Kenner *et al. EURASIP Journal on Embedded Systems* (2017) 2017:12
DOI 10.1186/s13639-016-0045-7

EURASIP Journal on Embedded Systems

RESEARCH Open Access

# Comparison of smart grid architectures for monitoring and analyzing power grid data via Modbus and REST

Susanne Kenner*, Raphael Thaler, Markus Kucera, Klaus Volbert and Thomas Waas

**Abstract**

Smart grid, smart metering, electromobility, and the regulation of the power network are keywords of the transition in energy politics. In the future, the power grid will be smart. Based on different works, this article presents a data collection, analyzing, and monitoring software for a reference smart grid. We discuss two possible architectures for collecting data from energy analyzers and analyze their performance with respect to real-time monitoring, load peak analysis, and automated regulation of the power grid. In the first architecture, we analyze the latency, needed bandwidth, and scalability for collecting data over the Modbus TCP/IP protocol and in the second one over a RESTful web service. The analysis results show that the solution with Modbus is more scalable as the one with RESTful web service. However, the performance and scalability of both architectures are sufficient for our reference smart grid and use cases.

**Keywords:** Smart grid, Real-time monitoring, Modbus, HTTP performance

## 1 Introduction

Researches at the smart grid topic are widespread and in progress worldwide. In [1], the authors published a survey on smart grid concepts and architectures in India, China, USA, and Europe and explained the different starting points and reasons of their studies.

In Germany, the transition from conventional power producers to renewable energy sources like wind and solar is one of the key points for researching. The increasing amount of distributed volatile energy production of renewable energy sources has a negative impact on the stability of the grid and in addition the demand and cost of energy will increase in the future. With the integration of communication technologies, sensor nodes, and smart regulation algorithms into the existing power grid, it is possible to counteract these effects. The sensor nodes (energy analyzers) are able to measure energy data like power and voltage from the grid and provide this data for monitoring and analyzing. Based on these data and analysis, further concepts can be developed for reducing energy demand and costs [1–7].

Electrical power supply is demand-oriented to date and not flexible enough for the challenges described above. In the future, it will be necessary to change the energy demand according to power generation of renewable energy sources, which means that the regulation will be moved from electricity suppliers to the consumer's side. Through suitable concepts of demand regulation and distribution, the power grid will be stabilized and optimized [8]. Possible concepts are demand side management and demand response, which are described in [9–11]. Demand side management techniques like load shifting and peak clipping are the most known approaches. At load shifting, the load is shifted from a peak period to an off-peak period without changing the total energy consumption of both periods, whereas at peak clipping, the load is reduced by reducing the power consumption [12]. To use these techniques, it is necessary to identify load peaks through analyzing power data from the grid. Based on analysis, an algorithm for shifting load peaks could be developed [13].

In this paper, we present two different architectures for gathering data from energy analyzers distributed at power grid of the Technical University of Applied Sciences

*Correspondence: susanne1.kenner@oth-regensburg.de
Department of Computer Science and Mathematics, Technical University of Applied Sciences, Regensburg, Germany

Regensburg (OTH) as well as a software application for data monitoring, visualizing, and analyzing. We compare the performance of data collection of both architectures with respect to real-time monitoring, load peak identification, and automated regulation. This article is an extended version of a conference paper published at 12th International Workshop on Intelligent Solutions in Embedded Systems (WISES), 2015 [14]. For a better understanding of the analysis, we explain the hardware, developed software, and test environment in more detail.

The paper is organized as follows: Section 2 describes the backgrounds of our work and name some similar works in this area. Section 3 presents our software application with two different system architectures. Section 4 discusses performance tests of collecting data over the network and compares the different architectures. Finally, Section 5 concludes this paper.

## 2 Motivation and related work

In the past few years, our research focus at OTH has been on communication networks and security in smart grids. In [15], mobile communication tests to acquire data from a medium voltage grid were carried out. The authors present a mobile communication architecture and tests of latency, transfer rate, reliability, and security for a smart grid application. Furthermore, practical tests of different mobile communication standards and providers on mobile routers were carried out, which are used in the same project cited above [16]. Other scientific groups at OTH are focused on electromobility, energy, and power management. So many energy measurement devices were integrated in the local power network for analyzing the grid data and to get a better comprehension over the power grid. Our software application "smart energy campus" collects, monitors, and displays measurements to get an overview on the grid. The application is written in Java, collects data from measurement devices every second, and saves them at a NO-SQL database Apache Cassandra. By using a web page, the user can view and analyze data like power, effective power, voltage, active energy, and further measurements as described in detail in Section 3.3. The project "smart energy campus" aims to analyze the grid in order to get a better comprehension about the technologies, monitoring and visualizing the grid. In the next step, we are going to analyze the data in order to identify potentials for an automated energy and power management with focus on load shifting and peak clipping. To realize this target, the software application needs functionalities like fast performance and low latency. Therefore, we test the performance of two different system architectures for data collecting and see whether these are usable for such a scenario.

Different architectures for smart grid applications are possible. Depending on the requirements of the applications, services, and environments, the suitable architecture must be chosen. Performance tests are one possible decision support for choosing an architecture. For example, in [17], the authors analyze the performance of different data processing architectures in smart grids. We analyze the performance and latency of the Modbus TCP/IP protocol and requests over a RESTful web service in our system. There are several systems in literature, in which web services in smart grid systems are used, for example, in [4–6, 18].

The Modbus and HTTP protocols are popular and used in industry and Internet for a long time. So some performance tests have been carried out. In [19], the authors present performance tests of the Modbus TCP/IP protocol with regard to the increasing number of real-time scenarios. Another real-time scenario is shown in [20]. In [21], the authors evaluate the performance of the Modbus TCP/IP protocol with focus on the response time. There are also some research papers where the authors analyze the performance of web services. For example, in [22], the authors compare the performance of RESTful web services and the Advanced Message Queuing Protocol. In [23], the authors evaluate the performance of REST and SOAP web services for mobile devices. Analysis of embedded web services for machine-to-machine communication is presented in [24]. In the last named paper, the author refers to the limits of web services in scenarios with a large number of data. For energy monitoring and an automated energy and power management, we have to handle a large number of data. In one architecture, we use the software GridVis from Janitza. This software is used by the Maintenance Service (MS) of our university. We test the performance of the RESTful web service of the GridVis Software below, especially to find out how much data the web service in one request provides and if it is possible to request a large number of data every second for our real-time scenarios. In contrast to the papers cited above, we carry out practical performance tests on a real power grid.

Additionally, we implement a software application, which visualizes the collected data and make them available for analyzing. There are some projects in the topic of monitoring and visualization cited below. In [25], the authors present an analysis of a Geographical Information System to display real-time data on maps. A further project is described in [26]. There, the authors present an energy monitoring system with 3D views and energy consumption data of a city. Another project presents the energy management system framework WattDepot, which is an open source framework and also visualizes energy consumption data on a map, for example, in [27]. In our work, we not only want to display data but also want to simplify the process of analysis with respect to identifying periodically load peaks to regulate the power

consumption. We want to automate the process of analysis of load peaks and so we implement an own software for these purposes.

## 3 Energy campus

About 35 energy analyzers are installed in the local smart grid. Inter alia, we can analyze data from six main low-voltage transformers, big electrical consumers like the canteen, laboratories, and the computer center. Data from small photovoltaic plants are also available. Applications written in java collect data and view them on a web page. The next section describes two different system architectures, which were designed and adjusted during project progression.

