the digital interfaces have been upgraded, the protocols used to transmit the data through them have certain particularities such the use of vendor data specific formats.

In addition, the analysis performed in different instruments and the network technologies that they use, has indicated that the data format and the protocol standards used are of low compatibility with capabilities such real-time data acquisition or data exchange.

The mainstream methodologies currently used to transmit the weather data, such the FTP or the use of CSV as data formats, are limiting the possibility to deliver data with frequency and accuracy high enough to consider it real-time data. Nevertheless, these methodologies are currently considered the state of the art and thought to be sufficient for performing in current architectures used to acquire data.

Though some organizations as NOAA or ICAO, have been creating some data formats for certain purposes (such air navigation or CWOP),nowadays , the global standard still not adapted for the weather industry. The WMO, conscious of this situation, started a process of standardization for weather data representation in 2002. At the moment, this process still under development without any official standard published.

The absence of a standard data format and a protocol to transmit it, is avoiding the possibility to take advantage of all the capabilities that an AWS can offer, more specifically the real-time data acquisition. Although the weather organizations have access to weather data samples updated with small frequencies of time, programs as GOS or GDPFS, are seeking to establish the basis of future systems for weather observation, providing features as real-time capabilities and compatibility between data formats.

All the issues mentioned previously have been considered during the development of OpenWeather. As solution for the problem statement, OpenWeather aims to provide all the features necessary to take advantage of the weather instruments concerning their capabilities to accomplish weather data transmission in real-time.

Based on the architecture used to collect weather data, we use its topology to adapt it to the P2P architecture. Thus, we transform any AWS in a node offering services to other nodes. To achieve such behavior, we developed the OpenWeather protocol from scratch, conceiving it will all the necessary properties to make it P2P and at the same time, adapting its core functionality to the weather data requirements. Being conscious of the absence of standards in such area, OpenWeather has been designed adopting as much standards as possible into its architecture, such the use of standard measurement units or date-time format.

As a result, OpenWeather provides a new way to transmit weather data and to interact with the AWSes.

The implementation of the protocol in a software prototype and its posteriorly use, verify its feasibility in order to translate the protocol specifications to a functional software implementation to be tested in a more complex scenario.

In the experimental setup we verify that OpenWeather —in its implementation as prototype— works in a scenario using the same technologies that are currently common among weather observation experts. The prototype implemented gives us the possibility to communicate with other nodes, executing the protocol operations designed to achieve the weather data transmission. In addition, the P2P functionality of the protocol has been tested, verifying that the AWSes can be treated as independent nodes, requesting and offering services at the same time, and still achieving a successful weather data transmission without a centralized collection point.

We identify as requirement the adaption of the intermediary layer developed to other vendor's data formats, in order to make compatible OpenWeather with different weather instruments from different brands.

Although we described how nodes using the OpenWeather protocol could be able to gather data between them, such functionality has not being implemented in the prototype. Hence, future research should be performed in order to evaluate the capabilities of the protocol to scale in large networks. In addition, the implementation of weather data networks using scalable methodologies, should be study together with their connectivity technologies. Thus, the possibility to use other protocols on the AWSes to transport data instead of TCP, should be considered, looking for protocols more optimized for low bandwidth availability.

Through the execution of the test cases, we analyzed the results of the protocol in the scenario given. These results show how the protocol can fit in the technical specifications of an AWS, making possible to use it in future adaptations.

The main goal of this thesis has been to study state of the affairs in weather observation systems, their technologies and methodologies, trying to find ways of their improvement. OpenWeather fits that goal. Through the prototype we can show how the weather data transmission can be improved in several aspects from network topology to data structure use.

