Assessment of High-performance Smart Metering for the Web Service Enabled Smart Grid

Stamatis Karnouskos, Per Goncalves da Silva, and Dejan Ilic
SAP Research
Vincenz-Priessnitz-str 1, D-76131, Karlsruhe, Germany
{stamatis.karnouskos,per.goncalves.da.silva,dejan.ilic}@sap.com

ABSTRACT

The electricity network is undergoing a significant change towards a more adaptive, intelligent, self-managing, collaborative and information-driven grid. According to the smart grid vision, any electronic device connected to it will be able to communicate its consumed or produced energy almost in real time. Based on the analysis of this newly acquired information, a new generation of services and decision support systems can be realized, enabling more intelligent decisions, and ultimately a more efficient energy system. Therefore, high-performance acquisition of smart metering information from large scale distributed infrastructures is of key importance for the upcoming Internet-based enterprise services and mash-up applications. We have used open source software to build a web service-based advanced metering infrastructure of simulated smart meters, concentrators, and a smart metering platform, all interconnected via web services. We measure in a methodological fashion the performance of the various components of the architecture and evaluate their limitations. Finally we identify key performance indicators that need to be considered when deploying large-scale smart metering systems, and discuss on challenges and directions that arise.

Categories and Subject Descriptors

H.0 [**Information Systems**]: General

General Terms

Measurement, Performance

Keywords

web services, smart metering, smart grid, performance evaluation

ICPE'11, March 14–16, 2011, Karlsruhe, Germany.

1. MOTIVATION

The smart grid [3, 10] is an emerging concept targeting to provide the next-generation electricity network which will boast capabilities such as self management, resilience, sophisticated services, and almost real-time monitoring and control. It is expected to be the key part in a global ecosystem of interacting entities, whose cooperation will give birth to innovative cross-domain services [7]. Key driving forces behind these efforts are the need for higher energy efficiency and better management of available resources in the electricity grid. To achieve these objectives, fine-grained monitoring of energy inside the grid (such as consumption and production) is essential.

Recent market statements for the smart-grid era, even considered with a grain of salt, provide some hints on expected growths and business significance: Marie Hattar, vice president of marketing in Cisco's network systems solutions group, estimated in 2009 that the smart-grid network will be "100 or 1000 times larger than the Internet". Similarly Vishal Sikka, CTO of SAP, stated in 2009 that "The next billion SAP users will be smart meters".

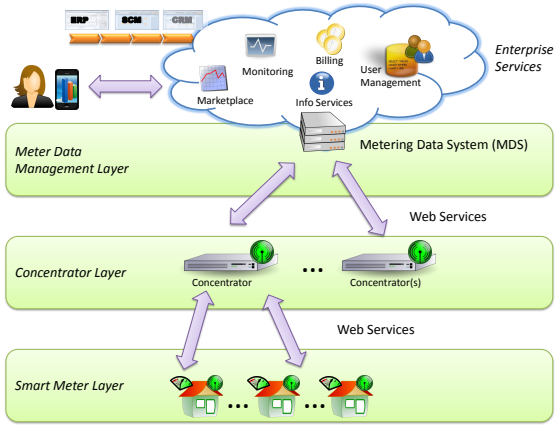

Figure 1: AMI overview in the smart grid era.

An Advanced Metering Infrastructure (AMI) needs to be in place in order to measure, collect, and analyse energy via the usage of meters for electricity, gas, heat, water, etc. As depicted in 1, metering data collected from various sources is gathered at strategically positioned concentrators. The concentrator is the interface between many low-speed, heterogeneous, usually asynchronous, channels and one or more high-speed, usually synchronous, channels. It acts as an in-

terface between the smart meters and the enterprise. It is also responsible for aggregating the meter reading data and submitting it to the metering server.

As AMI is expected to be highly heterogeneous and widely distributed. Various communication technologies will be used in order to acquire the metered data, either on-demand or on predefined schedules. Today's metering resolution, in world-wide smart-grid efforts, is in a time interval of 15 minutes. However, it is expected that it will move towards near-real-time metering, or in other words "high-resolution metering". The last will push further the requirements towards high-performance distributed platforms tailored to the specifics of smart-grids e.g. metering.

The Internet of Things envisions billions of connected devices sensing, possibly collaborating and providing the real world information to the enterprise systems. A common functionality expected, by this vast number of devices, is the capability of any device to provide (in a standardized [2] way) its energy production and consumption e.g. via web services over Internet. This will result in an exponential growth of the number of metered points, leading to an exponential growth in the stress on the respective layer in AMI.

Although today, the killer application for smart metering is billing, this is expected to change. Options for innovative applications and advanced enterprise services will arise together with the "high-resolution metering". Cooperation [6] among the devices, in-network services, and enterprise applications (based on timely assessed real-world data) may have a positive impact to energy efficiency, better resource management, etc. If so, this result might take us one step closer to the smart-grid vision.

1.1 Context of investigation

Although smart metering is a key milestone towards realizing the smart-grid vision, not much evaluation of it is done at a detailed level. This is especially true when referring to a large number of metering points submitting data upstream. Typical evaluations refer to the number of measurements that can be achieved in 15-minute intervals. However, most real-world trials focus only on a few hundred meters.

Our goal is to investigate a use case where high-resolution metering is done over the Internet enabled smart grid [8]. To achieve this, open source software is used for development and investigating metering limitations. As such, our goals and restrictions are:

- to design a simple and scalable approach for large-scale and low-cost smart metering
- to use standardized Internet-based technologies i.e. web services between the AMI layers
- to use existing (off-the-shelf) open source software, and commonly available PCs as the hardware platform
- to simulate large number of metering points (smart meters)
- to evaluate the performance limits of the key AMI components, i.e. at concentrator and metering-server level
- to acquire hands-on experience and insight into large-scale smart metering performance

2. ARCHITECTURE

2.1 Components

A typical set-up in AMI is a three-layered hierarchical architecture, similar to what is depicted in figure 1. Bottom-up we can clearly distinguish:

- Meter Layer: the last mile, where the (residential) meters are passively tapping in and measuring the energy consumption or production of the attached devices.
- Concentrator Layer: the meters connect to this layer via various (often proprietary) protocols to report their measurements. The reported data is aggregated and submitted to the metering data system (MDS).
- Metering Data Management Layer: here usage data and events with respect to the infrastructure are collected for long-term data storage, analysis and management. This is typically used by enterprise services in order to empower applications such as billing, forecasting, etc.

For our approach we adopted the same model, albeit some technological and context constraints were considered. In the context of this work, we assume a fully IP-based three-layered service-oriented infrastructure. This implies that all messaging among the layers is done over web services. Also, in the smart grid context, the components depicted in the layers (such as meters, concentrators, and MDS) are designed to handle high volumes of data at high rates, hence, we expect permanent connections with possibly high bandwidth among them. As one may notice, there is a clear analogy to the Internet which is composed of end-devices, routers, and servers. Similar motivation such as heterogeneity management, scalability, and performance exists in the smart-grid.

According to the Internet of Things vision [4], billions of connected devices will exchange information which will to a drastic increase of metering points as these devices will be able to measure and report their current energy consumption or production. As more real-world enterprise services will depend on them, it is crucial to investigate important performance indicators under the context depicted in the section 1.1.