### 3.1 Architecture A

This architecture was designed at the beginning of the project and is shown in Fig. 1. It shows three main areas representing three stakeholder groups, which need the data from energy analyzers for research, daily operations, or cost control. The stakeholder groups are the Department of Electrical Engineering and Information Technology (EI), Department of Computer Science and Mathematics (IM), and the Maintenance Service (MS).

*Hardware*: Each group works on a virtual server machine with Windows Server 2008 R2 Enterprise operating system with Service Pack 1. The virtual Server of IM has 2 Intel(R) Xeon(R) E5645 processors with 2.40 GHz and 4-GB main memory. It is connected to the network over a 1 GBit/s Ethernet interface. The virtual server of EI has 2 Intel(R) Xeon(R) X5650 processors with 2.67 GHz and also 4-GB main memory. The connection to the network is accomplished with a 10 GBit/s Ethernet interface. All servers are connected to the network of the OTH with a large amount of users at university campus.

On a previous project, energy analyzers from Siemens (Siemens PAC4200) [28] and Janitza (UMG96RM) [29] companies were integrated in the local power grid and connected with the IT network via 10/100 MBit/s Ethernet. The energy analyzers are able to measure about 300 different values from the power grid. Both device types are equipped with a display unit to receive a quick overview of the current situation of the relevant power grid section and in addition, the devices provide Modbus RTU and Modbus over TCP/IP as communication protocols, which can be used to request data from the devices and for configuration. The devices refresh the measurements every 200 ms. This is the smallest interval for requesting data. In our scenario we collects data over the Modbus TCP/IP protocol every second.

*Software*: The companies Siemens and Janitza sell software applications for their energy analyzers to collect data and visualize them. These commercial products are used from EI and MS. The application Siemens Powermanager is used for training lessons and for visualizing information about the local Smart Grid on monitors, which are installed around the campus. For this purpose, the Powermanager is useful but it needs a long training period due to its complexity. A little bit easier to handle is the software application GridVis from Janitza, which is used by the MS. Both applications collect data over the Modbus over TCP/IP interface from energy analyzers and save them in a relational database. The Powermanager software provides many kinds of visualization, but it is not possible

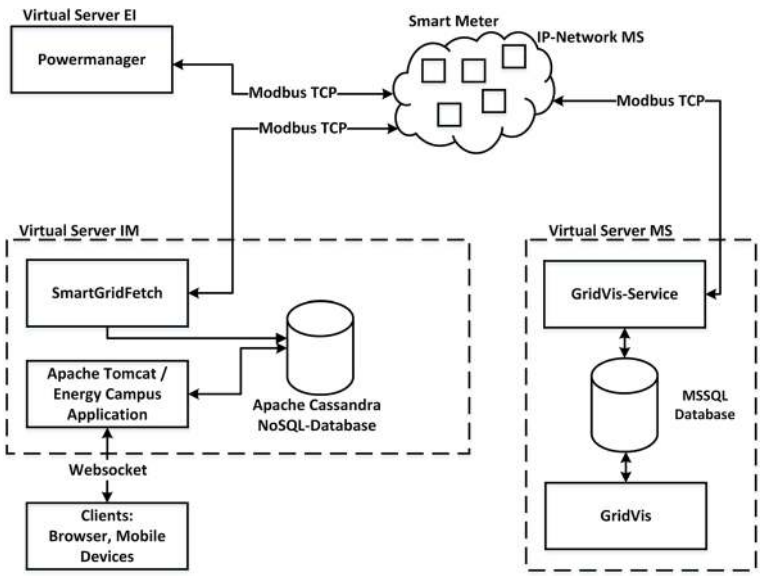


**Fig. 1** Architecture A

to connect devices from other companies to this software and to extract out collected data. The MS uses GridVis to get an overview of the grid and to control energy costs. It can integrate devices from other companies, but they not fully supported through the application, so some functions are not available. Additionally, the possibilities to visualize the data in charts are limited. Both applications provide data collection every second, but they do not save them in a database. However, second-by-second collected measurements are very interesting for research in the power grid. For example, the EI analyzes relationships of different measurements and the changes of the values from one second to the next. Therefore, we implement our own java application to be independent and to eliminate the imperfections of the commercial software applications. To simplify the visualization and analysis of the grid data and to create charts especially for load peak analysis are other aspects to implement our own application.

The software SmartGridFetch was implemented at a previous student work and continually developed and adjusted on the circumstances of the current work. It runs on the virtual machine of IM, collects data from energy analyzers over Modbus protocol, and saves them in the Apache Cassandra database. The software is written in Java and consists essentially of two main classes for requesting and receiving data over the Modbus TCP/IP protocol and for saving the values in a database. The software starts a thread pool with several threads for requesting data from energy analyzers. Each thread requests 18 measurements from one device every second. After receiving the response, each thread saves the measurements with a timestamp at database. The settings and device parameter like IP address, device description, measurements to be requested, and further settings can be set over an option class. The measurements were requested over the Modbus Read Holding Register function, so several continuous registers can be read out in one request. The initial SmartGridFetch application only provides requests of continuous register blocks over Modbus TCP/IP and only Siemens PAC4200 devices. For the current work, we adjust the application, so that it is possible to request measurements saved at discontinuous register blocks and to support different device types, especially the Janitza devices we use. With these features, the software is more flexible.

Cassandra is a NoSQL (not only SQL) column-oriented database and has a flexible database scheme [30], which could be suited for changes of the energy analyzer infrastructure. Furthermore, the database scales horizontally in contrast to the most relational databases, which scale vertically. In a smart grid, the data volume is high and so the performance and flexibility of the database are important properties for real-time monitoring and demand-side management in a smart grid [31, 32]. The web application energy campus communicates over web socket technology with clients and allows the user to view and analyze the data of the local smart grid via browser.

### 3.2 Architecture B

The amount of projects accessing the energy analyzer over Modbus protocol increased. At the beginning of the project, two access operations were planned for each device. However, now there are more accesses and the Siemens devices cannot handle more than two connections at the same time. This resulted in access contentions and data loss in all systems and therefore we modified the architecture as shown in Fig. 2. In this architecture, the data from energy analyzers are collected from the GridVis Software and consequently, there is only one access operation. The energy analyzers are in a separate IP network and only GridVis is able to establish a connection to the devices. Outside the network, a test network exists, which is accessible from all systems for testing different energy analyzers and software applications. All other stakeholders receive the grid data via the RESTful interface of GridVis. For this purpose, we implemented the additional software application GridVisFetch.

GridVisFetch sends data requests to the RESTful interface of GridVis every second and every 15 min. The data are saved in the Apache Cassandra database as described above. From this database the Powermanager receives the data through implemented virtual devices, which simulate the energy analyzers. The MS has the total control of the most devices and data, but through this architecture, all stakeholders receive measurements for research and the device access operations are minimized. In addition, all heat and water analyzers, which are integrated in the software GridVis, are also available for other stakeholders.

### 3.3 Web application

The web application featured many views, charts, and a device and user management. It is the interface between user and power grid data for visualization and analysis. In the following, the web application is described in detail.

