# A Lightweight Messaging Protocol for Smart Grids

Eric MSP Veith*, Bernd Steinbach† and Johannes Windeln‡

*‡Institute of Computer Science
Wilhelm Büchner Hochschule
Pfungstadt, Germany
e-mail: eric.veith@wb-fernstudium.de

*†Institute of Computer Science
Freiberg University of Mining and Technology
Freiberg, Germany
e-mail: veith@mailserver.tu-freiberg.de

***Abstract*—The smart grid concept introduces more software control at both endpoints of the energy consumption chain: The consumer is integrated into the grid management using smart metering, whereas the producer will be host to a distributed agent-based software approach. Including more renewable energy sources in the energy mix will increase the necessity for a finer-grained, automatic control of changes in the energy level. Such changes need to be communicated for a distributed system to be able to calculate supply and demand. We therefore propose in this paper a lightweight protocol, which can be implemented on top of existing technology providing the needed communication interface. We also specify common behavior and protocol semantics for all implementing nodes, which forms the basis of a distributed, decentralized demand and supply calculation in a future energy grid.**

***Keywords*—*smart grid; messaging; protocol description; renewable energy sources***

## I. Introduction

The term "smart grid" unifies a number of concepts related to an automated, information-supported management of the energy grid. In his paper "integration is key to smart grid management" [1], J. Roncero shows how different technologies are involved in the rather abstract smart grid concept.

Although a tighter integration of customers via smart metering is considered one of the cornerstones of a smart grid, the increased usage of renewable energy sources will also play an important role. However, although better appliances lead to a more efficient usage of renewable energy sources, this also leads to a higher dependence on energy which is not entirely controllable by the utility, since energy generated by wind parks or solar panels depends on the wind speed or solar radiation, more specifically, the weather.

This means that there are variances on both sides of the producer-consumer chain. Forecasting, as it is already employed via, for example, standard load profiles, helps to create more certainty regarding the variances itself, but it will also increase the number of calculations needed and the amount of data analyzed. One could therefore argue that a "divide et impera" approach is necessary, which leads to a decentralized, agent-based infrastructure. Such an approach, however, needs an information interchange protocol.

In this paper, we propose a lightweight protocol which can be used in such a grid based on distributed software and control. It will define certain basic protocol semantics which will enable this grid to organize itself based on the information available. This protocol, as described here, is not based on a specific implementation: We propose the fields required and how they are used but refrain from creating a bit-for-bit specification. This protocol can, however, easily be implemented on layer 7 of the ISO/OSI protocol stack using, for example, JavaScript Object Notation [2] (JSON) as a common data interchange format.

The remainder of the paper is structured as follows: After describing our initial motivation in Section II, we will outline the basic protocol structure in Section III, along with common behavioral rules in Section IV. Afterwards, Section V will describe the different types of messages available in the protocol. The discussion in Section VI will show how the protocol can be applied and outlines several scenarios and the protocol's behavior therein. We conclude with our plans for future work in Section VII.

Since this paper describes a protocol, requirement levels for implementors must be clear. In Sections III–V, this paper makes use of the keywords listed in RFC 2119 [3]. This includes "must", "must not", "required", "shall", "shall not", "should", "should not", "may" and "optional".

## II. Motivation

The International Electrotechnical Commission's IEC 61850 standard has first been issued for communication within a subsystem automation system [4], but, in the meantime, has been expanded to other applications as well [5]. Higgins et al. [6] show how IEC 61850 can effectively be used for automatic failover.

While IEC 61850 proposes a rich data model for smart grid devices, it does not define mechanisms for a pro-active, decentralized interaction of the different components. The increasing inclusion of renewable energy sources tied to external influences — like wind or solar radiation — also increases the need for an higher frequency of control messages.

These messages could be issued by a central control unit which observes the state of the whole or a part of the energy grid and, given all information available, decides on the proper course of action. Such a central control unit, however, must be properly equipped to handle the information load, must be equally well connected to avoid that the communications infrastructure becomes a bottleneck, and must be extendible to add features which help towards a more pro-active operation, such as forecasting.