This topic suggests deeper research, as it could provide a solid basis for future implementation of a global real-time weather observation with a high capability in data exchange operations. In addition, in this thesis we have not treated security matters related with the weather data transmission. Despite the nature of the weather data, a complete solution has to consider security threats. Thus, an independent study is required to evaluate how the weather data transmission can be protected. Although, it would be possible to use cryptographic protocols such as Transport Layer Security (TLS) together with OpenWeather, such combination will have an impact on the bandwidth used to transmit weather data. In addition, Access Control List (ACL) mechanisms could be considered to assure the identity of the nodes and their locations, in order to guarantee their legitimacy. Moreover, weather data networks can be an objective of Denial-of-service (DoS) or Distributed denial-of-service (DDoS) attacks. Although this should be treated independently of OpenWeather protocol, future adaptions of it should have these threats in consideration to provide methodologies to lead with them.

The involvement of organizations such WMO and the vendors, is critical to make this happen, possibly in cooperation with standardization organizations for communication protocols such the IETF.In addition, any adaption of the industry to protocols designed and adapted for a most efficient use of resources available, will provide an improvement in their products, providing new ways to use their instruments to understand the weather phenomena.

Finally the author believes that the understanding of the weather phenomena will be accompanied by open and scalable network technologies. Thus, the Open-Weather protocol could be a first step to make it happen.

Bibliography

[1] C. Donald Ahrens. *Meteorology Today: an introduction to weather, climate, and the environment.* Brooks/Cole, Cengage Learning, ninth edition edition, 2009. ISBN 978-0-495-55573-5.

[2] ANSI/EIA/TIA-232-F. *Interface between Data Terminal Equipment and Data Circuit-Terminating Equipment Employing Serial Binary Data Interchange.*, 1997.

[3] ANSI/EIA/TIA-422-F. *Interface between Data Terminal Equipment and Data Circuit-Terminating Equipment Employing Serial Binary Data Interchange.*, 2005.

[4] ANSI/EIA/TIA-485-F. *Interface between Data Terminal Equipment and Data Circuit-Terminating Equipment Employing Serial Binary Data Interchange.*, 2003.

[5] Internet Assigned Numbers Authority. Port number assignments, 2011. URL `http://www.iana.org/assignments/port-numbers`. [Online; accessed 2011-08-15].

[6] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986 (Standard), January 2005. URL `http://www.ietf.org/rfc/rfc3986.txt`.

[7] S. Bradner. The Internet Standards Process – Revision 3. RFC 2026 (Best Current Practice), October 1996. URL `http://www.ietf.org/rfc/rfc2026.txt`. Updated by RFCs 3667, 3668, 3932, 3979, 3978, 5378, 5657, 5742.

[8] G. Camarillo and IAB. Peer-to-Peer (P2P) Architecture: Definition, Taxonomies, Examples, and Applicability. RFC 5694 (Informational), November 2009. URL `http://www.ietf.org/rfc/rfc5694.txt`.

[9] Flavio Castelli. Qjson, 2011. URL `http://qjson.sourceforge.net/`. [Online; accessed 2010-12-02].

[10] Bram Cohen. Bittorrent protocol specification v1.0, 2008. URL `http://wiki.theory.org/BitTorrentSpecification`. [Online; accessed 2010-11-02].

[11] Gerald Combs and contributors. Wireshark -protocol analyzer-, 2011. URL `http://www.wireshark.org/`. [Online; accessed 2011-07-26].

[12] Microsoft Corporation NEC Corporation Koninklijke Philips Electronics N.V. Compaq Computer Corporation Hewlett-Packard Company Intel Corporation, Lucent Technologies Inc. *Universal Serial Bus Specification Revision 2.0*, 200.

[13] Unicode Consortium. *The Unicode Standard - Version 6.0 – Core Specification-*. Unicode Consortium, 2011. ISBN 978-1-936213-01-6.

[14] World Wide Web Consortium. W3c xml schema definition language (xsd), 2009. URL `http://www.w3.org/TR/xmlschema11-2/`.

[15] Intel Corporation. Endianness white paper. Technical report, Intel Corporation, 2004.

[16] D. Crockford. The application/json Media Type for JavaScript Object Notation (JSON). RFC 4627 (Informational), July 2006. URL `http://www.ietf.org/rfc/rfc4627.txt`.