#### *3.3.1 Technologies*

The web page is written in HTML5 and JQuery. The application visualizes the data with the JQuery libraries Highcharts and Highstock [33]. As middleware, we implemented a server application deployed on an Apache Tomcat 8 Server. This software communicates with the database and calculates some measurements. Clients communicate over bidirectional web sockets with the server. We use the web socket technology for real-time monitoring and sending push notifications from the

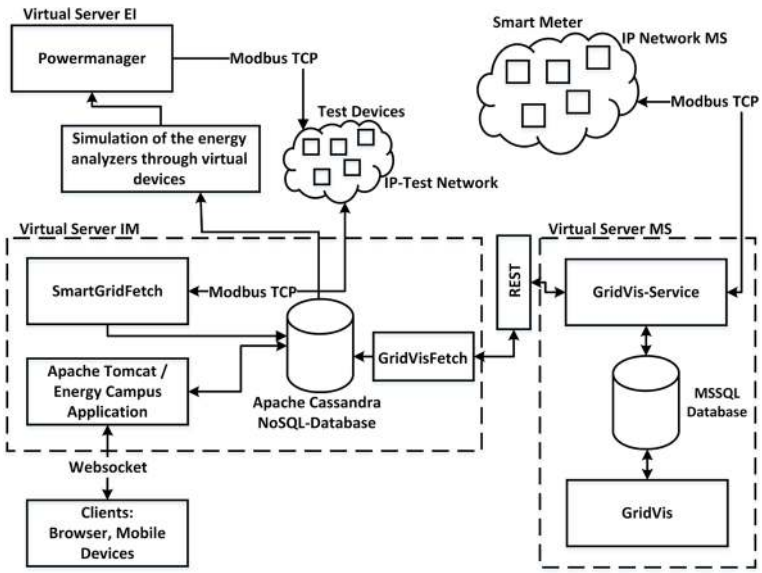


**Fig. 2** Figure shows an architecture without concurrent device access operations

server to the clients. Through the bidirectional communication, the server can react to warning events etc. and forward them to the clients [34, 35].

#### *3.3.2 Data visualization and analysis*

The web application consists of the following views: dashboard, analyze, map, and management. The dashboard shows real time data of active energy consumption and power generation through the photovoltaic plant as shown in Fig. 3. From left to right, the charts at the top show the total amount of the current active energy consumption of the OTH, the daily total amount of active energy consumption, and the current produced energy through the photovoltaic plants. The installed energy analyzers are grouped in three main areas and the charts at the bottom line show the total amount of current active energy consumption from these three areas. All values are refreshed every second, so it provides an up-to-date

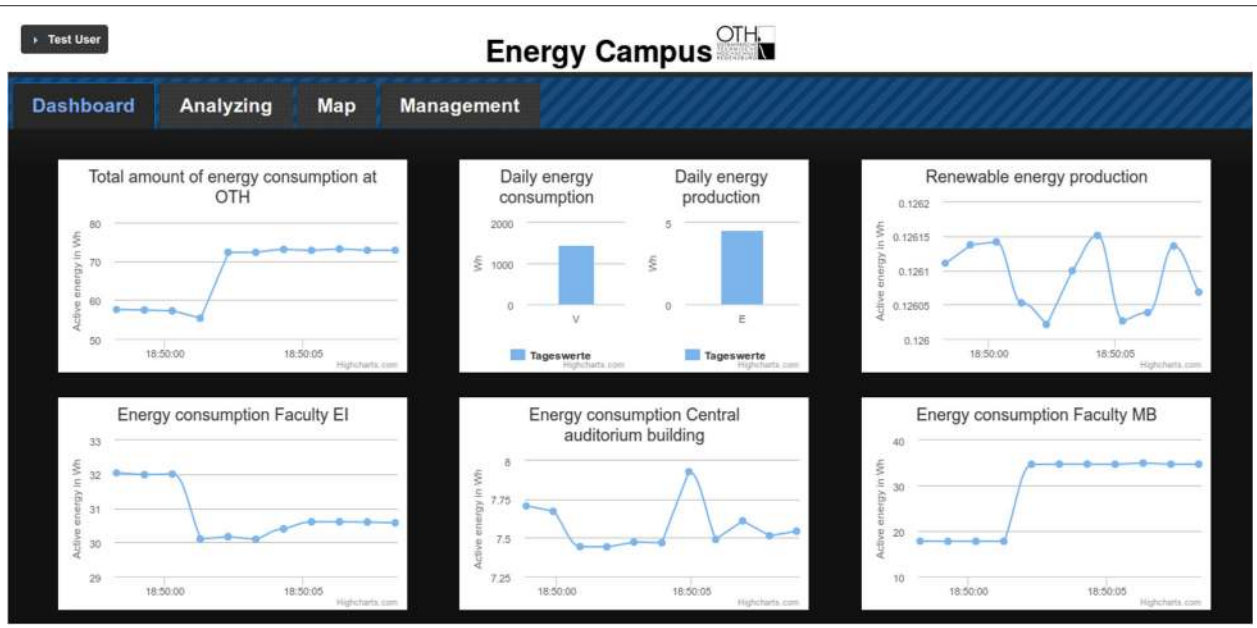


**Fig. 3** Dashboard of energy campus application. The dashboard shows current values of the power grid. The data were refreshed every second

overview. While the dashboard shows current data from grouped energy analyzers, the map view supports data visualization from single devices and for any time period. It displays the topology of the local smart grid on an interactive map. There, the user sees all energy analyzers marked at the map, so he is able to choose a device and request data from each device. The measured data are aggregated to 1-, 5-, and 15-min values, so it is possible to choose the level of aggregation. The response data are viewed in an interactive line chart with zoom in and out functionality.

At the analyze tab, users are able to create charts to study data and to identify load peaks. Comparisons of different measurements and devices are possible. For example, one chart shows the amount of available data at database, in which the available data were put in relation to the amount of expected data. Different types of load peak charts are supported, which make it possible to identify load peaks from single devices or compare load peaks of several devices in one chart. In addition, there are charts simplifying the identification of regularities in the timing of load peak events. Devices are installed at the power grid, are viewed in a tree structure, which is constructed as the network plan. Charts were generated by selecting devices on the tree structure and choosing a chart type. The charts a user generated during his session were saved at database and loaded automatically at the next login. So users can continue their work at a later time without requesting all data again.

The last tab of the web application is a simple user and device management. To protect the sensitive data against unauthorized access and manipulation of device parameters, the users of the web page are limited and the access is password protected. In addition, the web page is only accessible within the network of OTH.

## 4 Architecture analysis and results

According to the use cases of our application and future works in energy and power management, we analyzed network traffic of two scenarios based on the architectures described above. To realize real-time monitoring, load shifting, and future research projects, it is necessary to fetch data from energy analyzers every second. In this section, we present the results of network analysis of both architectures with focus on latency, amount of transferred data, and needed bandwidth of second-by-second fetched data.

### 4.1 Scenario A

Scenario A is based on the initial system architecture shown in Fig. 1 and described above. We use the Modbus TCP/IP protocol, described in Section 4.1.2, to request data directly from energy analyzers. Afterwards, we describe the test environment we used, the tests and their results. Test results are compared with theoretical analysis.

#### *4.1.1 Test environment*

*Server*: As a server, the virtual machine of EI with Windows Server 2008 R2 Enterprise 64 Bit operating system is used. The virtual machine has two Intel(R) Xeon(R) X5650 2.67 GHz processors and is integrated in the IP network via a 10 GBit/s Ethernet interface.

*Energy analyzer*: In the tests, we used 27 physical devices, which are integrated in the network over 10/100 MBit/s Ethernet. Because of hardware problems, some energy analyzers were unavailable at time of testing.