 ISBN: 978-1-61208-292-9

Since such a central control unit also poses a single point of failure, it is often assumed that a de-centralized layout of cooperating agents would be a better approach [7]. This would not concentrate the control logic in one point, but distribute it over the grid. Such a distributed system would need a protocol that would allow for interchanging information critical to the actual operation.

IEC 61850 offers a very fine-grained data model for electric grids. However, it misses a simple protocol mechanic that would be applicable in the distributed supply-demand calculation that immediately becomes necessary in a smart grid consisting of distributed agents. These agents would have to interchange information about the energy state, i.e., demand or over-supply that is introduced by weather changes or consumer behavior. A projected increase in wind speed at evenings would lead to an over-supply, whereas employees returning home at a projected time of 6pm would mean a demand.

These two simple examples show how the need for a distributed supply-demand calculation arises. We therefore propose a lightweight, simple high-level protocol that can form the basis of this calculation.

While the protocol as it is described here is not based on IEC 61850 per se, it can still be used on top of IEC 61850 by employing an appropriate application programming interface (API). In fact, we avoid to enforce implementation details wherever possible to make a widespread adaption possible.

## III. Basic Protocol Structure

Nodes within the grid exchange data via *Connections*. A connection is, to the protocol, a virtual concept which resides in layer 7 of the ISO/OSI protocol stack. As such, it is not tied to Internet Protocol (IP) addresses or other concepts of lower levels in the ISO/OSI stack. Connections must be end-to-end; they are bi-directional communication channels between exactly two nodes. Concepts such as multicast must be realized on top of this.

A connection serves two purposes. First, it identifies the two endpoints. Second, by establishing a (largely virtual) network of nodes and connections, this protocol creates a communications structure that resembles the actual power grid, recreating it on top of any other networking structure, such as an IP-based wide-area network (WAN). This way, the power grid and the telecommunications infrastructure do not have to match in their layout. The layout recreation algorithm must be implemented by the actual connection facilities which, e.g., map to an IP network.

Having those virtual connections represent the actual physical power supply line also enables us to model “dumb” cables, which have no other properties than a maximum capacity and a line loss. Taking these attributes into account, the actual power transfer becomes part of the protocol. Smart power supply lines which are equipped with, e.g., metering devices, become nodes of their own. The simple power line–connection unit then evolves into a connection–power line–connection building block, which also adheres to the protocol semantics described in the following section.

Messages can travel further than the node–connection–node boundary. To enable nodes to answer to requests which do not originate from their immediate neighbors, each node must be uniquely identifiable. The *Sender ID* of a node must be unique at any given time. It is an opaque bit array of arbitrary length and must not contain any additionally information about the node itself or anything else. Generating an universally-unique identifier (UUID) [8] whenever the node’s software boots is one way to get such an identifier.

Each message must contain an unique identifier (*ID*). This is important since messages fall into two distinct categories: requests and answers. A request is sent actively by a node because of an event which lies outside the protocol reaction semantics, such as a changed power level. Answers are reactions which occur because of the protocol semantics as described below. Since any reaction pertains to an original action, it needs to identify this action, which is the reason for the unique identifier of each message. Reactions must carry a new, unique identifier, too.

The type of the message must be denoted by a *Message Type* field. The mapping is outlined in Table 1. These numbers are simple integer values with no coded meaning whatsoever. We do not distinguish between message classes or priorities here: The goal of the protocol is to remain simple, and we believe that the message types outlined here suffice in reaching the primary goal of the protocol, i.e., energy supply-demand mediation.

A message must also contain a *Timestamp Sent* field denoting the time and day when the message was initially sent as an Unix Timestamp (see [9] for the definition of the Unix Timestamp).

To prevent messages from circulating endlessly, a time-to-live (TTL) field is introduced. This TTL has the same semantics as the IP TTL [10] field: It starts at a number greater than 0. Whenever a message is forwarded or sent, the TTL is decremented by 1. If the TTL reaches 0, the packet must not be forwarded or otherwise sent but must be discarded. Messages with a TTL value of 0 may be processed.