[17] D. Crockford D. Crockford D. Crockford D. Crockford D. Crockford. Json (javascript object notation), 1999. URL `http://www.json.org/`. [Online; accessed 2010-11-08].

[18] CWOP. Citizen weather observer program, 2011. URL `http://www.wxqa.com/`.

[19] ECMWF. European centre for medium-range weather forecasts, 2011. URL `http://www.ecmwf.int/`. [Online; accessed 2011-05-29].

[20] Third edition, editor. *ISO 8601:2004(E). Data elements and interchange formats, Information interchange. Representation of dates and times.* ISO, 2004.

[21] K. Egevang and P. Francis. The IP Network Address Translator (NAT). RFC 1631 (Informational), May 1994. URL `http://www.ietf.org/rfc/rfc1631.txt`. Obsoleted by RFC 3022.

[22] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. URL `http://www.ietf.org/rfc/rfc2616.txt`. Updated by RFCs 2817, 5785.

[23] FMI. Finnish meteorological institute, 2011. URL `http://www.ilmatieteenlaitos.fi/`. [Online; accessed 2010-09-22].

[24] The APRS Working Group. *APRS protcol reference*. APRS Working Group, 2000.

[25] David Helms. *CWOP Weather Station Siting, Performance, and Data Quality Guide*. CWOP, 2005.

[26] IETF. Internet engineering task force, 2011. URL http://www.ietf.org. [Online; accessed 2010-09-13].

[27] ISO. *ISO 80000 - Quantities and Units-*. International Organization for Standardization, 2009.

[28] G. Klyne and C. Newman. Date and Time on the Internet: Timestamps. RFC 3339 (Proposed Standard), July 2002. URL http://www.ietf.org/rfc/rfc3339.txt.

[29] Meteoclimatic. Meteoclimatic, 2011. URL http://www.meteoclimatic.com. [Online; accessed 2011-03-25].

[30] D. Mills. Network Time Protocol (Version 3) Specification, Implementation and Analysis. RFC 1305 (Draft Standard), March 1992. URL http://www.ietf.org/rfc/rfc1305.txt. Obsoleted by RFC 5905.

[31] NOAA. Meteorological station location information, 2011. URL http://weather.noaa.gov/tg/site.shtml. [Online; accessed 2011-06-12].

[32] NOAA. National oceanic and atmospheric administration, 2011. URL http://www.noaa.gov. [Online; accessed 2011-01-16].

[33] Nokia. Qt framework, 2011. URL http://qt.nokia.com/. [Online; accessed 2011-04-01].

[34] National Institute of Standards and Technology. *Secure Hash Standard (SHS)*. FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION. Information Technology Laboratory, National Institute of Standards and Technology, October 2008. URL http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf.

[35] Elth Ogston and Stephen A. Jarvis. Peer-to-peer aggregation techniques dissected, 2009.

[36] International Civil Aviation Organization. Meteorological service for international air navigation (metar). Technical report, International Civil Aviation Organization-WMO, 2007.

[37] Daniel Pasette. Binary-json, 2011. URL http://bsonspec.org/. [Online; accessed 2010-07-12].

[38] J. Postel. File Transfer Protocol specification. RFC 765, June 1980. URL http://www.ietf.org/rfc/rfc765.txt. Obsoleted by RFC 959.

[39] J. Postel and J. Reynolds. File Transfer Protocol. RFC 959 (Standard), October 1985. URL http://www.ietf.org/rfc/rfc959.txt. Updated by RFCs 2228, 2640, 2773, 3659, 5797.

[40] SMEAR. Project. University of Helsinki, 2010. URL http://www.atm.helsinki.fi/SMEAR/. [Online; accessed 2010-09-10].

[41] P. Srisuresh, B. Ford, and D. Kegel. State of Peer-to-Peer (P2P) Communication across Network Address Translators (NATs). RFC 5128 (Informational), March 2008. URL `http://www.ietf.org/rfc/rfc5128.txt`.

[42] Jaan Tallinn. Documentation of the known parts of the fasttrack protocol, 2001. URL `http://developer.berlios.de/projects/gift-fasttrack/`. [Online; accessed 2011-07-03].