*Software*: The Java software application SmartGridFetch collects data over Modbus TCP/IP from the energy analyzers. For each device, the application starts a thread which requests 18 measurements at the same time every second. For each measurement, we request 2 registers with 2 bytes over the Read Holding Register function and so each measurement has a size of 4 bytes.

#### *4.1.2 Modbus over TCP/IP protocol analysis*

The Modbus TCP/IP protocol is a request/reply protocol and consists of an application data unit (ADU) and a protocol data unit (PDU). The data are stored in 2-byte registers. To request data from a device over Modbus TCP/IP, we use the Read Holding Register function and read contiguous blocks of registers. The maximum size of a Modbus protocol data unit is 253 bytes. In the request, 3 bytes are used by a function code (1 byte) and the starting register address (2 bytes). Two bytes are used for the amount of registers to read, so the maximum is 125 registers. The response consists of 1 byte for the function code, 1 byte for count, and the remaining for the values we read out [36]. The Modbus communication over TCP/IP is shown in Fig. 4. Between opening and closing the communication, several request-response blocks are possible. Each block consists of one Read Holding Register request, followed by an acknowledgement (ACK) and response data, and finished through a further ACK. For analyzing latency, amount of transferred data, and needed bandwidth of second-by-second fetched data, we focus on the request-response blocks, which cause the most data. Figure 5 shows the request and response packets transported over the network. We assume that no additional options are set at the IP and TCP header, so we calculate with the minimal header size. One packet consists of an Ethernet (14 bytes), IP (20 bytes), and TCP (20 bytes) header, which encapsulate the Modbus ADU ($8 + x$ bytes), followed by the Ethernet checksum (4 bytes). Therefore, each packet has a size of $14 + 20 + 20 + 8 + x + 4 = 66 + x$ bytes. For the Read-Holding-Register request, the packet size with $x = 4$ bytes is 70 bytes. The response is depending on the amount of requested values $N$. With

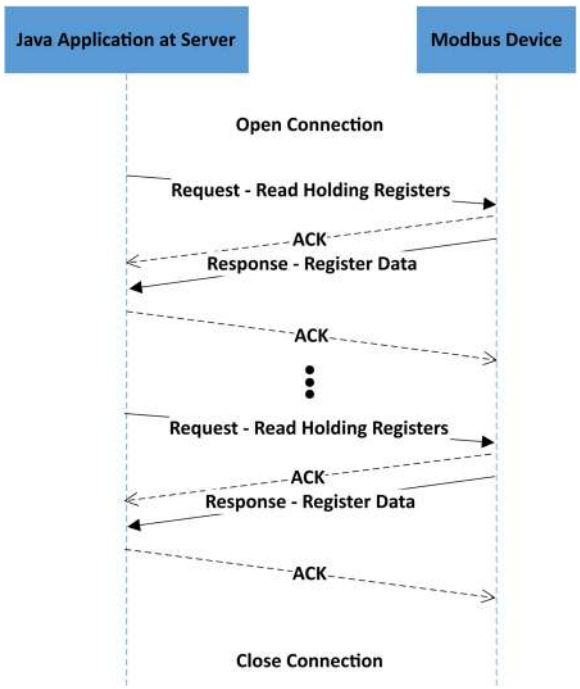


**Fig. 4** Modbus TCP/IP communication

$x = 1 + 2 \cdot N$ bytes, the response packet has a size of $67 + 2 \cdot N$ bytes. The ACK is an empty TCP packet with some activated option flags and has a size of 58 bytes for header overhead + 6 padding bytes, due to the fact, that every Ethernet frame musts be at least 64 bytes. On the basis of these values, we have a request-response block size of $70 + 2 \cdot 64 + 67 + 2 \cdot N = 265 + 2 \cdot N$ bytes, which is transferred each second for one Modbus device.

In an ideal network, without concurrent network traffic, application, and other overhead, latency and size of transferred packets depending on amount of requested registers $N$ is shown in Fig. 6. With each register, the latency grows in steps of 0.00016 ms. The latency for the maximum possible amount of registers in one request is 0.0412 ms for 62 measurements (515 bytes packet size). As described above, we request 18 measurements for each energy analyzer. For 18 measurements (36 registers), the request-response block has a size of 337 bytes and a latency of 0.02696 ms. Table 1 shows the values for up to 5 devices as example. Each device requests 18 measurements per second. For 5 devices, 2575 bytes/s are transferred over the network in 0.206 ms. Calculated to 100 devices with a total amount of 33700 bytes, the latency is 2.696 ms and the needed bandwidth is 0.2696 MBit/s.

#### *4.1.3 Tests and results*

In our test, we send a request to one device and read out 18 measurements. Then, we increment the amount of devices and respectively the amount of threads. We measure the time between sending the request and getting the response data. With Wireshark the transferred traffic is captured and shown in Table 2. The occupied bandwidth and all other values increase linearly and they are proportionally to the amount of requested data. If we calculate these values to 100 devices, we get a bandwidth of 0.3 MBit/s and a bandwidth of 3 MBit/s for 1000 devices. This means that the bandwidth of 100 MBit/s is enough for a big infrastructure of energy analyzers. The last row shows the average latency of request/response period. This value is not proportional to the amount of devices and transferred data. The latency increases slowly and a further test with all 27 devices results in an average latency of 68.58 ms. In this case, we request $27 \cdot 18 = 486$ measurements in less than 100 ms. For our scenario, this is sufficient. If we assume that the time increases linearly with a value of 3 ms for each additional device as an upper limit, we get also fast transferred periods for this scenario. In this theoretical case, we get an average latency time of 328.08 ms for 100 devices (1800 measurements). In comparison with the values in an ideal network, as described above, the latency is very high and fluctuating. Due to