Additionally, an *Hop Count* is introduced. The Hop Count is the reverse of the TTL: It starts with 0 and must be incremented upon sending a message. It allows to measure the distance between two nodes in the form of hops.

A message must carry an *Is Response* flag to distinguish original requests from responses. If the *Is Response* flag is set, the ID of the original message is contained in the *Reply To* field. If Is Response is not set, the Reply To field must not be evaluated; however, if a response is indicated, *Reply To* must contain a value which must be evaluated by the receiving system.

An answer must also contain the original message’s *Timestamp Sent* field (in addition to its own), and the *Timestamp Received* denoting the time when the original message was received.

To summarize, each message must contain at least the fields of the following enumeration. In parentheses, we give a proposition of the identifier that could be used in an actual

 ISBN: 978-1-61208-292-9 

| Value | Type |
|---|---|
| 0 | Null Message |
| 1 | Echo Request |
| 2 | Echo Reply |
| 3 | Online Notification |
| 4 | Offline Notification |
| 5 | Demand Notification |
| 6 | Offer Notification |
| 7 | Offer Accepted Notification |
| 8 | Offer Acceptance Acknowledgement |
| 9 | Offer Withdrawal Notification |

Fig. 1. Message Types

implementation.

1) message ID (`ID`)
2) message type (`type`), see Figure 1 above
3) original sender ID (`sender`)
4) timestamp sent (`sent`)
5) TTL (`TTL`)
6) hop count (`hops`)
7) is response (`isResponse`)

The message type defines what additional values a message carries; these message types are described in Section V. The message type itself is a simple integer value field with type-to-number mapping shown in Table 1.

If *Is Response* is true, the following fields must be added:

1) reply to (i.e., original message ID) (`replyTo`)
2) timestamp received (`received`)

## IV. Common Protocol Semantics

The following rules must be applied to each message, regardless of their type.

First, a message must not be ignored ("no-ignores" rule).

All messages except the *Null Message*, the *Echo Request Message* and the *Echo Reply Message* must be forwarded, partially answered and forwarded, or answered. This is the "match-or-forward" rule. It becomes important with requests and offers and is it further specified in Subsections V-F and V-G.

*Forwarding* denotes the general process of receiving a message and resending it. The message may be modified in this process, for example, the requested energy level must be lowered when a node can fulfill a portion of the request (see below).

When forwarding, a given message must be sent on all connections except the connection on which the original message was received. This prevents message amount amplification: Would the receiver also send the message on the connection on which it was originally received, it would be useless since the original sender already knows about its offer or request. It would thus only lead to additional processing and unnecessary use of bandwidth ("forwarding" rule).

Each node must keep a cache of recently received messages. If a message is received again, it must not be answered or forwarded ("no-duplicates" rule).

## V. Message Types

### A. Null Message

The *Null* message is the simplest message available in the protocol. It contains no additional information besides the basic protocol fields each message carries.

Null messages can be used as a form of heartbeat information. This is especially useful on weak links, for example for a remote wind park which might only have a mobile phone (GSM) connection. It thus can be sent at regular intervals to keep the line open.

A Null message in JSON representation is shown as an example in Figure 2.

### B. Echo Request Message

An *Echo Request* can be sent on any connection to see if the endpoint is still alive and reachable. It must be answered. An Echo Request must not be an answer, and it also must not contain any additional information.

### C. Echo Reply Message

An *Echo Reply* is the answer to an *Echo Request*. It must always be an answer and thus cannot be sent independently. This message type also does not contain any additional information; the proposed common fields (*Timestamp Sent* and *Timestamp Received*) are sufficient for Round-Trip-Time measurements.

### D. Online Notification

Using this message, a node in the grid can notify its neighbors that it is going online or will be online at a certain point in the future.

To actually be able to carry the second kind of information, i.e., going online at a certain point in the future, this message contains two additional fields: *Valid From* (`validFrom`) and *Valid Until* (`validUntil`). A message using validity dates must use the *Valid From* field and may optionally make use of the *Valid Until* field.