2.2 Implementation

The implementation of the architectural components attempts to adhere to current industrial trend for interconnection of heterogeneous components, namely service-oriented architecture (SOA) and more specifically the use of web services. In this specific implementation, we have experimented with (widely used) web services that rely on the Simple Object Access Protocol (SOAP). Thus, all components and their functionality were implemented in Java as web services, enabling them to be remotely accessed by any device capable to exchange a SOAP message.

The architecture realization is shown in figure 2. Two web services were implemented: one for the MDS and one for the concentrator, i.e. `MeterReadingService` and `ConcentratorService` respectively. Both services were realized using the Enterprise Java Bean (EJB) 3.0 technology. Consequently, all necessary business logic is encapsulated and executed by the application server (AS). Multiple requests

can be processed in parallel, since many EJBs instances can coexist in the AS.

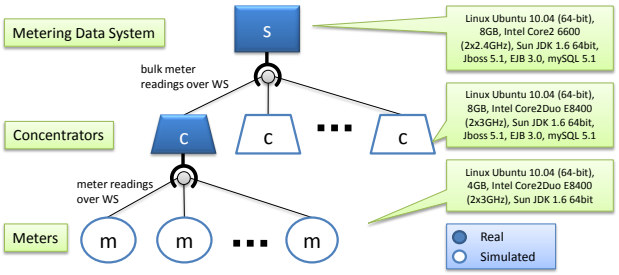

Figure 2: Topology and implementation of the architecture.

By using EJB web service annotations, the beans can accessed remotely using standard HTTP requests. Thus, the meters submit their readings to the EJB bean that represents the `ConcentratorService` web service running on a concentrator device. These `ConcentratorService` EJBs are designed to share a memory buffer of configurable size. Once a reading is received, each bean will store received values inside this buffer. This service is designed to entirely fill in the buffer before triggering the event for upstream transmission of aggregated meter readings i.e. to the MDS. Once the `MeterReadingService` receives a list of readings, it iterates through it, and stores the readings to a local MDS database.

All database tasks are implemented using the Java Persistence API (JPA). The JPA automates table creation and data insertion processes, such as transferring the data between a database and an object (and vice-versa). This implies usage of the Object-Relation Mapping and it is done via EJB class called Entity Beans.

As previously mentioned, our aim was to acquire experiences for the large-scale deployment of the proposed smart metering architecture. However, due to lack of means for real-world deployment at such a large scale, we have created a mixture of real and simulated devices (as depicted in figure 2). Experiments are conducted by deploying the MDS and one concentrator on separate physical machines, while scaling up required simulated meters and multiple concentrators from separated physical machines.

These simulated entities mimic the functionality of the real architectural component by exchanging appropriately formatted messages over the web services. Each simulated entity is designed for high performance, thus it deals with low level programming i.e. `Socket` class from the `java.net` package. Each simulator creates a pool of parallel threads designed to open the TCP connection, transmit appropriately formatted SOAP message and to close the TCP connection. In the case of the meter simulator, each meter reading request is created and sent to the `ConcentratorService` service. Conversely, in the concentrator simulator, bulk reading requests are created and sent to the `MeterReadingService` web service.

The simulators are designed and implemented to accurately simulate a considerable number of nodes from one workstation. This, in turn, enables adequate stressing of the servers in order to determine their individual limits. In other words, the developed application is able to perform more parallel requests than a concentrator or MDS component (running on a physical machine) can process.

As indicated, we have used commonly available hardware and open source software. The configuration of the physical machines is depicted in figure 2 and consists mainly of three workstations running Sun JDK 1.6 64bit, 64-bit Linux Ubuntu 10.04 (with its out-of-the-box configuration). They are equipped with Intel Core2Duo processors and 4-8 GB of RAM each, while interconnected with a gigabit LAN. Presented web services are deployed on the JBoss Application Server (version 5.1.0.GA) where the standard JPA implementation for this AS is called Hibernate. For data storage, the MySQL Server (version 5.1.41) DBMS is used.

3. PERFORMANCE EXPERIMENTS

The performance experiments conducted were aimed at high volumes of metered data from the meters up to the MDS, so that enterprise applications can take advantage of almost-real-time data. In order to achieve our goals, we need detailed measurements of data exchanged between the architectural components as well as in their interworking. The overall performance of the architecture is measured in terms of its capacity to handle certain number of requests per second at different layers. These measurements are taken against the MDS and concentrator components in order to determine their limits, and also their reliability under heavy load.

In total four experiments have been conducted:

I. Assumption Validation

II. Concentrator Performance (multiple meters)

III. Multi-concentrator MDS Performance

IV. System-performance validation

Our methodology is to evaluate each component for its performance and then be able to make an assessment for the whole system. In the first experiment, major assumptions are quantified and validated. In the second experiment, the performance of a single concentrator handling a large variable number of incoming meter readings is measured. In the third experiment, the performance of the MDS is measured against multiple concentrators. Finally, in the fourth experiment, the results and acquired experience of the previous three experiments are gathered, and used to derive a theoretical high-throughput configuration, which is then assessed.

3.1 Metrics, tools and data collection

In order to provide an insight to the context in which the experiments were conducted, we discuss the metrics that were considered, the simulator we developed, and the data-collection approach that was followed.

3.1.1 Metric definition

In the hierarchical structure from figure 2, one can see that concentrators and MDS depend on number of sub-components (i.e. i meters or j concentrators). The key performance indicator common to all components is the *meter*

reading rate r of received meter readings from meters. However, due to the nature of the aggregation of meter readings by the concentrator, the *request rate* q variable is introduced, that depends on the aggregation of messages in bulks of size b done at each sub-component j. Therefore, for each concentrator or MDS, the request rate q is defined as:

$$q = \sum \frac{r}{b} \quad (1)$$

There are n single meter readings being submitted within a time interval t, or request of bulk size b coming from a concentrator. Thus, the rate of meter readings can be defined as:

$$r = \frac{nb}{t} \quad (2)$$

Throughout this work we assume that each meter m_i is submitting one measurement at a time (and not aggregated meter readings), thus $b_i = 1$. This is what we expect in real world applications, at least from the always-connected meters. As a result, the request rate r for a concentrator c is $q^c = r^c$. Similarly, if a single meter reading is also propagated further not as part of a bulk (thus $b = 1$) from the concentrator c to the MDS s, then $q^s = r^s$. Furthermore, assuming minimal impact on the rates (e.g. no losses, no significant processing overhead, etc.), it could be argued that $q^s \approx q^c$.

3.1.2 Meter simulation

As can be seen, in equations 1 and 2, there is a direct relationship between the request rates at each of the components and the number of meters, and, their submission interval and the bulk size b configured at the concentrators level. This leads to the first constraints to be considered, that is, if the processing limit of a component is reached, the system can be configured to respect that limit.

The developed simulator can simulate components (meters or concentrators) that continuously submit meter readings at configurable rates. figure 3 shows the period p and number of i simulated meters. Each thread represents a meter executing one fixed process, that is, open a connection, submit the message and close the connection. Subsequently, it sleeps for an interval p and then repeats its execution process. In total, there are i number of threads equalling the population of meters.

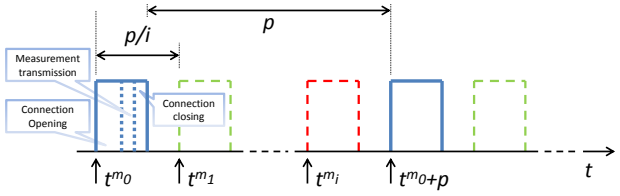

Figure 3: Tuning the request rate with the use of the simulator.