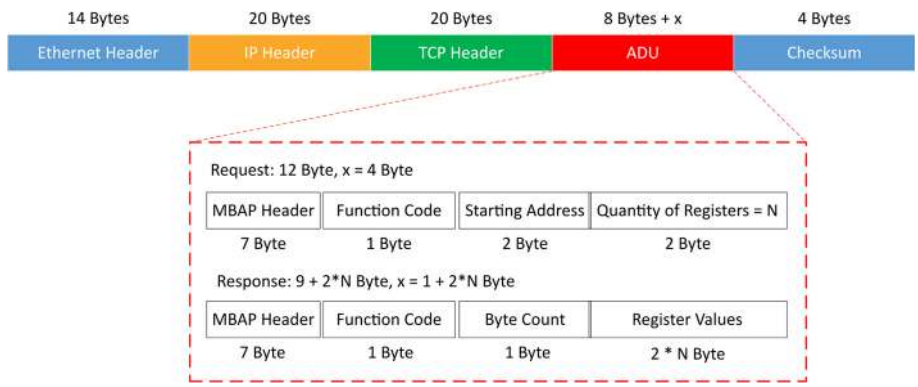


**Fig. 5** Modbus request/response packet size

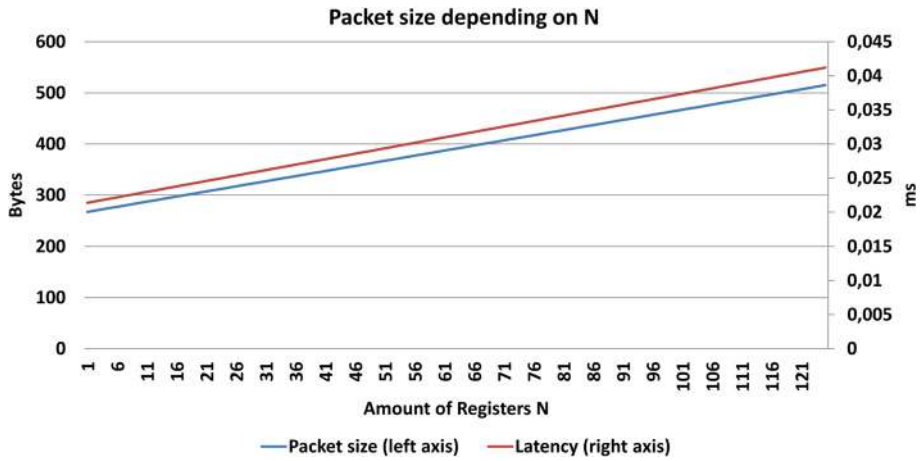


**Fig. 6** Modbus TCP/IP—latency in ideally network environment

the fact that the test environment is in a big network infrastructure at OTH Regensburg with many users and machines in the network and addition time delay through the Java test application and operations on the used server machine, the latency differs in a big range from the theoretical calculated values. However, the test gives a good impression for scaling in comparable environments.

### 4.2 Scenario B

Scenario B is based on the system architecture B shown in Fig. 2 and described above. Here, we use a RESTful web service of commercial software for collecting data from energy analyzers. Due to the limits of web services by requesting a large amount of data [24], among other things, we test how many data are possible to request over the web service before the request run into a FULL HEAD error at HTTP header.

#### *4.2.1 Test environment*

*Server and hardware*: We use the same amount of physical energy analyzers as in scenario A. The GridVis software is running on the server of EI and saves the grid data in a MYSQL database on the same server. The test application, which requests data over the RESTful web service, is running on the Server of IM.

*Software*: In this architecture, the software application GridVis from Janitza collects data from power network. GridVis also requests data over the Modbus TCP/IP protocol from energy analyzers. Every second, it requests data in the background and makes them available at a RESTful web service. The software aggregates the measured data to configurable values and save them in a MySQL 5.7.4.0 database. In order to do this, it is necessary to use an additional application called GridVis Service [37]. In the test environment, we installed GridVis 6.0.2-64 Bit and GridVis Service 6.0.2-64 Bit on the server. We implement small Java applications which send requests to the web service and measure the latency (time from sending the request to receiving the response data completely) and the size of the request data in the HTTP package. In addition, we use the application Wireshark to capture network traffic during tests.

#### *4.2.2 GridVis RESTful web service analysis*

The RESTful web service of GridVis provides several HTTP GET Requests to request data, gathered from the Modbus devices by GridVis. It is possible to request aggregated values for various time periods or to request so called "online" values, which are the last current values, fetched every second. The GET URL for a request consists of some project parameters and the request parameters

**Table 1** Calculated traffic from Modbus TCP/IP connections

| Devices | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Bytes | 337 | 674 | 1011 | 1348 | 1685 |
| MBit/s | 0.002696 | 0.005392 | 0.008088 | 0.010784 | 0.01348 |
| Latency in ms | 0.02696 | 0.05392 | 0.08088 | 0.10784 | 0.1348 |

**Table 2** Captured traffic from Modbus TCP/IP connections

| Devices | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Packets | 241 | 480 | 720 | 960 | 1200 |
| Avg Packets/sec | 4.02 | 8.07 | 12.01 | 16.03 | 20 |
| Avg Packetsize | 78.68 | 78.75 | 78.75 | 78.75 | 79 |
| Bytes | 18966 | 37800 | 56700 | 75600 | 94500 |
| Avg Bytes/s | 316.07 | 635.19 | 945.69 | 1262.33 | 1578.07 |
| Avg MBit/s | 0.003 | 0.005 | 0.008 | 0.01 | 0.013 |
| Avg Latency in ms | 28.08 | 30.5 | 31.9 | 35.3 | 38.5 |

such as device number and measurement identifier. For each measurement type, the device number and measurement identifier are appended to the URL. Due to the fact that the measurement identifiers are mapped to different string values instead of simple small numbers, the URL size fluctuates and increases rapidly depending on the requested measurements. To give a statement about expected packet size and latency as in Section 4.1.2, we analyzed the URL and parameter structure and approximate the packet size, depending on the requested measurements, in a mathematical function. Figure 7 shows the whole transport frame for HTTP request and response and the approximate function. Each transport packet consists of an Ethernet, IP, and TCP Header and an Ethernet checksum, as described above at the Modbus TCP/IP transport packet. The HTTP part of the packet is split in a request and response and both consists of a header and a body. At the HTTP Request, the HTTP body is zero. As described in the HTTP 1.1 specification [38], the header can consist of several header fields (depending on server) and the header size is not limited through specification. Therefore, we analyzed the Wireshark recordings of the communication between a Java application and the GridVis web service to identify the used header fields. The recordings show that the header consists of 157 bytes for several header fields and the Request URL. With the assumption that for each measurement all three phase values will be requested, we approximate the URL size for our work with the following equation: $x = 45 + 28 \cdot N/3$ bytes. The URL has a fixed part, which includes server and project parameters (45 bytes) and the variable part of the URL consists of the requested measurements as described above. On basis of this, the HTTP request has a size of $157 + 45 + 28 \cdot N/3 = 202 + 28 \cdot N/3$ bytes and the whole transport packet has a size of $260 + 28 \cdot N/3$ bytes, with $N$ is the amount of requested measurements.

At the response, the HTTP header has a fixed size of 127 bytes and the HTTP body is also depending on the amount of requested $N$ measurements. The response contains the data in JavaScript Object Notation (JSON) format. For each requested measurement, it includes the measured value and a timestamp. For both values it includes the device number and measurement identifier. Therefore, the size of the HTTP body increases as shown in the following approximated equation: $x = 40 + N \cdot 31 + N \cdot 38$ bytes with 40-byte JSON overhead, $N \cdot 31$ bytes for the measurements, and $N \cdot 38$ bytes for the timestamps. So, the HTTP response has a size of $167 + N \cdot 31 + N \cdot 38$ bytes. With Ethernet, IP Header, etc., the whole packet size is $225 + N \cdot 31 + N \cdot 38$ bytes. For a full request-response block, the size is $485 + 28 \cdot N/3 + 69 \cdot N$ bytes. In the next section, we compare these values with the real test results.

