A Concept for a High-reliability Meteorological Monitoring System Using AMQP

Anna Kostromina[1], Eduard Siemens[1] and Yurii Babich[2]

[1]*Future Internet Lab Anhalt,*
Anhalt University of Applied Sciences, Bernburger Str. 55, 06366 Köthen, Germany,

[2]*Department of Telecommunication Networks,*
Odessa National O.S. Popov Academy of Telecommunications, Odessa, Ukraine
anna.kostromina@student.hs-anhalt.de, eduard.siemens@hs-anhalt.de, y.babich@onat.edu.ua

Keywords: AMQP, Rabbit MQ, Meteorological Monitoring, Reliable Message Queues, Availability

Abstract: This paper describes a concept for a resilient meteorological monitoring system for reading data from sensors by using AMQP in order to increase reliability of the data acquisition system. A set of sensors is connected to the Beaglebone Black and is located in the mountains of north of Thailand. Gathered data are queued on a local SoC and sent to a server located in Germany whenever a network connection is available. Further in the work implementation and test of such a system in Thailand is discussed. Special challenges in the implementation of the system is the presence of frequent thunderstorms and outages caused by them. To improve the reliability of data transmission free AMQP implementation is used. The protocol has been studied, tested, and programs have been created for transmitting data from sensors to a server in Germany using the Rabbit MQ to store the data in the case of connection failures. Besides that, memory usage problems were raised when using AMQP in single-board computers, such as Beaglebone Black. The main task of this work is to propose the most stable and reliable operation of the data transmission system.

1 INTRODUCTION

Monitoring systems are nowadays used in a very wide range of technical and technological systems. This paper presents a concept of a meteorological monitoring system running in a rural area under harsh weather conditions in the north of Thailand. The goal of this research was to design and to deploy a monitoring station in an indigenous village of a Karens tribe. The station shall be involved in long-term monitoring of weather conditions in order to obtain the data required for simulations of energy availability and consumption minimization in the case of energy-autarkic cooling houses, designed for the ASEAN region.

This work was part of the preparatory work for the Silaa Cooling project where the team of the Future Internet Lab Anhalt (FILA), along with three industrial partners from Germany and one Thailand company are going to develop a coffee-cooling system for long-term coffee storage. Figure 1 shows a typical village, at which such a coffee cooling system has to be built and operated.

The main technical challenges under the given conditions are as follows:

- The system must have a battery buffered energy supply due to frequent power outages at the facility
- The system needs a solid weather protection but also a flash protection of all the electronic and electrical components due to frequent thunderstorms
- All the data must be gathered and reliably transmitted to a cloud environment at the facilities in Germany via a GSM modem. Since the internet connectivity at the location is very unstable and drop-offs of the internet connection can last for days or even weeks, some robust data queuing and delivery must be implemented, which stores data locally and, delivers queued data whenever the internet connectivity to the domestic cloud is available.

The queueing system shall have a persistncy of data which prevents data loss if a power outage occurs at any stage of data acquisition and queueing

- The data must be kept in a quite generic way since at the time of gathering of the data the exact needs of later data processing is not yet completely known. So, it is advisable to keep the data at the receiving site in two formats – in a non-growing data-base with data aggregation and compression, primarily for visual representation of the data which must be available in semi-real-time. Along with that, in a raw data format without data reduction is needed, for later in-depth analysis of the particular weather parameters.

Figure 1: Village in the Karens area at which the weather data have to be gathered.

2 RELATED WORK

The development of self-sustained and energy-autarkic systems significantly grows in the importance during recent years. Especially in the development aid as well as in the implementation of decentralised energy production. The chain between information and communication is crucial and promises to enable implementation of economically reasonable systems for assuring a subsistence of people in rural, mostly off-grid, areas. One of the approaches similar to SilaaCooling is the ColdHubs system. That project aims at supporting people in Africa to store the food in solar-powered cold rooms [1]. However, the technical approach of ColdHubs is quite static, the energy provision is secured by some overprovisioning of the amount of photovoltaic modules and of the lead batteries capacity, which makes hard to keep the deployment costs reasonably low. In contrary to ColdHubs, the SilaaCooling concept includes heuristic and proactive energy management, based on weather and energy demands prediction, operated on a computer on-side, equipped with a collection of sensors of ambient condition. The idea and technical realization of SilaaCooilng bases on a concept of the implementation of energy-autarkic decentralised energy supply for small villages in Siberia, developed in one of previous projects [2].