In order to avoid bursts, and have a more uniform distribution of the i meter readings per interval, we initiate each thread in a semi-sequential manner, i.e. every p/i interval. Once the thread is started, it follows its own life cycle. The life cycle ensures that the following loop is executed: (i) a connection is opened, (ii) the data is submitted, (iii) the connection is closed, and (iv) the thread sleeps for a period p. As such, the total time t of execution is $t = t_{thread\ execution} + p$. We expect, however, that for the real world $t_{thread\ execution} \ll p$. Typically p in smart grid trials today is $15\,min$ while at very fine grained cases, experiments are run for slightly lower periods (in minutes); however this usually does not include transmission of data, but only its aggregation and transmission every hour or maximum $15\,min$. With our simulator, we are capable of adjusting the rate to very high values, effectively flooding the system with requests. Since the behaviour is common to both meters and concentrators, and the payload is adjustable, we can simulate both meter as well as concentrator upstreams of variable bulk size. As such, the reflection of the real world could be well represented.

As a practical example, if it is known that a concentrator cannot cope effectively with r above its threshold, e.g. $q^c = r^c = 500$ requests/second, this implies that $i = 500$ meters can be installed per concentrator for interval $p > 1$ second. Similarly, if each meter submits the reading once every minute, the number of meters i cannot exceed 30000. As expected, the number of concentrators j and their bulk sizes b_j can be determined from the maximum request rate of the MDS. This is done by fixing one of two key parameters, the number of concentrators or the bulk size.

3.1.3 Data collection methodology

In order to achieve the fine-grained measurements, we had to embed hooks in the architecture externally to the implemented components. The goal is to be able to monitor the path any meter reading follows through the architecture components and measure the execution time of each of the components. As depicted in figure 4, two `valves` were implemented; one for the MDS and one for the concentrator layer. A `valve` is an object placed in the request chain of an application server. As such, any request coming to the server can be captured before it is processed, allowing a better understanding of server's behaviour. Thus, the time between the server receiving a request and the request reaching an EJB can be determined. `Valves` are a feature of the Tomcat web server, which is a component of the JBoss AS.

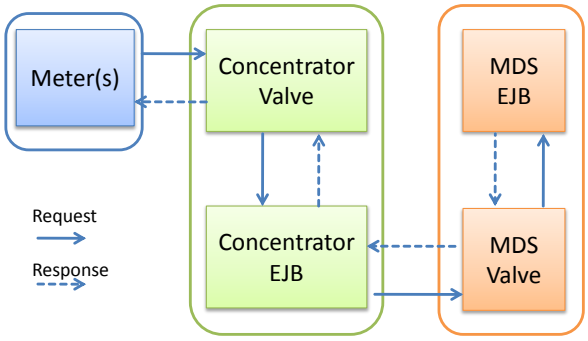

Figure 4: Server request/response process spawning the 3 physical machines and 5 components.

The path followed by each meter reading is depicted in the figure 4. Once submitted by the meter over the web service, it is firstly handled by the concentrator's valve. Subsequently it proceeds to the concentrator's EJB which, along with the other aggregated readings, is sent on to the MDS

as a bulk request. This request is first registered by the MDS's valve, and then arrives at the MDS's EJB for processing and storage. The responses of each request are also recorded. As such, the time for each request enters a component (valve or EJB) and the response leaves the component is collected. This time is measured in milliseconds, since for the implementation we use Java Threads with the default level of granularity i.e. milliseconds.

3.2 Assumptions and experimental validation

The experiments we carried out rely on two assumptions, both of which will be assessed experimentally in order to verify them:

I. Meter readings can be processed at a higher rate, if they are processed in bulk, i.e. multiple meter readings at time;

II. Different combinations of number of concentrators and periods, will result in the same request rate at the MDS.

The fist assumption comes from the fact that there is a time cost associated with the whole process. The cost for each message is associated to transmission, processing the XML and extracting the payload, storage etc. Most of these variables depend on a non-deterministic condition such as the network available bandwidth, the server load etc. To make it more concrete, for each measurement, a connection is established, the data is transmitted, and upon acknowledgement, the connection is closed. The server processes each request upon reception (by extracting the payload) and then stores the reading data for further processing. If this is done for one meter reading at time, cumulatively, the server will be spending a significant amount of processing time per request, leaving fewer resources for the payload processing (that is done by the EJB). If this processing overhead between receiving the request and sending the data to the EJB could be minimized, the throughput of the MDS (i.e. meter readings ratio) would increase. Additionally, more resources could be dedicated to processing and storing the actual metering data (payload).

We assume that there is a point from which the Δ between the cost associated with a single meter reading submission vs. a bulk of them makes a significant difference on the overall performance of the MDS. If this assumption is true, the proposed approach would outperform any other approach where the meters communicate directly with the MDS.

The second assumption is derived from the relationship described in equation 1. The equation defines an upper bound of theoretical expected performance, while excluding the influence of external factors. Such factors could be, quirks of the implementation, performance (e.g. scheduling/load) at operating system level, firewall packet inspection, application server thread priorities etc.

3.2.1 Experimental validation of bulk size considerations

In order to validate these assumptions, an experiment was conducted in which several requests were made from a single concentrator to the MDS, over a range of bulk sizes. For each bulk size, the time taken for the MDS to process the request, the server overhead, and the time taken for the EJB to process the metering data were measured. For the purposes of this experiment, a request rate was chosen so that the metering server would not be overburdened. As such, behaviour of the server could be measured under normal operating conditions.

We have made experiments where the selected range of the bulk size b parameter is defined as $\{b \mid 2 \leq b \leq 100, b \in \mathbb{N}_2\}$. For each b there are 1000 requests from the concentrator to the MDS. The average of meter reading rate r^s for each b is then calculated according to equation 2.

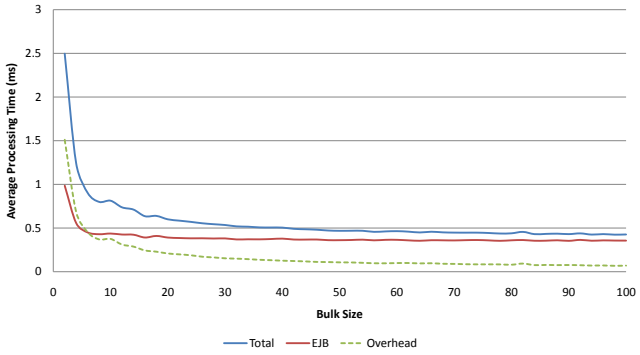

Figure 5: Average total processing time, average EJB processing time, together with the average AS overhead per meter reading for a range of bulk sizes.

In figure 5, the AS overhead, EJB and total processing time per meter reading are shown for each tested bulk size. The total processing time depicts the sum of the the two other values. As we can see there is a clear correlation between bulk size and performance. We also note that the average time to process a single meter reading decreases as the bulk size increases. Furthermore, the rate of improvement decreases as the server converges to its processing limit. An interesting observation is that the EJB processing time reaches its limit faster than the overhead. Thus, minimizing the AS overhead is a candidate for increasing the overall performance. This overhead occurs between the request arrival at the server and the meter reading data arrival at the EJB.