This concept of validity dates is used by other message types, too. It denotes a timespan between the time indicated by *Valid From* and *Valid To*, both inclusive. Both fields are Unix timestamps like, e.g., the *Timestamp Sent*. Whenever a node wants to indicate that a message is valid immediately, it places the current time and date in the *Valid From* field. A "valid until further notice" semantic can be achieved by omitting the *Valid Until* field entirely.

Any protocol implementor, however, must take care to adjust his implementation whenever the Unix timestamp data type changes. As the time of writing, a Unix timestamp of 64 bit width is typically used in modern operating systems, which provides enough seconds since 1.1.1970 for the whole lifetime of this protocol. Previously, the `time_t` C type was specified as having 32 bits, which meant that an overflow would happen on 19.01.2038, the so-called "year 2038 problem".

Note that the Unix timestamp also allows for negative values to represent times before 1.01.1970. Although this would not be a necessary feature in the terms of this protocol, we advise

 ISBN: 978-1-61208-292-9

```
{
ID: "c3e5aca2-616f-4003-bbc6-eb9e90335495",
type: 0,
sender: "2d60a262-903e-4f70-a9de-e4d9b83d2bb7",
TTL: 42,
sent: 1367846889,
hops: 23,
isResponse: false
}
```

Fig. 2. An example for a null message, encoded as JSON

against choosing an unsigned type as it would introduce the need for additional programing quirks for implementors.

An *Online Notification* may be forwarded, but can also be discarded. This type of message is important for all directly connected nodes, because it has influence on the wires connecting the originating and its neighbor nodes. Any change in power levels, however, will be communicated using demand/supply messages which will be described later.

### E. Offline Notification

The *Offline Notification* is the counterpart of the aforementioned *Online Notification*. It notifies the neighboring nodes that the originating node will be offline (i.e., possibly disconnected from the grid), utilizing the same *Valid From* and *Valid To* timestamp fields.

Unlike the *Online Notification*, this type of message must be forwarded. It provides additional information to the energy supply/demand solving algorithms of other nodes, which get a chance to re-calculate their supply plans. It is assumed that a demand or supply message which reaches the node sending the *Offline Notification* means that the *Offline Notification* will also be received by the original sender of the demand/supply message since the hop count is the same both ways.

However, since the *Offline Notification* message does not contain a field supplying the change in the grid energy level when the shutdown happens, an additional supply/demand message must be sent if the node has influence on the grid's energy level.

### F. Demand Notification

A *Demand Notification* message indicates the need for energy of a particular node. It carries the *Valid From* and *Valid To* fields.

Additionally and primarily, it carries the quantified demand for energy in watts in the *Power* (`power`) field. Fractions of watts are not supported, i.e., the lowest amount that can be requested is 1 W. The field must not be 0, as this would make the message itself superfluous. This field must not carry negative values; those would mean an offer which has its own message type.

*Demand Notification* messages must be forwarded if they cannot be (completely) fulfilled. Each node must try to react to a demand message, i.e., try to match it and supply the energy requested. This is called the "match-or-forward" rule as described above. If it cannot fulfill the demand, it must at least forward it under the semantics outlined in Section III.

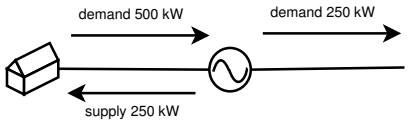


Fig. 3. A *Demand Notification* having the "match-or-forward" rule applied

If the node can supply the requested amount of energy completely, it must notify the requester using an *Offer Notification* message. It must not forward the original *Demand Notification* then.

If, however, the demand can only be partially fulfilled, the node must send an *Offer Notification* indicating the amount of energy that can be offered. It must then subtract this value from the original value indicated in the request and forward the thusly modified message. It must not change the message's ID or the message's sender ID ("same-ID" rule). The partial matching described in this paragraph is depicted in Figure 3.

A *Demand Notification* message must not be an answer.