#### 4.2.3 Tests and results

First, we start testing to request data every second. There, we create the request URL to get 18 measurements from one device in one request. In every test, we increase the amount of measurements by adding one device with 18 measurements to the Request URL. We get six values for voltage, three values (one for each phase) for effective power, reactive power, apparent power, and current. Further measurements and devices were requested by expanding the URL. Due to the fact that just 27 energy analyzers are available, we simulated the additional devices by adding more measurements to the available devices. In this case, 18 measurements represent one device. We test from 1 up to 35 devices. At 35 devices,

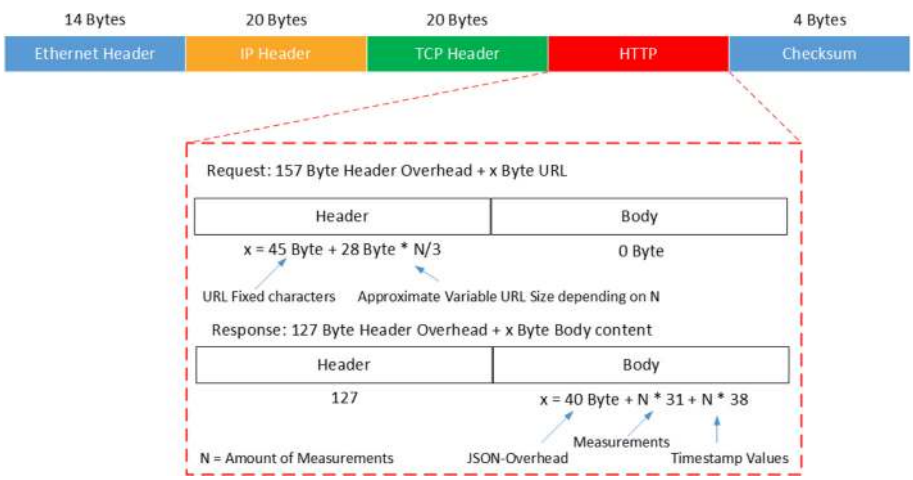


**Fig. 7** HTTP request/response packet size

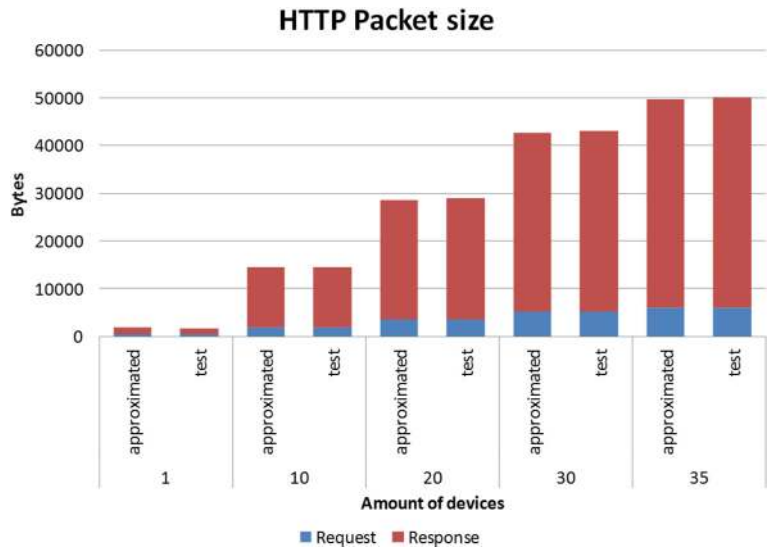


**Fig. 8** Packet size—without overhead from underlying protocols

we get $35 \cdot 18 = 630$ measurements in one request. Every additional measurement causes the expected full head status code at the HTTP header so we receive no data, because the Jetty Server application of the web service limits the Request URL. Figure 8 shows the request and response packet size in comparison with the approximated packet size. The approximated packet size is calculated with the mathematical function described above. The figure shows that the approximated packet size just changes in a small range from the real tested values. Figure 9 shows the real transferred data, captured with Wireshark, and compares the real packet size with the approximated packet size. It shows that the real packet size is larger than the approximated size. In the approximated model, we did not consider ACK messages in the communication, so the size of real tested data is different. Additionally, we did not consider that big packets could be fragmented, so the overhead from underlying protocol stacks increases. Figure 10 shows the average of the needed bandwidth in MBit/s and Fig. 11 presents the average latency in milliseconds between sending a request and receiving the data of the corresponding response completely. The test is over a period of 60 s as shown at the $x$-axis. The lines show the average latency in milliseconds at the primary (left) $y$-axis for the different amount of devices. The dots present the maximum measured latency in the test period of 60 s at the secondary (right) $y$-axis in milliseconds for the different amount of devices. The blue

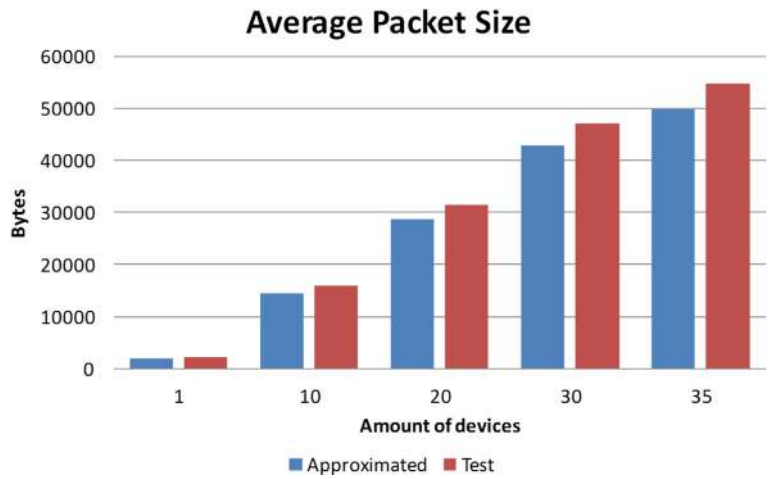


**Fig. 9** Packet size—with overhead from underlying protocols

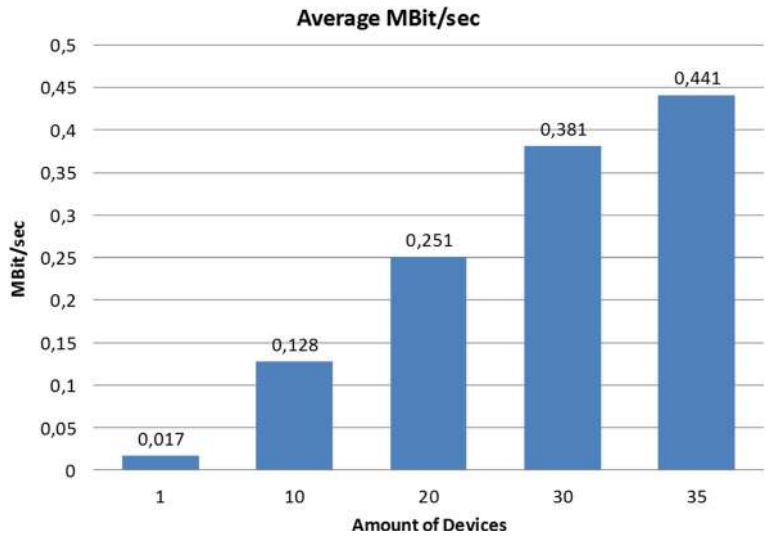


**Fig. 10** Average MBit/s

dot shows the maximum measured latency for one device, the red one for 10 devices, and so on. The average latency is low for 1, 10, and 20 devices. For 30 and 35 devices, the average latency is also less than a second but there, the maximum measured latency is above 1 s. In the case of 35 devices, we get latency above 1 s twice. This means that in a period of 60 s, we receive 58 response packets with measurements instead of 60. The results show that the average latency is not increasing proportionally with the transferred data size, which is almost linearly to the number of requested measurements. In our case, we need measurements in an interval of 1 s for real-time monitoring, future energy, and power management and for the analysis in EI. The measured latency, with outliers shown above, is suitable for these use cases. Additionally, the measurements were aggregated to 1-min values, so a data density of 96.67 % is acceptable, because the measurements of the power network do not differ in a big range from 1 s to the next. Figure 12 shows the average latency in comparison with the approximated latency. As such, as we described for the Modbus protocol, the latency differs in a big range from the approximated latency through high network load, application time delay, and utilization rate of the used server machine.