3.2.2 Experimental validation of request rate considerations

To validate our second assumption, an experiment is conducted where the number of concentrators j increases, but the interval p is changed in order to keep a constant request rate q^s (in total) at the MDS. The number of concentrators c_j varies as $\{j \mid 1 \leq j \leq 100, j \in \mathbb{N}\}$, the number of requests per concentrator was set to 50, the bulk size was set to 120 (stability validated in the section 3.2.1). The period p is calculated (in nanoseconds) by the form $p = 1000/\frac{32}{j}$, where i is the number of concentrators attached to the MDS in order to keep the request rate q^s constant for the arbitrarily chosen value of 32 requests/second. With this request rate, p would vary between roughly between 30 milliseconds and 3 seconds, yielding a wide sampling of periods. The arbitrary values q and b are derived from the section 3.2.1, as they result to processing requirement slightly bellow the limit of our testbed server.

The results depicted in figure 6 show that there is a difference between the (theoretical) expected request rate and the actual request rate we measure. This may be attributed

either to the change in the number of concentrators, or the change in p. Given that the simulator creates one thread per (simulated) concentrator, it is unlikely that having a larger number of threads will detrimentally affect performance, in this context. However, small values of p could have negatively effect in the request rate generated by a (simulated) concentrator.

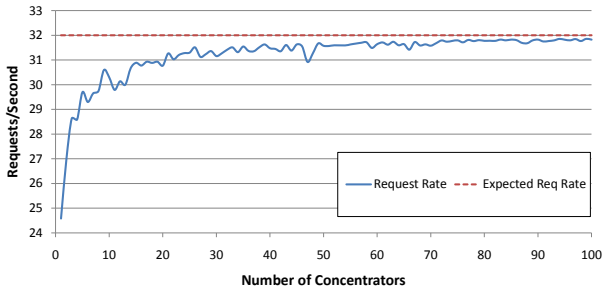

Figure 6: Effect of concentrator population to the requests rate at MDS.

Consequently, we conducted another experiment where the number of concentrators is fixed, and only p varies. In this way, the behaviour of a fixed number of concentrators sending requests at different periods could be isolated. The values for the number of concentrators and bulk size were arbitrarily chosen to be $j = 7$ and $b = 120$ respectively and the period was set $\{p \mid 50 \leq p \leq 2000, p \in \mathbb{N}_{50}\}$. Although value of b is justified before, the value of j is derived from the practical limit of the c component (described later in the section 3.3) and requirement of 1 million metering units for $p = 5\,min$. Experiment results are depicted in the Figure 7, where one can clearly see that the performance impact for low values of p.

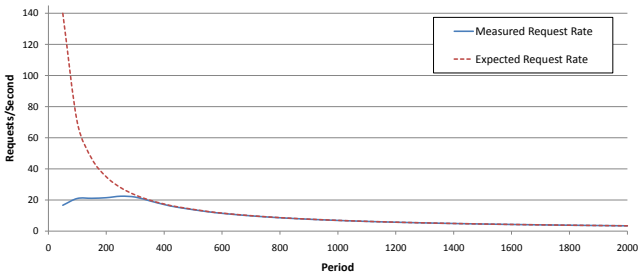

Figure 7: Effect of the concentrator period to the requests rate at MDS.

3.2.3 Summary

To conclude, the results show that the first assumption was correct. There is a performance benefit for processing the meter readings in bulk as it has been analysed in section 3.2.1. As for the second assumption, there was some discrepancy between the theoretical rate and the measured rate for low values of p as analysed in section 3.2.2. This comes from the lower bound limitations that the communication channels place on request rate. That is, since each subsequent request is made only after the previous request has finished, the request rate is limited by average time taken to service each request.

3.3 Concentrator performance

A key consideration when designing AMI infrastructure is the identification of the component performance bottlenecks. The rate r^c of incoming meter readings that a concentrator can handle is important since it may provide a rule of thumb for the number of required concentrators e.g. per geographical region. Therefore we have conducted an experiment for a variable reading rate $r^c = q^c$ where the range $\{r^c \mid 454 \leq r^c \leq 2000, r^c \in \mathbb{N}\}$. This range of values is used to find the hard limit at the concentrator component i.e. the highest value of r^c that the concentrator can cope with, while still maintaining the operational stability. Even if r^c is set to a rate beyond the concentrator's capabilities, it is expected that the actual measured rate will still be constrained by the concentrator's processing limits.

Another parameter examined in this experiment is the bulk size of the concentrator, a parameter that regulates the request rate from the concentrator to the MDS (as depicted in equation 1), and as such may have an impact on the maximum reading rate r handled by the concentrator. Our goal is to find out how high is the rate of requests a concentrator can handle. This of course implies (as denoted in equation 2) a large number of meters sporadically transmitting or a lower population of them but with more often transmissions. Since for each request we assume one connection (with its full lifecycle), the problem may arise when the concentrator is not able to manage them in high rate, and start rejecting further TCP handshakes. Additionally other parameters may interfere with network conditions such as flow control, congestion, window scaling etc. Having said that, we have tried to experiment with rates that would not create the aforementioned problems, but nevertheless be in the high resolution area.

In order to evaluate results from the experiment, we considered the ratio of the measured difference between actual t_a and expected t_e (theoretically calculated) run time. Once this ratio reaches the value of 1, the concentrators is considered stable, since the measured runtime approximates the expected runtime.

$$\lim_{r^c \to 0} t_a / t_e = 1$$

This behaviour is presented with the heat-map graph of figure 8, where the colours vary from white, to green, to red representing the scale from lowest to highest values of the ratio. Thus, the white surface indicates the stability area of the concentrator, while the green and red areas are interpreted as unstable. It is observed that for higher r^c these performance decreases for smaller bulk sizes, since the concentrator has to generate the SOAP envelope (with XML data) more frequently. In any case we do not further consider this area as the concentrator is clearly unstable for very high rates.

Since the stability of the concentrator can be reached only for lower values of r^c (where one clearly sees irrelevance of the b), the information of figure 8 can be also represented as function of average ratio of actual vs. expected execution time $\langle t_a / t_e \rangle$ over b dimension as depicted in figure 9. This clearly shows how the component becomes stable when the average rate converges to the expected rate. For our exper-

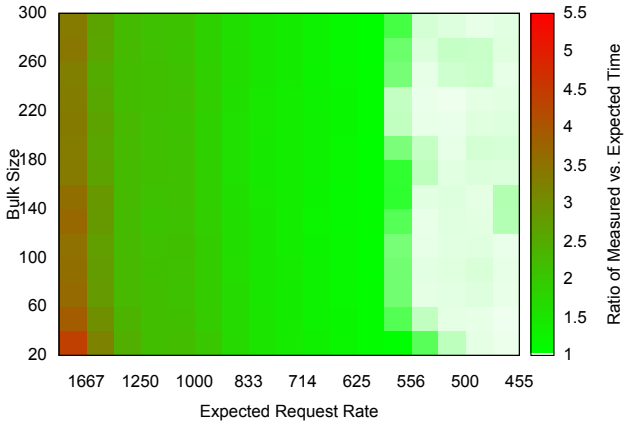

Figure 8: Indicator for the stability of the concentrator based on the ratio of measured vs. expected execution time for different bulk sizes.

iment we conclude that the concentrator is stable for rates $r^c \leq 550$.