### G. Offer Notification Message

This type of message indicates an offer to the grid. It carries the fields *Valid From* and *Valid Until* as they are described in Subsection V-D and the amount of energy offered in the field *Power*. This number is an unsigned integer and is expressed in units of watts with no fractions possible.

Additionally, the offer includes a field *Cost*, which carries the cost of this offer in cents per kilowatt hour (ct/kWh). This allows for implementing cost-based policies, such as accepting energy only if it is cheap.

An *Offer Notification* may be an answer. If so, it is an answer to a previous *Demand Notification*, as described in the above subsection. A node receiving multiple offers must prefer offers of lower hop count over those with higher hop count. This favors micro-grids and reflects the actual flow of energy.

However, *Offer Notification* messages may also be sent as a request. This is the case whenever the agent estimates that it will output more power than it currently does. Consider for example a wind park which is dependent on the weather. If the agent's forecasting module predicts an increased wind speed in an hour and therefore an increased energy output, it may send an *Offer Notification* instead of pitching or stalling the wind turbines.

Just like a *Demand Notification*, such an original offer must be matched by nodes in the grid. The difference between an original offer and one that is an answer to a request is the value of the *Is Answer* field: If set to 0, the offer must be matched.

For matching and forwarding, the same mechanics as for the *Offer Notification* message type applies, especially if it can only be partly fulfilled.

### H. Offer Accepted Notification

Whenever a request for energy is made and the offers have been received, there may arise a situation when more energy

 ISBN: 978-1-61208-292-9

is offered by all nodes than originally requested. For example, if a wind park, a solar park and a traditional power plant send *Offer Notification* messages after a request has been sent, the sum of energy offered is likely to exceed the original amount requested.

For this reason, a node must indicate which offer it accepts. Otherwise, all offers would be fulfilled, leading to an over-supply of energy in the grid which would be fatal.

As soon as the node finishes its demand/supply calculation, it must send *Offer Accepted Notification* messages to all nodes that were offering energy. In the body of the message, it must list the IDs of those nodes whose offer it takes. All other nodes will notice that their ID is missing from the notification and thus not actually deliver the energy they offered.

An *Offer Accepted Notification* must be an answer. It must also be sent of the node is taking on an original offer (as indicated above). In that case, the *Offer Accepted Notification* must be addressed to the offering node only, while the original offer must be forwarded if it cannot be completely fulfilled as described in Subsection V-G.

### I. Offer Acceptance Acknowledgement

After an offering node has received an *Offer Accepted Notification*, it must reply with an *Offer Acceptance Acknowledgement* to indicate that the offer is still valid. This message type must always be an answer.

### J. Offer Withdrawal Notification

If a node has offered a certain amount of energy, be it as an answer or as an original offer, and it can no longer stand up to the offer, it must withdraw it. This type of message is always an answer, carrying the ID of the original offer (in case of an original offer that was withdrawn) or the ID of the original request in the *Reply To* field.

If a node can still offer energy, but the amount has changed, the original offer must be withdrawn using this message type, and the new amount must be announced separately.

## VI. Discussion

Based on the message types, the protocol structure and the semantics defined in the corresponding section, we will now illustrate how the protocol helps in a distributed supply-demand calculation. Therefore, we will not only discuss general properties of the protocol, but also present scenarios to show the protocol's behavior.

To support the goal of a distributed supply-demand calculation, the protocol must first and foremost help to make a demand or over-supply known. This is, of course, in the first place the task of the node itself; the protocol assists by merely providing a means to transport this change in the grid's energy level via the *Demand Notification* and *Supply Notification* messages.

A wind park, which is dependent on the weather, can already use the supply message to indicate changing power levels. Together with the validity dates, the wind park can also employ a forecasting algorithm to notify other nodes in the grid of the changed energy production beforehand.

Since the protocol uses virtual connections, the layout of the grid does not have to correlate with the layout of the communications network. That way, a simple WAN connection can be used and maintained while the virtual connections still allow all grid nodes to keep the same, consistent logical view on the energy grid.