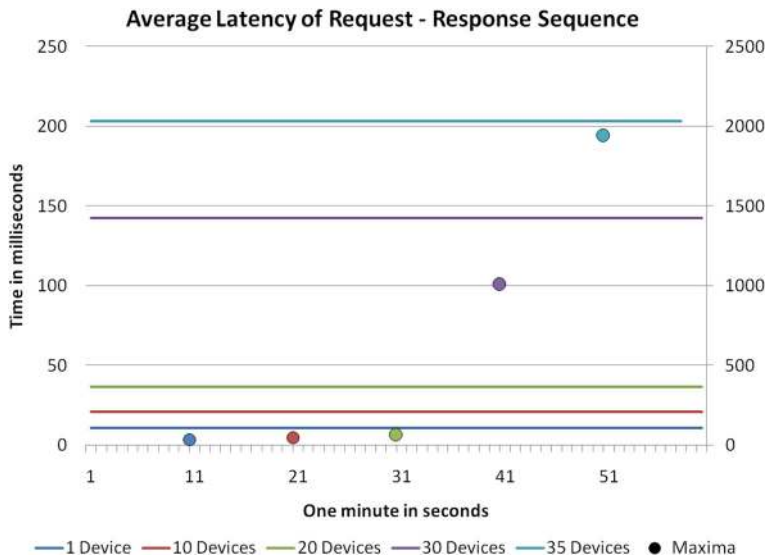


**Fig. 11** Average latency

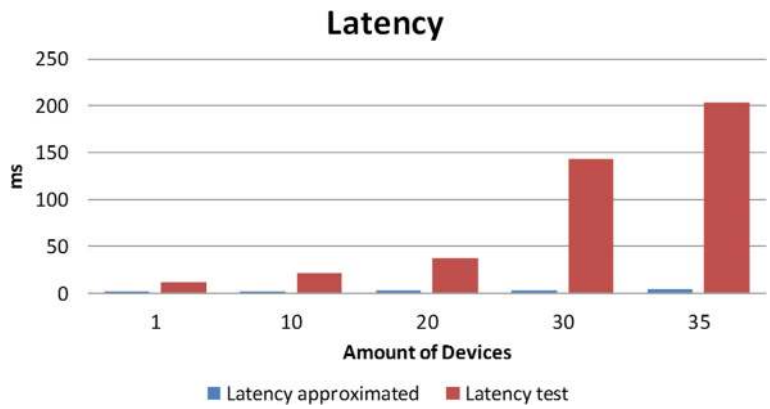


**Fig. 12** Comparison of average latency

The tests show that time and needed bandwidth are not the limiting factors but the amount of requested measurements. It is possible to request 630 measurements, which is equivalent to 35 devices with 18 measurements, at one request in less than a second. For our power network infrastructure, this is a suitable value. Through splitting the amount of requested devices and running several threads, we can increase the amount of devices. Each thread requests 10 devices, respectively 180 measurements. We run this test with 10, 20, 30, and 40 threads. Figure 13 shows the average of needed bandwidth in MBit/s. Here, the latency for 10 and 20 threads is also in our range of 1 s. The bandwidth for 1, 10, and 20 threads, respectively, 10, 100, and 200 devices, which increases linearly, is acceptable. At the test with 30 and 40 threads, the REST interface is operating to full capacity and the CPU capacity of the server is 100 %. Therefore, we do not receive all data we requested.

## 5 Conclusions

This paper presents a software application for analyzing and monitoring real-time data of a smart grid. This application forms the basis of future projects with focus on load shifting and peak clipping. We describe the web application and two possible architectures for collecting data in order to avoid concurrent access operations on energy analyzers. On the basis of the described architectures, we carry out performance tests for each architecture. The tests show that both architectures are currently useable in a small energy landscape and give a good impression for scaling in comparable environments. The Modbus TCP/IP protocol is a fast communication protocol for this use case. The solution with the GridVis software where we get data over the REST interface is useable for a small amount of energy analyzers. In contrast, if the energy landscape is expanded, the REST interface of this software runs into its limits.

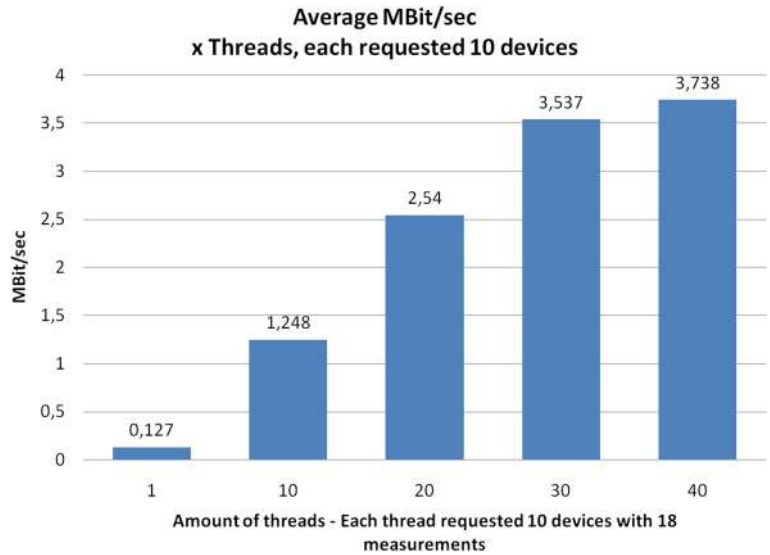


**Fig. 13** Average MBit/s—threads

**Acknowledgements**
This work was supported by the Regensburg Center of Energy and Resources (RCER) and the Technology and Science Network Oberpfalz (TWO). Further information is under www.rcer.de.

**Competing interests**
The authors declare that they have no competing interests.

Received: 29 February 2016 Accepted: 23 July 2016
Published online: 12 August 2016