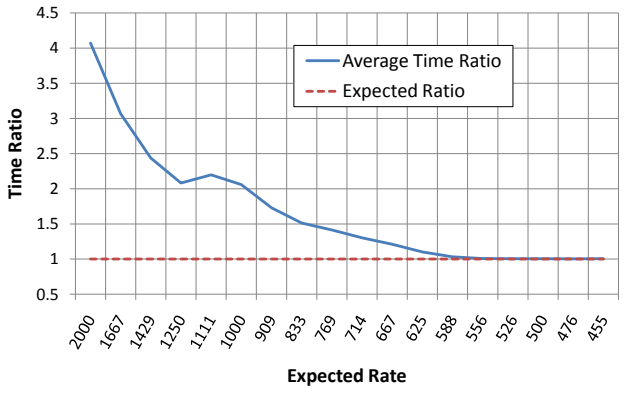

Figure 9: Visualisation of the stability convergence of the concentrator based on the average ratio of measured vs. expected execution time.

Another view of the experimental results we obtained, is through the measurement of r as it is experienced by the concentrator. In this way, one can clarify how fast the concentrator can process requests (with respect to b) for a set of r values. This observation might be irrelevant for high values r, as the system is not stable, however it becomes relevant when the actual rate r_a^c approaches the expected rate r_e^c. If the stability factor is not respected, the processing time required is greater than the processing capacity of the concentrator, which will result in the concentrator being overloaded. As we can see in figure 10 is not quite clear at which r the stability is achieved, but at least we can identify the difference between high r and higher values of b. Here one can see that the measured request rate can be relatively high, at around $r^c = 600$ meter readings per second, however this corresponds to a red region in figure 8, i.e. stability can not reached.

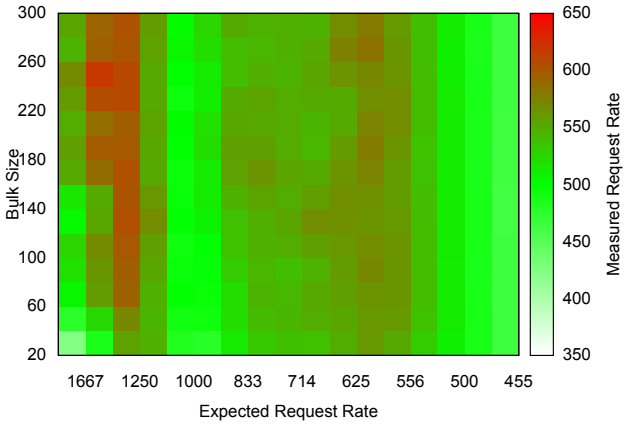

Figure 10: Measured request rate (meter readings/second) for different bulk sizes.

To reiterate, the bulk size parameter variation becomes irrelevant for lower values of r. In figure 11 the average of rates over dimension of b is shown. From this perspective it can be seen how and where the request rate r_a^c is approximately equal to the rate r_e^c. What is interesting is to understand why the approximation is done with such a high variability, however this is considered future work.

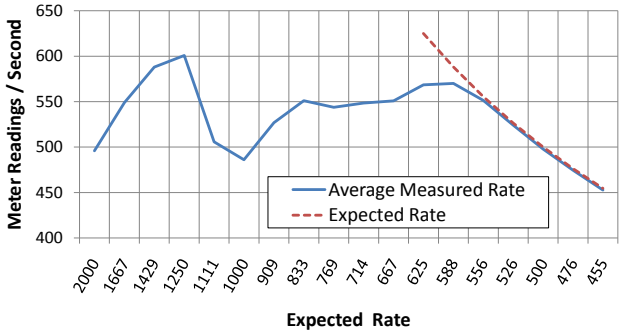

Figure 11: Average measured rate approximating the expected rate.

3.4 Mutli-concentrator MDS performance

Having already verified the performance effect of bulk processing, our focus goes onto the performance of the MDS under the load of a number of j concentrators. We monitored the average request response time for a variety of request rates and bulk sizes. The main objective here is to determine the best performing bulk sizes for particular request rates experienced by the MDS. Once the boundary conditions is ascertained, predictions can be made as to how to best configure the infrastructure in order to handle certain predetermined rate of incoming meter readings.

As mentioned we have build a simulator and used it to simulate multiple concentrators by generating the desired load for the MDS. A fixed number of parallel threads (equal to the number of concentrators) were making periodic requests

in this scenario. Consequently, the total request rate experienced by the MDS depends on the meter reading rate on each individual concentrator. Therefore, j concentrators can deliver the same request rate q of $j/2$ concentrators where each deliver $2q$ request rate. Several combinations of these parameters can lead to the same overall measured request rate at the MDS layer.

The expected request rate q can be calculated as shown in equation 2, out of which also the theoretical rate of meter readings arriving at the MDS can be extracted. By comparing the theoretical meter reading rate r_e^s and the actual (measured) rate r_a^s, for each parameter setting, the processing limits of the MDS can be determined. The processing capacity on s has to be tested for certain bulk sizes. As we can see in figure 12 , the practical capacity on MDS grows in-line with the parameter b. More specifically it grows fast for small bulk sizes, and for higher ones (e.g. $b = 100$) it starts to converge towards $r^s \approx 3900$.

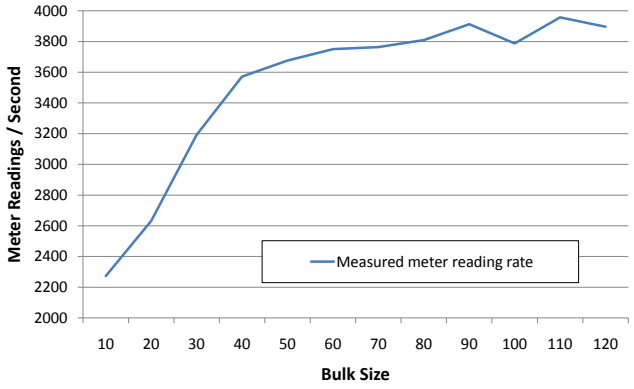

Figure 12: Stable processing capacity of s over the dimension of b.

Once the impact of b on r^s is assessed, we chose a few scenarios to prove this finding. For this experiment, the bulk sizes used are $\{b \mid 10 \leq b \leq 200, b \in \mathbb{N}_{10}\}$. Additionally, five MDS request rates are chosen i.e. $\{q \mid q = 100, 50, 33.33, 25, 20\}$, yielding 100 parameter combinations. The results are depicted in figure 13.

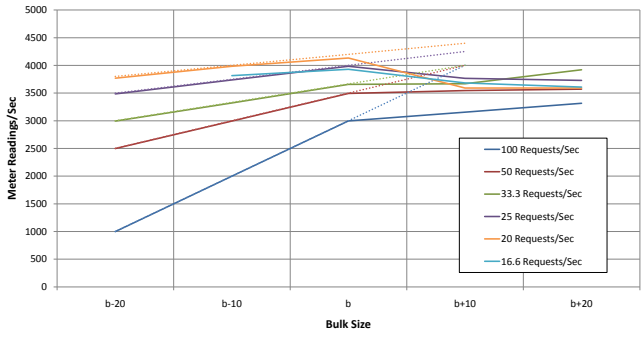

Figure 13: Change of MDS processing capacity for different request rates and bulk sizes. The curves pivot around their threshold bulk sizes.