[43] Jack Taylor William A. Beech, Douglas E. Nielsen. *AX.25 Link Access Protocol for Amateur Packet Radio*. Tucson Amateur Packet Radio Corporation, 1997.

[44] WMO. World meteorological organization, 2011. URL `http://www.wmo.int`. [Online; accessed 2011-01-19].

[45] WMO. Development of the wmo core profile of the iso metadata standard, 2011. URL `http://www.wmo.int/pages/prog/www/WDM/Metadata/documents.html`. [Online; accessed 2011-02-09].

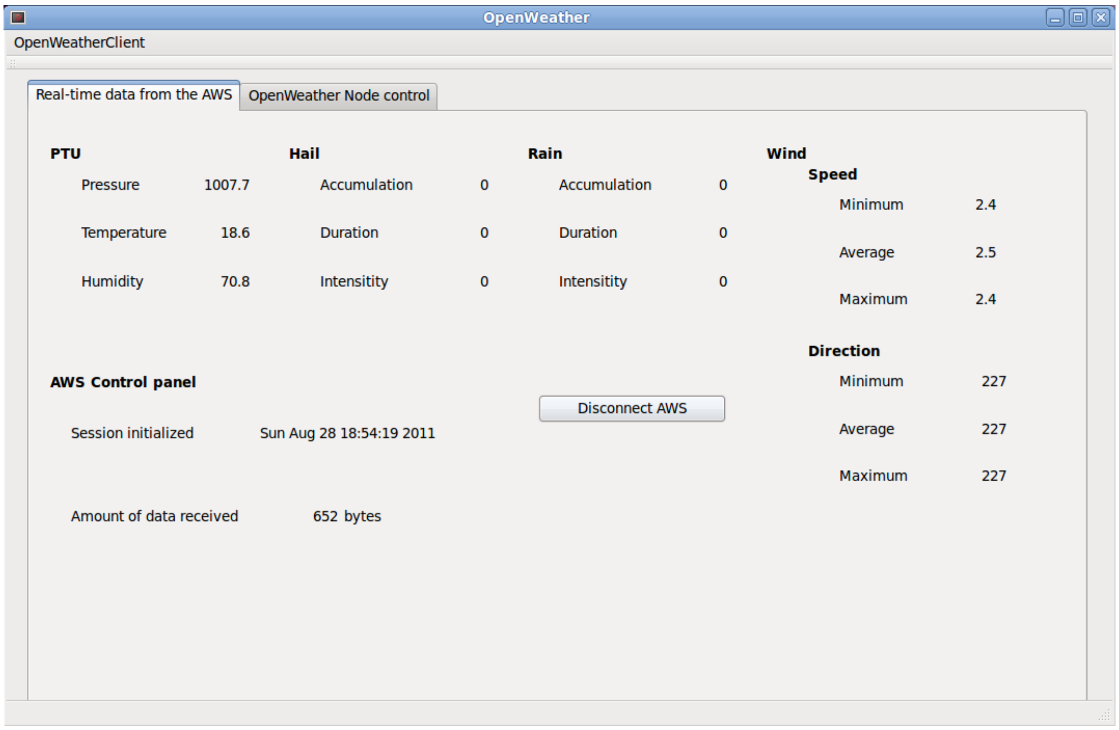
[46] World Meteorological Organization (WMO). *Guide to Meteorological Instruments and Methods of Observation*. WMO, 2008. ISBN 978-92-63-100085.

[47] F. Yergeau. UTF-8, a transformation format of ISO 10646. RFC 3629 (Standard), November 2003. URL `http://www.ietf.org/rfc/rfc3629.txt`.