However, reliable metrological and weather data acquisition and delivery is also used in other application cases. So in [3] a data acquisition system consists of a set of wireless sensors for measuring meteorological parameters. While the focus in that research is on the local system design for connecting wireless sensors, our focus is mostly on the implementation of a robust resilient data delivery system.

To avoid data loss on data delivery to the cloud, existing transmission and queuing protocol called Advanced Message Queueing Protocol (AMQP) [4] has been investigated and tested. Among several AMPQ implementations, the Rabbit MQ system has been compared with Active MQ. A good comparison of the said implementations is given in the article “The analysis of the performance of RabbitMQ and ActiveMQ” [5]. Furthermore, in [6] and [7] an in-depth performance analysis for distributed message delivery using RabbitMQ is given. However, due to a quite low message load in our target scenario, performance is not considered as a bottleneck of our system. The mentioned resilience and persistency of data is in the foreground of our investigations.

A good and detailed tutorial of use of RabbitMQ is given in [8]. Here, a lot of important information about the work and writing programs for messaging using RabbitMQ is given.

3 THE PROJECT ELEMENTS

3.1 Hardware of the project

3.1.1 Components of monitoring system

In this work we mostly focus on the choice of a robust queuing and message delivery system able to work under the harsh conditions described above.

To build a self-sufficient food storage we need to collect the following weather data: ambient temperature, humidity, wind gust, solar irradiance. The latter parameter is crucial for the later dimensioning of the solar panel size and also for the size of battery buffer for the SilaaCooling system, to be developed.

For gathering data in Thailand, in Mai Lae, a Beaglebone Black (BBB) has been installed, running Debian 8 Linux. The cheapest and easiest way to get necessary weather parameters was to get an off the shelf weather station with a USB interface:

- WH1080 Radio weather station [9] is connected to the BBB, which gathers the ambient temperature, humidity, wind data, air pressure and amount of rain in a 10 minutes interval to the weather station via an UHF radio link. Up to 14 hours of data can be buffered on the weather station without data loss in case of downtimes of the BBB.
- For redundancy and for verification of the data gathered by the weather station, we have decided to add an additional sensor set to the system. For this an AM2302 temperature-humidity sensor, placed at a JeeNode board has been placed on a pillar next to the sensors of the weather station, which deliver their data via UHF to a JeeLink, plugged via a USB hub to the BBB.
- For solar irradiance metering a Si-RS485TC-T sensor, manufactured by Ingenieurbüro Mencke & Tegtmeyer GmbH [10], is used, which is connected to the BBB via an USB/RS 485 adapter. A daemon running on the BBB is reading the irradiance data in a 30 seconds cycle. In contrary to temperature and humidity, this sensor was not doubled, for cost reasons and since the sensor is a calibrated one with certification.

AM2302 is a temperature and humidity sensor used to verify temperature and humidity data, measured by the weather station. The sensor is mounted on a JeeNode which transmits data via UHF band to the Jeelink board, connected to the BBB, see Figure 2.

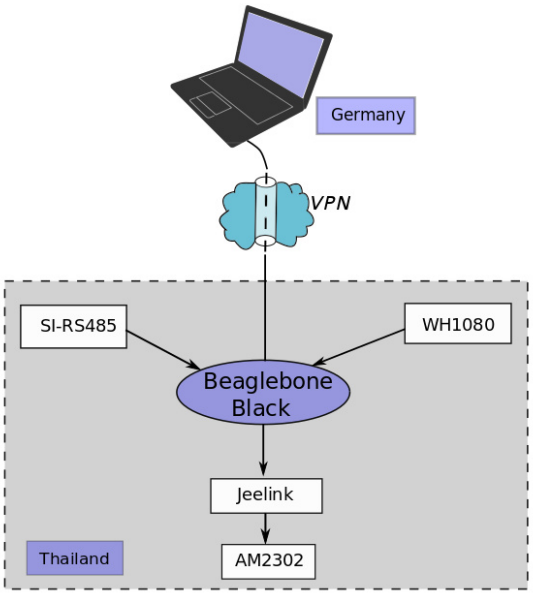

Figure 2: Components of monitoring system.

3.1.2 The data acquisition system

Data from the three different sources – the weather station, irradiance meter and JeeNode connected sensors are read by a tiny Python program. For reading data from weather station WH 1080, the API of “pywws” [11] is used by our Python program. The main loop of iteration through the different sensor data is running in a daemon fashion, which is monitored by the Supervisor daemon of Linux. The JeeNode sensor data are also written by a python script, which has been developed in a preparatory work for the SilaaCooling project. Figure 8 shows how output data from producers sending on Rabbit MQ and then on servers.