This is important considering the normal current flow in a grid, which cannot be easily directed. Together with the protocol's rule to prefer messages with a lower hop count, a correct implementation of this protocol steers node behavior to mimic the energy grid's behavior. Thus, it automatically favors local micro grids and reduces load on the transmission system.

A change of the energy level, may it be imminent or forecasted, cannot go unnoticed since the "match-or-forward" rule applies to every node. Provided that a transport-layer protocol such as the Transport Control Protocol (TCP) [11] or the Stream Control Transmission Protocol (SCTP) [12] provides transport layer safety, a demand or supply message will reach other nodes. This separation of concerns accords with the ISO/OSI stack design principles.

However, the "match-or-forward" rule has one corner case where it can lead to an endless amplification in the number of messages currently circulating in the network: When none of the nodes can match the demand. In this case, a message must be discarded after a certain amount of time. This is done based on the TTL (time-to-live), which is a common rule also applied to IP packets. The initial TTL shall be user-configurable and must be high enough for a node's message to reach a destination which can answer the request.

The simplest layout involving a consumer and a producer exists when producer and consumer are directly connected. In this simple case, when the consumer demands more energy and the producer can match the request, it will answer with a message indicating the offer of the required amount of energy. This basic behavior is dictated through the requirement that each offer or demand message must be matched or forwarded.

On this basis, more complex layouts can also be tested. Consider the still rather simple, circular grid layout in Figure 4.

The consumer (labeled C), the factory pictured above, requests more energy. The request reaches the northern of the two transformers, which cannot influence the energy balance, but has to forward it based on the "match-or-forward" rule. The request message is copied and sent along both links, since a message must be sent on all links except the receiving one when forwarding ("forwarding" rule). The message reaches the two wind turbines P1 and P2 simultaneously, meaning that link latencies are ignored in this example.

If each one of them can provide half the amount of energy requested in the time frame it was requested for, it will send an offer and re-send the modified request. All four messages will then reach the conventional power plant, P3. This will also answer, in this example with the total amount of energy requested. The five messages on the wire will eventually reach

 ISBN: 978-1-61208-292-9 

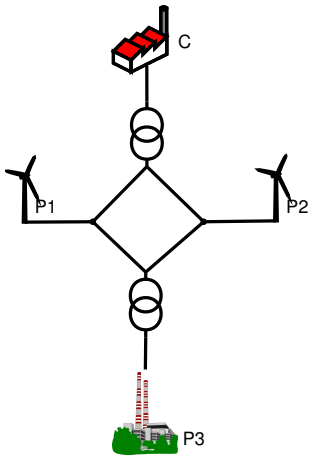


Fig. 4. A simplified, circular grid layout

P1, P2 and P3 again. By examining the message ID, which will remain the same even if the request was previously modified ("same-ID" rule), they can re-identify the message and will remain silent ("no-duplicates" rule).

The answers will eventually reach the requester, C. It will examine the offers and choose those made by P1 and P2 over the offer made by P3 since the message's hop counts are lower. This way, local renewable energy sources near to a requester are automatically preferred. Finally, the *Offer Accepted Notifications* are sent out, reaching P1, P2, and P3. Since P3 doesn't find its offer listed in the acceptance notifications, it will refrain from powering up later. As the last step, P1 and P3 acknowledge the process by sending out their *Offer Acceptance Acknowledgements*. The demand-supply calculation algorithm C has started when the need for energy became apparent then stops.

Please note that in both examples, the power line loss has been ignored. Such power line losses can either be described using the Connection attributes, or by creating "smart lines", which then become nodes of their own. Due to the "match-or-forward" and "forward" rules, the line loss is simply subtracted from the message's Power value.

It can easily be noted that neither the protocol's semantics nor its basic representation make an effort at security: We do not propose a message encryption or impose an authentication algorithm.

Partly, this is intentional: All nodes in the network shall be treated equally. However, using a public transport such as the Internet of course requires additional security measures to be taken. Since this protocol would be part of the ISO/OSI stack at level 7, we deem it sufficient to use the encryption/authentication facilities this stack already offers, such as IP Secruity [13] (IPsec).