In figure 13, a qualitative view of the effects of the request rate and bulk size on the processing rate can be seen. The expected request rate performance vs. the actual one is depicted. We can clearly identify a "turning point" for each experiment where the request performance of the MDS starts to greatly deviate from the theoretical performance (as calculated in equation 1). All curves are pivoted around the bulk size where the threshold between normal and deviating server behaviour can be seen.

From the actual threshold bulk size value for each of the request rates (as seen in figure 13) we identify that the threshold bulk size increases as the request rate decreases. This is a consequence of equation 1, as the lower that request rate is, the bigger the bulk size needs to be in order to maintain the same total meter reading ratio r^s at the MDS.

From figure 13 one can conclude that the "turning point" for the $q^s = 100$ requests per second is at $r^s = 3000$ meter readings per second, which is significantly lower than the $q^s = 20$ requests per second rate which has a "turning point" of approx 4100 meter readings per second. The result of this experiment is that the lower the request rate q^s is, the highest the meter reading rate is, where one can verify the effect of the bulk size parameter b. As such, the "turning points" described here are not entirely accurate, because the true "turning point" will be somewhere in the range $[b..b + 10]$. However, these results are more than enough to prove the importance of parameters that may variate in these scenarios.

It should also be noted that the measured values are tied to the hardware and software configuration used for the experiment. Therefore, different configurations will yield different results. This is also a justification for the rather large step size used for each of the parameters. For example, b was increased in steps of 10 and the p in steps of 1000.

3.5 System performance validation

With all assumptions checked and quantified, and the performance of the individual components assessed, we have gathered the necessary experience to configure and test the overall architecture performance. As depicted in section 3.3, in our current testbed the concentrator can process somewhere between $500 \leq r^c < 550$ meter readings per second, and the MDS can process somewhere close to $r^s \approx 4000$ meter readings per sec. These observations are taken from figures 11 and 13 respectively.

If we take the lower bound for the concentrator performance i.e. 500 meter readings/second, a total of 8 concentrators would be needed to fulfil the 4000 meter readings/second at the MDS layer. However, due to the interdependency of period and bulk size, and the constraints found in sections 3.4 and 3.2, values for these parameters that result in a reliable system performance cannot be found. If the transmission period is low, the dynamics of the simulator threads start to impact the performance; conversely if the bulk size is set too high, the MDS cannot cope with it.

Nevertheless a smaller scale experiment is conducted in which only the high performance of the MDS is targeted as overall goal. In this experiment, the period p of all concentrators is uniformly set to 2000, a value that we have witnessed the saturation from previous experiments resulting with $t_{thread\,execution} \ll p$. This value produces reliable predictions for the number of requests/second at the MDS (figure 7). The bulk size was fixed to 120, a value in range

of the tests conducted in section 3.4. Finally, the number of concentrators was varied in the range $\{c \mid 20 \leq c \leq 75, c \in \mathbb{N}\}$. The bulk size was fixed to 120 since it was one of the threshold bulk sizes found in section 3.4. For a rate of 33.3 request/second, it was found that the MDS performance should decline, as can be derived from figure 13.

According to our methodology which we followed so far, at 20 concentrators and a period of 2000, the MDS should receive around $q^s = 10$ requests/second, or $r^s = 1200$ meter readings/second. For each additional concentrator, the request rate should increase by 0.5, thus the meter rate by 60. It is expected that the measured meter reading rate should match the prediction for rates bellow the 4000 maximum rate of the MDS.

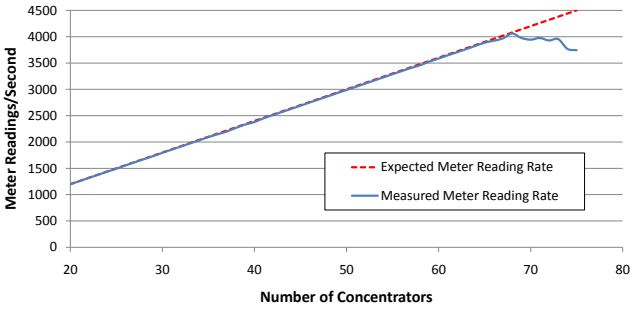

Figure 14: Measured meter reading rate at MDS in multi-concentrator testbed.

Running the experiment with the assumptions we mentioned, we acquired the results depicted in figure 14. As it can be seen, the practical results match the expectation (hence validating the approach). Once the there are enough concentrators to reach meter reading rates greater than 4000, a discrepancy between the predicted and measured rates starts to appear. By comparing these results to figure 13, we notice that rates close to 4000 were not experienced at a 33.3 request rate. Having said that, it should also be noted that the step between bulk sizes in that experiment is 10. As such, it is possible that at, say, bulk size $b = 119$ the system can still perform reliably at meter reading rates nearing 4000.

The aim of this experiment was to accumulate the experiences from the previous three in one composite test. We select the values that gave best system performance (for current setting), predict the theoretical limits, and validate in practice the actual limits. As we have demonstrated these concur, which is a validation of our methodology. Additionally we have clearly demonstrated the architecture's ability to scale and accommodate a higher rate of meter readings by simply adjusting the number of concentrators and configuring them so with the system-wide performance in mind.

4. DISCUSSION

Our goal was to shed more light to the performance considerations that arise when one attempts to realize the AMI envisioned by the smart grid. Since little quantitative work is in bibliography, we have investigated how easy it is to implement it with open source tools and made several thoughts about the possible problems that one has to deal with. The implemented testbed was used as a proof of concept. Several results have been already analysed during the experiment description sections, and it is obvious that trying to tune the whole system towards high performance (within the constraints we have listed in section 1.1), more complex inter-dependencies need to be considered.

4.1 Key findings

We have identified two interrelated key performance indicators that we used to evaluate the performance of the architecture i.e. request rate q, and meter reading processing rate r. Measuring the capacity of a component to handle different magnitudes of these rates is the first step in configuring the parameters of the architecture for high throughput. These two rates are of course related by the bulk size b, which, as we have witnessed (in sections 3.2.1 and 3.4), plays a pivotal role in the maximum capacity that a component can reach.

The experiments presented are designed and implemented, with the objective of probing our considerations in practice. First we verified two major assumptions adopted in this work. The first being that the concentrator plays a key role towards increasing the overall performance of the architecture, was proven to hold true. The second assumption, that different configurations of concentrator request rates and bulk sizes, could yield the same request rate on the MDS. In this case, the assumption was practically shown to hold true for large values p. Although the p value in tests was valid for laboratory environment, this does not necessarily hold true for more complex network settings e.g. spawning multiple infrastructures and/or communication channels.

In the experiment of section 3.3, the performance of a single concentrator was measured. By varying the request rates (at the concentrator) and the bulk sizes, we were able to ascertain a region within which the system would behave as expected. This region in our specific setup was found to be for request rates of less than roughly 550. The reason for this limitation can be attributed to the software or hardware components used.

In the experiment of section 3.4, we measured the performance of the MDS against several concentrators. Here, we found that the maximum throughput of the MDS varied according to the period. While increasing the bulk size increased the throughput, this only happened up to a particular threshold bulk size. This behaviour can be explained from the results of the first experiment (section 3.2). Increasing the bulk size increased the throughput of the server by lowering the overhead processing on the application server. However, since more payload (larger number of meter reading due to bulk size) needs to be processed as well, the request response time also increases. Since the measurements integrate the time for processing a request, and this time increases as the bulk size increases, it is clear that the bulk size cannot grow arbitrarily.