3.2 The software concept

The main idea in the implementation of the system is to use publicly available open-source code wherever it is possible for keeping the development effort for the system low. For gathering the weather data from the weather station, the open source python package “pywws” [11] has been used.

The main technical challenge of the system is however the message queuing and delivery system, which shall on the one side work on a low-performance SoC like BBB and on the other hand be very robust with avoidance of data loss even in the event of abrupt power outages, connection drops or link failures to the sensors. For this, the Advanced

Message Queuing Protocol (AMQP) has been deployed on the system [3]. Though this system is originally known as a message broker in banking transactions, the robustness shall be used in our design. While a wide range of AMQP implementations are known on the market, there are some lean implementations which promises to run on a low-performance PC platform.

Rabbit MQ is an open source AMQP suite implementation which provides an exhaustive Python API for Linux. Reasons to choose Rabbit MQ are the following [8]:

- RabbitMQ is the only one open source implementation of the AMQP standard besides Qpid, whereby Qpid doesn't provide a Python API
- Essential features like federation, clustering, persistency are implemented in RabbitMQ
- Clustering became simpler because of Erlang
- RabbitMQ is more reliable and crush resistant that it competitors

Clustering connects multiple machines together to form a single logical broker. Communication is via Erlang message-passing, so all nodes in the cluster must have the same Erlang cookie. The network links between machines in a cluster must be reliable, and all machines in the cluster must run the same versions of RabbitMQ and Erlang [12].

By default, each Rabbit MQ instance delivers its messages from the queue to the consumers using the Round-Robin algorithm. Also by default, each message it is deleted from the queue on delivery. To change this behaviour were used flag auto_delete=False

3.2.1 Implementation

The instantiation of a RabbitMQ broker consists of the following steps [8]:

1. Connect to Rabbit MQ
2. Obtain a channel
3. Declare an exchange
4. Create the message
5. Publish the message
6. Close the channel
7. Close the connection

3.2.1 The publish/subscribe model

In the application of weather station gathering two independent data destinations are necessary – one which delivers to a fixed size data base for monitoring purposes and another one with ever growing raw data. The both subscribers (consumers) can then be co-located at the same machine or at geographically remote locations.

For implementing this, a channel publishes two queues and two clients, which run AMPQ consumers, consumes the messages from the respective queue and stores the sensor data to each data base. This pattern is known as publish/subscribe pattern (Figure 3).

- Producer (P) is a user application that sends a message to an exchange
- Exchange(X) receives messages from producer and push them to the queues
- Queue is a buffer that stores the messages
- Consumer(C_1, C_2) is a user application that receives messages v

In figure 3 is shown as a working publish/subscribe algorithm.

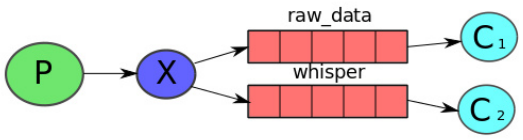

Figure 3: Publish/subscribe.

In case of publish/subscribe using the exchange type as fanout. From exchange data were sent into queues raw_data and whisper.

The following customization has to be applied to the default configuration of RabbitMQ to meet the requirements of the weather data gathering for SilaaCooling. The data gathering has to be reliable even in cases of frequent abrupt and long-term power outages. The first measure is making the message queues **persistent** and **durable**. With persistency, each data chunk written to the queue is copied to a SD card. Also the file system on the sd-card is not buffered, so at each instance the queue is mirrored to a non-volatile memory. This approach for sure reduces the transport performance of the queue to the read/write speed of the SD card. However, this must be pretty enough for delivering a few messages per minute with some hundreds of bytes per read/write. With the durability, the AMQP is saving **the configuration** of the published queues to a non-volatile memory so, on a re-start of the system and of the AMQP daemon (which provides the exchanges, channels and queues) the queues of the messages are

automatically published and so the clients can re-subscribe to the queue and continue the message reception from the queue.

Between reading the data from the queue by a consumer and bringing it to save harbour in a data base, some network errors, data base errors or file system errors can occur. So, a message shall be deleted from the queue only if the data base confirms the storage of the data item. For this an acknowledgment of messages has been configured in the RabbitMQ system.

3.2.2 Error resilience tests

Before the system can be deployed in the field in the mountains of Thailand, a series of scenarios of power disruptions, network errors and other failures has been defined, with which message integrity checks has been performed. The system for testing is shown on Figure 4.

The Rabbit MQ server and producer which read the sensor data are running locally on a Beaglebone Black.