Appendix I

Protocol code	**Description**	**Category**
100	HANDSHAKE	Protocol codes - Requests
101	HANDSHAKE-S	Protocol codes - Status
102	SERVICES-AVAILABLE	Protocol codes - Requests
103	SERVICES-AVAILABLE-R	Protocol codes - Retrievals
104	SERVICES-AVAILABLE-S	Protocol codes - Status
104	LIST-PEERS	Protocol codes - Requests
105	LIST-PEERS-R	Protocol codes - Retrievals
106	LIST-PEERS-S	Protocol codes - Status
200	REAL-TIME-DATA	Peer codes - Requests
201	ON-DEMAND-DATA	Peer codes - Requests
300	REAL-TIME-DATA-R	Peer codes - Retrievals
301	ON-DEMAND-DATA-R	Peer codes - Retrievals
500	REAL-TIME-DATA-S	Peer codes - Status
501	ON-DEMAND-DATA-S	Peer codes - Status

Appendix II

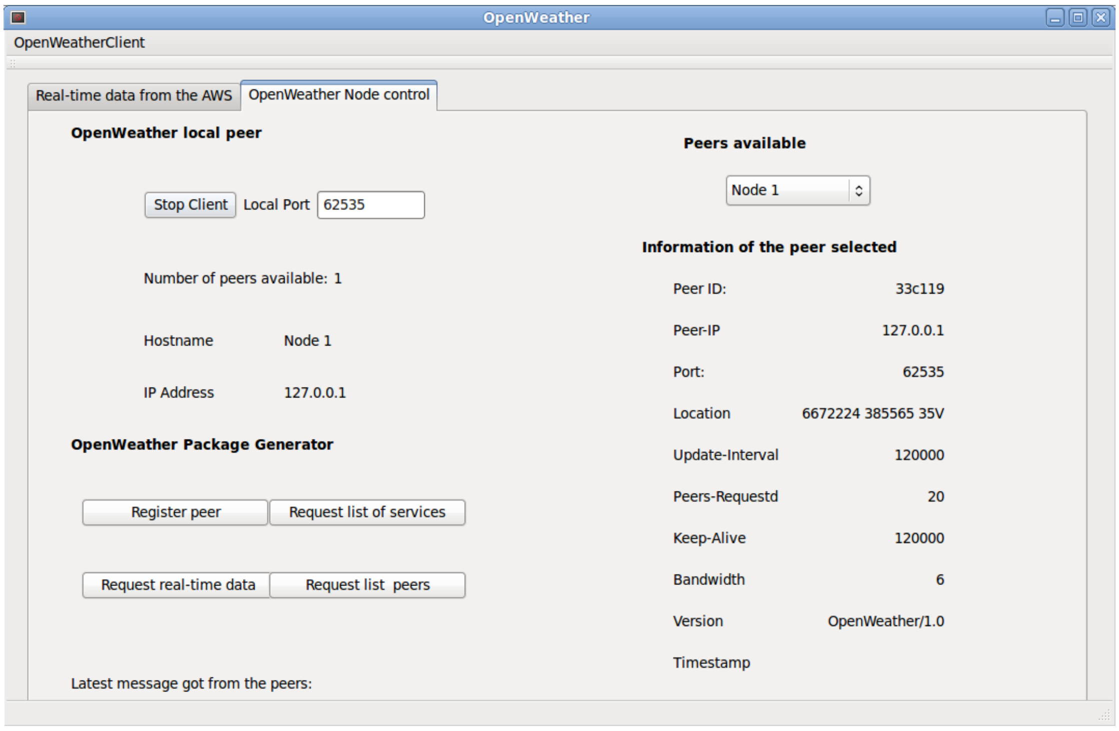


PTU		**Hail**		**Rain**		**Wind**			
							Speed		
Pressure	1007.7	Accumulation	0	Accumulation	0			Minimum	2.4
Temperature	18.6	Duration	0	Duration	0			Average	2.5
Humidity	70.8	Intensitity	0	Intensitity	0			Maximum	2.4
							Direction		
								Minimum	227
								Average	227
								Maximum	227

AWS Control panel

Session initialized	Sun Aug 28 18:54:19 2011
Amount of data received	652 bytes

Figure 8.1: GUI of the OpenWeather prototype -AWS control-.