Due to all aforementioned constraints, in the experiment of section 3.5, we only targeted the full capacity of the MDS. Based on our findings, using the concentrator at its full capacity would always result in the bulk size being too large to be handled within the specified period. As such we resulted to demonstrate the scalability of the system, as more and more concentrators were added to no detriment of performance (until the MDS's capacity was reached).

One key result that struck us was the amount of band-

width waste. We already knew that a big percentage of the message transmitted would be devoted to the actual wrapping and envelope of the data in XML. As you can see in figure 15 for a single HTTP request 60% of the message is occupied by the SOAP envelope while only 9% is devoted to the actual meter reading data. In case of bulks where we have aggregated meter readings the percentage of information increases but still e.g. for $b = 100$ approximately 68% is occupied by SOAP while 30% of it is devoted to the actual meter reading data.

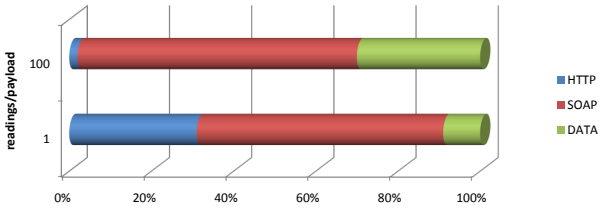

Figure 15: Message payload efficiency.

The experiments have demonstrated the importance of the concentrator component. While the MDS and the concentrator cannot be compared exactly one to one, the MDS can cope with nearly 8 times the number of meter readings. This would suggest that another avenue for exploration would be bulk metering data being sent from the meters to the concentrators. This could further improve the capacity of the concentrator and therefore reduce the required number concentrators needed in a deployment. The only down side is that, while one could attain a higher level of granularity in the readings, the time taken for the user to access them through any enterprise system would still be limited by the period. For instance, if a meter submits a meter reading every 5 minutes, a bulk message with 5 meter readings could be sent at the same rate, thus attaining one minute granularity in the readings.This could be a cost effective strategy for increasing the reading granularity with little changes to the system.

4.2 Limitations

An unexpected result of our test also demonstrated that there is a minimum period, p, that can be set at a component. This is a result of the fact that the requests are made serially, resulting in a detrimental impact on the request rate of a component, when the period is smaller than the time required for a request to be made. Unsurprisingly, further evidence of this effect can be seen in figure 13, where the threshold bulk size increases as the request rate drops (i.e. the period increases). A higher period will give the thread more time to do its work; at the same time, the higher the bulk size, the longer a request will take. Thus, it is clear that the bulk size and the period are more correlated than initially thought, and as such, great care needs to be taken when configuring (or choosing) the system components. Alternatively, however, if the requests for each component are done in parallel, perhaps this effect can be minimized.

For low values of p we have run into simulator implementation drawbacks. First of all, the thread that simulates the concentrator may not have enough time to do complete its work. As clarified in 3.1.2 the total time t of execution is $t = t_{thread\ execution} + p$, where the expectation was that

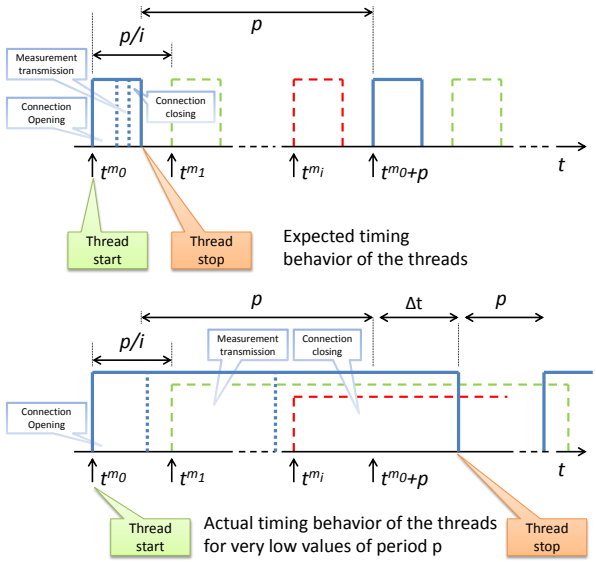

Figure 16: For very low values of p, thread runtime is comparable or even exceeds the p.

$t_{thread\ execution} \ll p$. However since the simulator creates multiple threads, when p has very low values (in millisecond domain), it might be that $t_{thread\ execution}$ and p are in the same magnitude of values or even $t_{thread\ execution} > p$ (as depicted in 16). We have to consider though that the operations of the thread, that is, opening of the connection, submitting the data, and closing the connection, have almost fixed limits (due to the physical communication involved). In the millisecond domain this may lead to the fact that the assumption of a uniformly fixed number of threads operating within a fixed time frame may not hold true. Up to now it was assumed that this interval was approximately p, however for very low values of p it may be a multiple of p. This creates a variable delay for each execution of the respective thread which is propagated additively to its overall lifetime. Additionally, since the meter simulator shares the same code base and conceptual model, the extended lifetime effect holds true for both simulators.

4.3 Market relevance

There is not much info available towards high-performance or large scale high resolution smart metering. Itron Inc. [1] for instance indicates the import, validation and storage of 26389 measurements per second (190 million reads in a two-hour processing window) in its enterprise product. However, this metric is only partially comparable to our results as it seems that this was not live data, but a database import in the specific product. The most competitive high performance reading to our knowledge is the research result achieved within the DEHEMS project [11] where three million meters were simulated and rates of 40k meter readings per second have been achieved by focusing on a high performance DB. As you will see later in Table 4.3 we achieved results that can scale up to these values.

We have created a testbed depicting the AMI functionality relevant to the smart metering. We have used open source components and commonly available cheap hardware. It is therefore possible to have the basic functionality envisioned

Interval	Max Meters/Conc.	Max Meters/MDS
1 min	30000	240k
5 min	150000	1.2M
15 min	450000	3.6M

Table 1: Maximum number of meters per concentrator and MDS assuming the highest meter reading rates for each component (500 and 4000 respectively).

Interval	# MDS	# Conc.	# Meters
1 min	1	$[20 \dots 66]$	$[72k \dots 237.6k]$
	2	$[40 \dots 132]$	$[144k \dots 475.2k]$
	3	$[60 \dots 198]$	$[216k \dots 712.8k]$
5 min	1	$[20 \dots 66]$	$[360k \dots 1188M]$
	2	$[40 \dots 132]$	$[720k \dots 2376M]$
	3	$[60 \dots 198]$	$[1.08M \dots 3564M]$
15 min	1	$[20 \dots 132]$	$[1.08M \dots 3564M]$
	2	$[40 \dots 132]$	$[2.16M \dots 7128M]$
	3	$[60 \dots 198]$	$[3.24M \dots 10692M]$

Table 2: Maximum number of meters for a range of concentrators accepting readings at a rate of 60 meter readings/second.

by the smart grid at low cost; however it is clear that this is only a tiny fraction of the real needs. Although system reliability was not a concern in our experiments, it is of utmost importance in a real world system where several levels of redundancy would need to be incorporated. Additionally load-balancing techniques might be employed to control the load at concentrator and MDS level. We expect that an MDS cluster will be strategically located depending on the scale of the underlying infrastructure and the performance requirements set by the real-world enterprise applications.