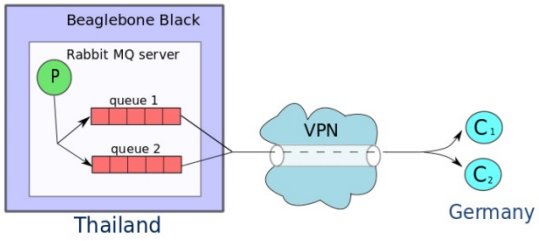

Figure 4: System for sending a data by using Rabbit MQ.

A first test scenario contains the following steps of tests:

1) abrupt disconnect of consumers (Figure 6)
2) crash of the BBB
3) stop a consumer and re-start after several minutes to hours
4) power on of the BBB
5) re-connect the producer (while the consumer is already running)

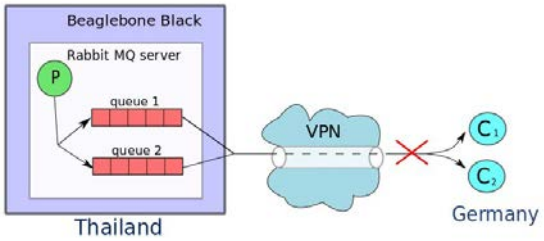

Figure 5: Disconnect of consumers.

In all the test iteration, no messages where lost. All of them has been delivered and stored in the respective data bases at the consumer's site. n this case messages were successfully delivered after the disconnect.

In a second series of tests, the behaviour of the system in case of producer crash or BBB crash including the Rabbit MQ server is investigated. For that the following sequence has been executed:

1) Disconnect of consumers
2) Killing Rabbit MQ server and producer (Figure 6)
3) Restarting Rabbit MQ
4) Reconnect consumers
5) Restart of the producer

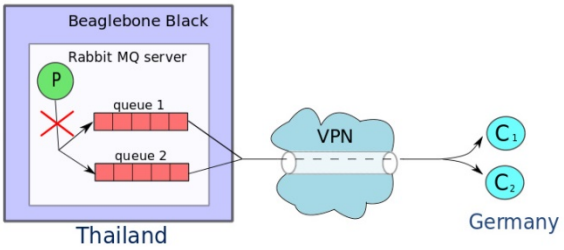

Figure 6: Crush of Rabbit MQ producer and server.

All messages were delivered after restart. Testing were successful.

3.3. Final concept of system for data transmission

Finally we have implemented the concept as follows. Figure 7 shows how data from producers are sent to consumers. For each program a producer (solar_producer, pywws_producer, silaacooling_producer) is instantiated and then data are written to the queue through the AMQP exchange. For every program two queues are created – the raw_queue and whisper_queue. All data from raw queues are going to the consumers on the cloud server and data from the whisper queues are going to the consumers on the other server. These servers are located in Germany.

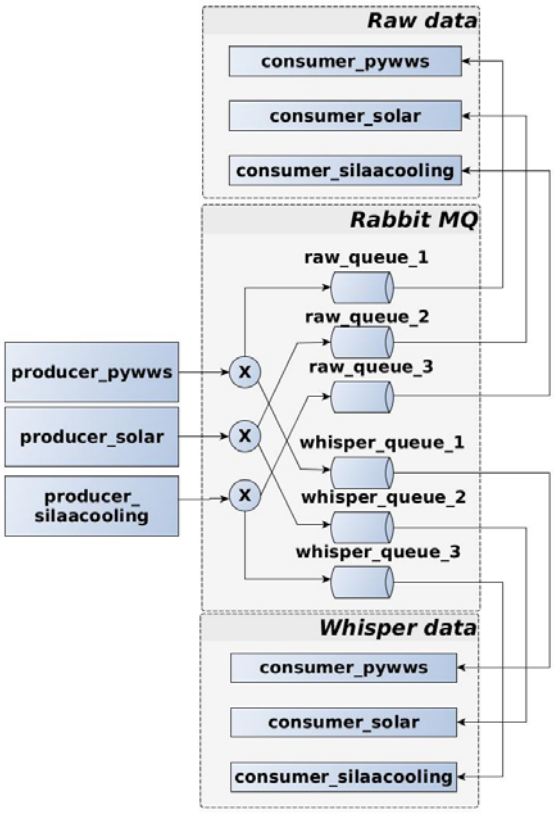

Figure 7: Sending data on Rabbit MQ.