OpenWeather local peer

Local Port 62535

Number of peers available: 1

Hostname	Node 1
IP Address	127.0.0.1

OpenWeather Package Generator

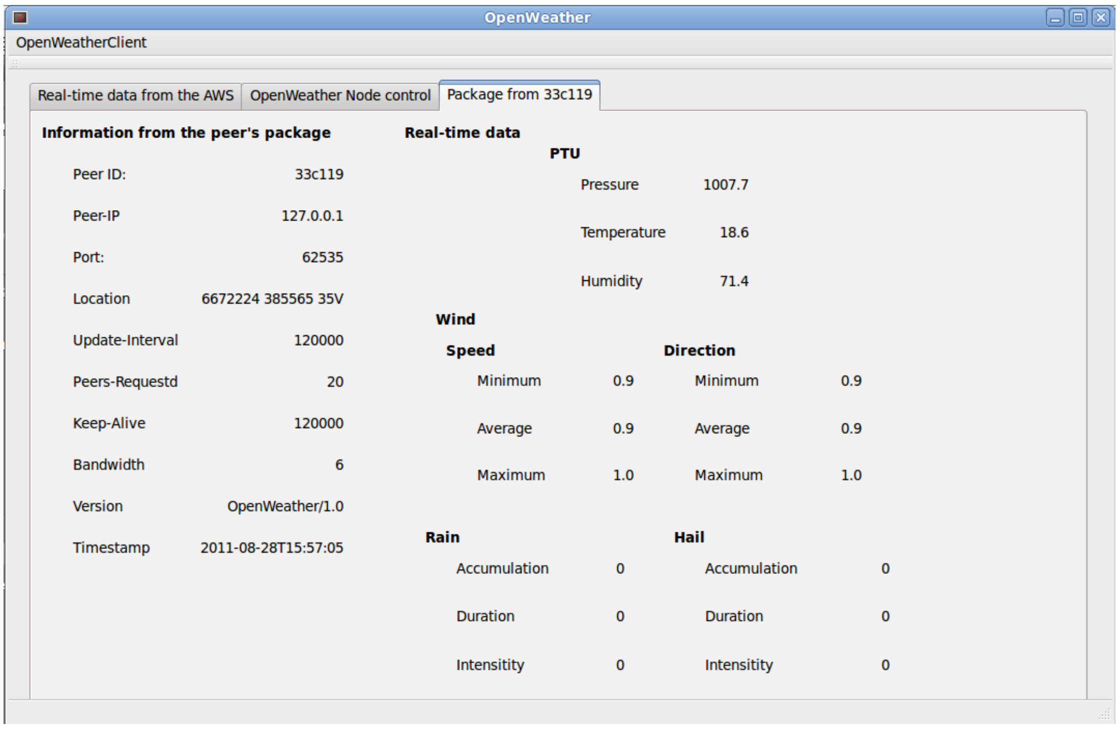
Latest message got from the peers:

Peers available

Information of the peer selected

Peer ID:	33c119
Peer-IP	127.0.0.1
Port:	62535
Location	6672224 385565 35V
Update-Interval	120000
Peers-Requestd	20
Keep-Alive	120000
Bandwidth	6
Version	OpenWeather/1.0
Timestamp	

Figure 8.2: GUI of the OpenWeather prototype -Node control-.



Information from the peer's package

Peer ID:	33c119
Peer-IP	127.0.0.1
Port:	62535
Location	6672224 385565 35V
Update-Interval	120000
Peers-Requestd	20
Keep-Alive	120000
Bandwidth	6
Version	OpenWeather/1.0
Timestamp	2011-08-28T15:57:05

Real-time data

PTU

Pressure	1007.7
Temperature	18.6
Humidity	71.4

Wind

Speed		**Direction**	
Minimum	0.9	Minimum	0.9
Average	0.9	Average	0.9
Maximum	1.0	Maximum	1.0

Rain		**Hail**	
Accumulation	0	Accumulation	0
Duration	0	Duration	0
Intensitity	0	Intensitity	0

Figure 8.3: GUI of the OpenWeather prototype -Data message visualizer-.