**References**

1. M Hashmi, S Hänninen, K Mäki, in *Innovative Smart Grid Technologies (ISGT Latin America), 2011 IEEE PES Conference On*. Survey of smart grid concepts, architectures, and technological demonstrations worldwide (IEEE, 2011), pp. 1–7
2. Y Yan, Y Qian, H Sharif, D Tipper, A survey on smart grid communication infrastructures: Motivations, requirements and challenges. IEEE Commun. Surv. Tutor. **15**(1), 5–20 (2013)
3. J Gao, Y Xiao, J Liu, W Liang, CP Chen, A survey of communication/networking in smart grids. Futur. Gener. Comput. Syst. **28**(2), 391–404 (2012)
4. E Tronci, T Mancini, F Mari, I Melatti, RH Jacobsen, E Ebeid, SA Mikkelsen, M Prodanovic, JK Gruber, B Hayes, in *Proceedings of the Work in Progress Session (euromicro Dsd/seaa 2014)*. SmartHG: energy demand aware open services for smart grid intelligent automation, (2014)
5. V Alimguzhin, F Mari, I Melatti, E Tronci, E Ebeid, SA Mikkelsen, R Hylsberg Jacobsen, JK Gruber, B Hayes, F Huerta, et al., in *Digital System Design (DSD), 2015 Euromicro Conference On*. A glimpse of SmartHG project test-bed and communication infrastructure (IEEE, 2015), pp. 225–232
6. F Pallonetto, E Mangina, D Finn, F Wang, A Wang, in *Proceedings of the 1st ACM Conference on Embedded Systems for Energy-Efficient Buildings*. A restful API to control a energy plus smart grid-ready residential building: demo abstract (ACM, 2014), pp. 180–181
7. RF Eggea, M Ferreira, AR Aoki, RJ Riella, in *Innovative Smart Grid Technologies Latin America (ISGT LATAM), 2015 IEEE PES*. Energy management including photovoltaic panel and energy storage for smart grids through mobile application (IEEE, 2015), pp. 177–181
8. S Chakraborty, MD Weiss, MG Simoes, Distributed intelligent energy management system for a single-phase high-frequency AC microgrid. IEEE Trans. Ind. Electron. **54**(1), 97–109 (2007)
9. P Palensky, D Dietrich, Demand side management: demand response, intelligent energy systems, and smart loads. IEEE Trans. Ind. Inform. **7**(3), 381–388 (2011)
10. BP Esther, KS Kumar, A survey on residential demand side management architecture, approaches, optimization models and methods. Renew. Sust. Energ. Rev. **59**, 342–351 (2016)
11. P Siano, Demand response and smart grids—a survey. Renew. Sust. Energ. Rev. **30**, 461–478 (2014)
12. J Ekanayake, N Jenkins, K Liyanage, J Wu, A Yokoyama, *Smart Grid: Technology and Applications*. (Wiley, United Kingdom, 2012)
13. R Palma-Behnke, C Benavides, F Lanas, B Severino, L Reyes, J Llanos, D Sáez, A microgrid energy management system based on the rolling horizon strategy. IEEE Trans. Smart Grid. **4**(2), 996–1006 (2013)
14. S Kenner, R Thaler, M Kucera, K Volbert, T Waas, in *Intelligent Solutions in Embedded Systems (WISES), 2015 12th International Workshop On*. Smart grid architecture for monitoring and analyzing, including Modbus and rest performance comparison (IEEE, 2015), pp. 91–96
15. G Zenger, S Kenner, K Volbert, T Waas, M Kucera, in *Intelligent Solutions in Embedded Systems (WISES), 2013 Proceedings of the 11th Workshop On*. Acquiring energy data from a medium-voltage grid for future smart grid solutions: a practical smart grid application example realized by use of cellular communication networks of the 2 nd and 3 rd generation (IEEE, 2013), pp. 1–8
16. P Schlegl, P Robatzek, M Kucera, K Volbert, T Waas, in *Second International Conference on Advances in Computing, Communication and Information Technology (CCIT'14)*. Performance analysis of mobile radio for automatic control in smart grids (Seek Digital Library, 2014), pp. 135–141
17. S Akshay Uttama Nambi, M Vasirani, RV Prasad, K Aberer, in *Innovative Smart Grid Technologies Conference Europe (ISGT-Europe), 2014 IEEE PES*. Performance analysis of data processing architectures for the smart grid (IEEE, 2014), pp. 1–6
18. MM Rahman, M Kuzlu, M Pipattanasomporn, S Rahman, in *Innovative Smart Grid Technologies Conference (ISGT), 2014 IEEE PES*. Architecture of web services interface for a home energy management system (IEEE, 2014), pp. 1–5
19. E Joelianto, et al, in *Wireless and Optical Communications Networks, 2008. WOCN'08. 5th IFIP International Conference On*. Performance of an industrial data communication protocol on ethernet network (IEEE, 2008), pp. 1–5
20. Q Liu, Y Li, in *Intelligent Control and Automation, 2006. WCICA 2006. The Sixth World Congress On*. Modbus/tcp based network control system for water process in the firepower plant, vol. 1 (IEEE, 2006), pp. 432–435
21. B Kim, D Lee, T Choi, in *TENCON 2015-2015 IEEE Region 10 Conference*. Performance evaluation for Modbus/TCP using network simulator NS3 (IEEE, 2015), pp. 1–5
22. JL Fernandes, IC Lopes, JJ Rodrigues, S Ullah, in *Ubiquitous and Future Networks (ICUFN), 2013 Fifth International Conference On*. Performance evaluation of restful web services and AMQP protocol (IEEE, 2013), pp. 810–815
23. H Hamad, M Saad, R Abed, Performance evaluation of restful web services for mobile devices. Int. Arab J. e-Technol. **1**(3), 72–78 (2010)
24. Z Shelby, Embedded web services. IEEE Wirel. Commun. **17**(6), 52–57 (2010)
25. D Srinivasan, T Reindl, et al, in *Smart Grid Technologies-Asia (ISGT ASIA), 2015 IEEE Innovative*. Real-time display of data from a smart-grid on geographical map using a GIS tool and its role in optimization of game theory (IEEE, 2015), pp. 1–6
26. SA Kim, D Shin, Y Choe, T Seibert, SP Walz, Integrated energy monitoring and visualization system for smart green city development: designing a spatial information integrated energy monitoring model in the context of massive data management on a web based platform. Autom. Constr. **22**, 51–59 (2012)
27. RS Brewer, PM Johnson, in *Smart Grid Communications (SmartGridComm), 2010 First IEEE International Conference On*. Wattdepot: an open source software ecosystem for enterprise-scale energy data collection, storage, analysis, and visualization (IEEE, 2010), pp. 91–95
28. Manual: SENTRON PAC4200. https://support.industry.siemens.com/cs/document/34261595/systemhandbuch-sentron-multifunktionsmessger%C3%A4t-sentron-pac4200?dti=0&pnid=19739&lc=de-DE, Accessed 12 Apr 2015
29. Manual:UMG96RM. https://wiki.janitza.de/display/GRIDVIS50EN. Accessed 12 Apr 2015
30. Apache Cassandra 2.0 - Documentation. http://docs.datastax.com/en/cassandra/2.0/pdf/cassandra20.pdf. Accessed 27 May 2016
31. M Arenas-Martìnez, S Herrero-Lopez, A Sanchez, JR Williams, P Roth, P Hofmann, A Zeier, in *Smart Grid Communications (SmartGridComm), 2010 First IEEE International Conference On*. A comparative study of data storage and processing architectures for the smart grid (IEEE, 2010), pp. 285–290
32. A Nayak, A Poriya, D Poojary, Type of NOSQL databases and its comparison with relational databases. Int. J. Appl. Inf. Syst. **5**(4), 16–19 (2013)
33. Highcharts, Highstock Libraries. http://www.highcharts.com. Accessed 19 Feb 2016
34. V Pimentel, BG Nickerson, Communicating and displaying real-time data with websocket. IEEE Internet Comput. **16**(4), 45–53 (2012)
35. A Wessels, M Purvis, J Jackson, S Rahman, in *Information Technology: New Generations (ITNG), 2011 Eighth International Conference On*. Remote data visualization through websockets (IEEE, 2011), pp. 1050–1051
36. Modbus Application Protocol Specification V1 1b3. http://www.modbus.org. Accessed 14 Apr 2015
37. GridVis-Documentation. https://wiki.janitza.de/display/GRIDVIS50EN/GridVisDocumentation+5.0. Accessed 12 Apr 2015
38. R Fielding, J Gettys, J Mogul, H Frystyk, L Masinter, P Leach, T Berners-Lee, Rfc 2616, hypertext transfer protocol-HTTP/1.1 (1999). https://www.ietf.org/rfc/rfc2616.txt. Accessed 01 Jul 2016