4 PREDICTIVE MONITORING OF MEMORY USAGE FOR RABBIT MQ

Rabbit MQ provides a basis for a robust and error resilient message delivery system in presence of a variety of network and system errors in the field, which is lean enough to run on a single-board computer with rather limited CPU and memory resources. However, crashes due to memory overruns can lead to message loss, which must be avoided by additional means. To control memory usage we propose to implement a predictive monitoring approach described in [15]. In this case, the monitored parameter is the value of memory in use marked *y*. It makes sense to use only one threshold value corresponding to available memory denoted as y_{kP}. If the value of memory in use reaches the threshold value we consider having emergency situation, which must be prevented by means of the predictive monitoring. In order to minimize load on single-board computer resources (memory, processor) we use only one forecast horizon out of three specified in [15]. The following expression corresponds to the normal operation of the system.

$$\hat{y}\left(n_{\min}^{C} + L_C\right) < y_{kp}, \quad (1)$$

where $\hat{y}$ denotes a forecasted value of the *y* parameter estimated using the minimum required series of the $n_{\min}^{C}$ size with the time horizon of L_C.

If the term (1) is violated memory usage predictive monitoring system alerts about the expected emergency situation in order to let take actions and prevent the situation. If it is impossible to take any actions due to lack of connection, than the predictive monitoring system deletes some data from the memory in order to prevent system crash.

One message of *pywws* program contains 98 bytes on average, one message of the *solar_producer* program (for reading Irradiance) contains 93 bytes and one message of *silaacooling_producer* program contains 362 bytes. Consequently, for 24 hours without connecting to consumers the data will use 41.376 Kb of memory.

In the case of Beaglebone Black or other type of SoC it is very important to check if there is enough memory or not. In particular case is good decision to use some mechanism for removing old log files from the system without creating problems when reading data. In case of using a similar system a calculation, how long it will take to fill the entire free space on the computer's disk is inevitable.

5 CONCLUSIONS

A system design, for data delivery from the acquisition part to different consumers positioned thousands of kilometres away, interconnected via narrowband erroneous GPRS links is shown in the paper. Stable and resilient data delivery without message losses, even in cases of network outages for several days has been reached with the system. It has turned out as very flexible, so the location of the consumers can be easily migrated from one location to another one. Using two different queues leads us to a very convenient decoupling of different kinds of use of the data for post-processing. The memory usage predictive monitoring system has been implemented in order to prevent system crash caused by the lack of available memory, which is crucial for the Rabbit MQ implementation on a single-board computer.

REFERENCES

[1] Project Coldhubs, http://www.coldhubs.com/

[2] S. Zinov, E. Siemens. "The Smart Lighting Concept". Workshop on Problems of Automomous Power Systems in the Siberian Region. Koethen, October 2013.

[3] M.Benghanem, "Measurement of meteorological data based on wireless data acquisition system monitoring", Applied Energy, pp. 2651-2660, December 2009

[4] S. Vinoski, "Advanced Message Queuing Protocol", IEEE Internet Computing, vol. 10, Issue 6, pp. 87-89, November 2006

[5] V. M. Ionescu, "The analysis of the performance of RabbitMQ and ActiveMQ", IEEE, Romania, pp. 132-137 , October 2015

[6] Jones, et. al., "RabbitMQ Performance and Scalability Analysis", project on CS 4284: Systems and Networking Capstone, Virginia Tech 2011

[7] M. Rostanski K. Grochla,A. Seman "Evaluation of highly available and fault-tolerant middleware clustered architectures using RabbitMQ", IEEE, Poland, pp 879-884, October 2014

[8] A. Videla and J. Williams, "RabbitMQ in action. Distributed messaging for everyone." Manning, April 2012.

[9] "Radio Weather Station with USB and Touchscreen" [Online]; http://www.produktinfo.conrad.com/datenblaetter/650000-674999/672861-an-01-ml-FUNK_WETTERSTATION_MIT_USB___TOUCH_de_en.pdf

[10] "Digital silicon Iradiance Sensor Si-rS485-TC-T" [Online], http://imtsolar.com/wp-content/uploads/2013/02/Si-Sensoren_RS485_E.pdf

[11] J. Easterbrook, "pywws" [Online] https://github.com/jim-easterbrook/pywws

[12] "Distributed RabbitMQ brokers" [Online], http://www.rabbitmq.com/distributed.html

[13] "RabbitMQ documentation" [Online], http://www.rabbitmq.com/documentation.html, accessed 21.01.2014.

[14] "Advanced Message Queuing Protocol. Protocol Specification" [Online], https://www.redhat.com/f/pdf/amqp/amqp_0-8_specification.pdf

[15] Y.O. Babich and L.A. Nikityuk, "Functional improvement of monitoring the dynamic characteristics of information and communication networks", vol. 4/9 (76), Eastern European Journal of Enterprise Technologies, Kharkiv, 2015, pp.9-14.