IPsec offers the ability to create a public key/certificate chain infrastructure. Certificates would authenticate nodes in the same way as they authenticate an online-banking web server today. Certificate revocation lists can be used to block compromised nodes within the smart grid.

## VII. Conclusion and Future Work

In this paper, we have outlined the ground rules of a messaging protocol which allows proactive communication of nodes in the Smart Grid. The goal was to enable each node, consumer and producer alike, to communicate their changing needs or offer for energy, while allows other nodes to pick up these pieces of information and act accordingly. This forms the basis of a tighter integration of renewable energy sources and allowing them to become more dominant in the energy mix, even though those sources might be dependent on external sources of influence outside our control, like the weather.

Proceeding further from that basis, we are going to propose an agent architecture for the Smart Grid nodes that adheres to this protocol. It will use the protocol semantics to implement a solver for the demand/supply calculation process.

We are also going to demonstrate how this protocol can be implemented using a standard IP network.

## VIII. Acknowledgments

This paper has been created as part of a cooperative doctorate programme between TU Bergakademie Freiberg and Wilhelm Büchner Hochschule, Pfungstadt.

The author E. Veith would like to thank Nike C. Schmidt for her continuous organizational support.

## References

[1] J. R. Roncero, "Integration is key to smart grid management," *CIRED Seminar 2008 SmartGrids for Distribution*, no. 9, pp. 25–25.

[2] D. Crockford, "RFC 4627 - The application/json Media Type for JavaScript Object Notation (JSON)," IETF RFC, IETF, Tech. Rep. [Online]. Available: http://tools.ietf.org/html/rfc4627 [Retrieved 2013-05-26]

[3] S. Bradner, "Key words for use in RFCs to Indicate Requirement Levels," Internet Engineering Task Force, Fremont, CA, Tech. Rep. RFC 2119, March 1997. [Online]. Available: http://www.ietf.org/rfc/rfc2119.txt [Retrieved 2013-05-03]

[4] M. Kosakada, H. Watanabe, T. Ito, Y. Sameda, Y. Minami, M. Saito, and S. Maruyama, "Integrated substation systems-harmonizing primary equipment with control and protection systems," in *Transmission and Distribution Conference and Exhibition 2002: Asia Pacific. IEEE/PES*, vol. 2, 2002, pp. 1020–1025 vol.2.

[5] "Iec 61850 for power system communication."

[6] N. Higgins, V. Vyatkin, N.-K. Nair, and K. Schwarz, "Distributed power system automation with iec 61850, iec 61499, and intelligent control," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 41, no. 1, pp. 81–92, 2011.

[7] G. Zhabelova and V. Vyatkin, "Multi-agent Smart Grid Automation Architecture based on IEC 61850/61499 Intelligent Logical Nodes," *IEEE Transactions on Industrial Electronics*, no. 5, pp. 2351–2362.

[8] P. Leach, M. Mealling, and R. Salz, "Rfc 4122: a universally unique identifier (uuid) urn namespace," *Proposed Standard, July*, 2005.

[9] K. Thompson and D. M. Ritchie, *UNIX Programmer's Manual*. Bell Telephone Laboratories, 1975.

[10] S. Deering and R. Hinden, "RFC 2460 - Internet Protocol," 1998. [Online]. Available: http://tools.ietf.org/html/rfc2460 [Retrieved 2013-05-14]

[11] J. Postel, "Rfc 793: Transmission Control Protocol," 1981. [Online]. Available: http://tools.ietf.org/html/rfc793 [Retrieved 2013-03-19]

[12] R. Stewart, "Stream Control Transmission Protocol," RFC 4960 (Proposed Standard), Sep. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4960.txt [Retrieved 2013-05-13]

[13] S. Kent and K. Seo, "Security Architecture for the Internet Protocol," RFC 4301 (Proposed Standard), 2005. [Online]. Available: http://www.ietf.org/rfc/rfc4301.txt [Retrieved 2013-06-22]

 ISBN: 978-1-61208-292-9 