In the experiment demonstrated in section 3.5, each concentrator is sending meter readings at a rate of 60 meter readings/second. According to current smart grid industry practices, each meter under a concentrator is sending a reading every 15 minutes; with this level of granularity, a total of 54000 meters would be connected to the concentrator. The results show that, for the tested configuration, the MDS performance peaks at 66 concentrators. This results to roughly 3.5 million meters that can be incorporated in this configuration (as depicted in Table 4.3).

If we further reduce the granularity of the readings to 5 minute intervals, some 1.18 million meters can be handled by the architecture. It should be pointed out that 60 meter readings/second equates to a 12% capacity factor utilization of the presented concentrator. While, in this last experiment, it is only attempted to meet the 100% utilization of the MDS, further experiments should be conducted to tune the parameters to make higher use of the concentrator's capacity as well. While the number of meters that can be handled by the architecture is limited by the processing rate of the MDS, utilizing a higher capacity of the concentrators, will result in fewer concentrators being needed. This will obviously help to keep costs down, as fewer concentrators will be needed to be deployed, managed and maintained. The correlation between the time resolution, max meters per concentrator and max number of meters per MDS is depicted in 4.3. Following this line of thought, Table 4.3 depicts various configurations that can be used as a thumb rule when we set up an AMI.

4.4 Future directions

In the realised experiments we measured the performance of our approach in a high bandwidth, single hop and unconstrained network. This, however, is not a reasonable expectation for a real world scenario, at least for the mid-term. As such, the performance must be further scrutinized under far harsher network conditions that are more reflective of reality. Especially between the concentrator and the meter where a variety of heterogeneous networks is expected (e.g. residential ADSL connection, power line communications or even through existing wireless mobile phone networks), one should experiment and consider the constraints of those media on the overall performance. We also assume that since no global view of when to submit the data to the concentrator and MDS is available, this will lead to variable phenomena like peak loads and rejection of additional connections at those levels. Such diverse conditions are likely to affect the performance of the system, and these effects will need to be identified and measured.

From the technical viewpoint we considered a web service enabled infrastructure and more specifically the traditional implementation of web services where SOAP is used. However there are more lightweight approaches out there. We consider experimenting with REST (REpresentational State Transfer) architectures that promise a better performance. Eliminating SOAP is expected to benefit the message size as we have shown in figure 15, which will help also transmission over slower links as well as unmarshalling of the payload. Alternatively the use of binary XML (e.g. Efficient XML Interchange) may assist in reducing the verbosity of XML and the cost of parsing. Initial results for web service enabled devices such as the smart meters are under promising [9].

It is clear that the results presented are bound to the actual implementation tools and their limitations. We have not invested any effort to tune them for high performance. For instance we have used as an Application Server the widely popular and open source JBoss AS (version 5.1.0.GA). Validating that the same class of results can be realised also with other AS implementations such as Glassfish and Apache Geronimo would be also necessary as they might deviate from the JBoss out-of-the-box behaviour. It is also worthy to note that the Java EE (Enterprise Edition) version 5 was used in the development of the aforementioned web services. The new version, Jave EE 6 might similarly provide a performance boost.

In all of our experiments we have not addressed security and privacy. Clearly integrating any solution there will a significant impact on all layers. Experimenting with WS-Security, secure channels (HTTPS) or encrypted meter readings, will give an insight to the magnitude of impact [5]. Additionally the use of latest hardware (not dedicated though) such as Intel processors of the i5/i7 generation which have native AES support may assist in minimizing the impact of security. Without doubt further tuning of hardware and software may further enhance the overall performance.

5. CONCLUSION

Within the scope of this work we have demonstrated an approach that can be used in order to create a rule of thumb

when one is designing an AMI targeting high resolution meter readings. We have taken a straightforward 3-layered hierarchical architecture and evaluated its components from the performance point of view. In a methodological way we have identified and discussed potential problem areas as well as the line of thought that should be followed in order to find possible inter-dependencies and roadblocks. By investigating each component's limits we were able to narrow down the operational ranges that could be used to achieve high performance. Our initial assumptions were validated and some unexpected results that came up give avenues for further research in the area.

We have demonstrated that contrary to widespread opinion about the high cost of any AMI solutions, it is possible to realize a high performance AMI based on common hardware and open source tools. Although no tweaking was done on the hardware or general tools (i.e. operating system, application server etc.), we have shown that high resolution metering correlating to several millions of meters can be supported with a limited number of concentrators and MDS as depicted in Table 4.3. We are confident that these limits can be further pushed if one more coherently take advantage of software and hardware tuning, which would be of interest to commercial solution providers.

Although the actual experimental measurements may vary and depend on the hardware and (optimized) software that can be used, one can follow exactly the same methodology i.e. the steps we took to evaluate our set-up and identify the limits of their deployment. This is an additional contribution of this work. It is our hope that this methodology will be of usefulness to the ongoing efforts towards high resolution energy monitoring and timely assessment of it.

6. ACKNOWLEDGMENTS

The authors would like to thank for their support the European Commission and the partners of the projects SmartHouse/SmartGrid (`www.smarthouse-smartgrid.eu`), NOBEL (`www.ict-nobel.eu`) and CONET (`www.cooperating-objects.eu`) for the fruitful discussions.

7. REFERENCES

[1] Itron enterprise edition meter data management affirms greater scalability and performance in independent testing. Press Release, 2007.

[2] NIST framework and roadmap for smart grid interoperability standards. National Institute of Standards and Technology (NIST) NIST Special Publication 1108, National Institute of Standards and Technology (NIST), Jan. 2010.

[3] Federation of German Industries (BDI). Internet of Energy: ICT for energy markets of the future. BDI publication No. 439, Feb. 2010.

[4] International Telecommunication Union (ITU). Itu internet report 2005: The internet of things. 2005.

[5] M. B. Juric, I. Rozman, B. Brumen, M. Colnaric, and M. Hericko. Comparison of performance of web services, ws-security, rmi, and rmi-ssl. *J. Syst. Softw.*, 79:689–700, May 2006.

[6] S. Karnouskos. The cooperative internet of things enabled smart grid. In *Proceedings of the 14th IEEE International Symposium on Consumer Electronics, Braunschweig, Germany*, 07-10 June 2010.

[7] S. Karnouskos and O. Terzidis. Towards an information infrastructure for the future internet of energy. In *Kommunikation in Verteilten Systemen (KiVS 2007) Conference*. VDE Verlag, 26 Feb. - 02 Mar. 2007.

[8] S. Keshav and C. Rosenberg. How internet concepts and technologies can help green and smarten the electrical grid. In *Green Networking*, pages 35–40, 2010.

[9] G. Moritz, D. Timmermann, R. Stoll, and F. Golatowski. Encoding and compression for the devices profile for web services. In *The Fifth Workshop on Service Oriented Architectures in Converging Networked Environments (SOCNE 2010), Perth, Australia*, pages 514–519. IEEE Computer Society, 20-23 April 2010.

[10] SmartGrids European Technology Platform. Smartgrids: Strategic deployment document for europe's electricity networks of the future, Apr. 2010.

[11] V. Sundramoorthy, Q. Liu, G. Cooper, N. Linge, and J. Cooper. DEHEMS: a user-driven approach for domestic energy monitoring. In *Internet of Things 2010 Conference, Tokyo, Japan*, 29 Nov - 01 Dec 2010.