Secure Communication in a Distributed Load Management System

The transition from the traditional power generation using centralized power plants toward smaller, distributed facilities raises the importance of load management. To establish a balance between production and consumption, they have to be regulated in a dynamic fashion. Intelligent, networked systems are necessary to achieve that goal. A considerable part of the network communication is likely to be conducted over the public Internet, therefore secure communication is an important aspect. In this paper, we discuss the problems of existing systems that do not fulfill current and future security requirements, and we develop an approach that allows secure communication using the WebSocket protocol even when devices considered insecure are involved.

Peter Gysel, Matthias Krebs, Stefan Röthlisberger | peter.gysel@fhnw.ch

A lot of companies in Switzerland are already using *load manager* (LM) devices to optimize their power consumption. Their primary use case is to turn off non-critical appliances temporarily in order to prevent short peaks in electrical load (peak shaving). They thereby save money, because the price per kWh is based on the highest load peak within a 15-minute time slot.

Since such LM devices are widely deployed, they can be interesting to energy distributors. As the power grid has been constructed to support the highest peak power consumption possible, a system smoothening those peaks could potentially save a lot of money. It could also be used to help building autonomous communities. In which power is produced in small distributed power plants and consumed locally.

The Smart Grid, a modernized electricity grid that uses information and communications technology to gather and act on information, is an ongoing topic and is approached cautiously because of the high risk involved interconnecting the electricity grid for monitoring and control. It is more of an evolution than a revolution and thus the integration of legacy devices has to be considered. We are going to show how we integrate a LM device into a load management system to help an energy distributor stabilizing its power grid.

There are different scenarios where load management is used, as Figure 1 shows. Each scenario imposes different constraints on how the load can be managed. Some depend heavily on the time of the day, others just need to run for a certain amount of time per day. Different devices of several generations are being used and lots of them can be considered legacy. Nevertheless, they still work and their configuration is fine-tuned to the needs of the customer.

Legacy devices with network functionality (e.g. metering devices) often use aged communication protocols such as Modbus [1]. These protocols have been built for efficiency and robustness, but they do not offer any kind of security and thus are unsuited for communication over the Internet. Updating the devices is no option either, as they do not have the computational resources needed to implement any kind of encryption. They were designed in a time when the Internet, as we know it today, did not exist. Up to now, the problem has often been approached by tunneling all the communication through a *virtual private network* (VPN). A VPN interconnects networks which intro-

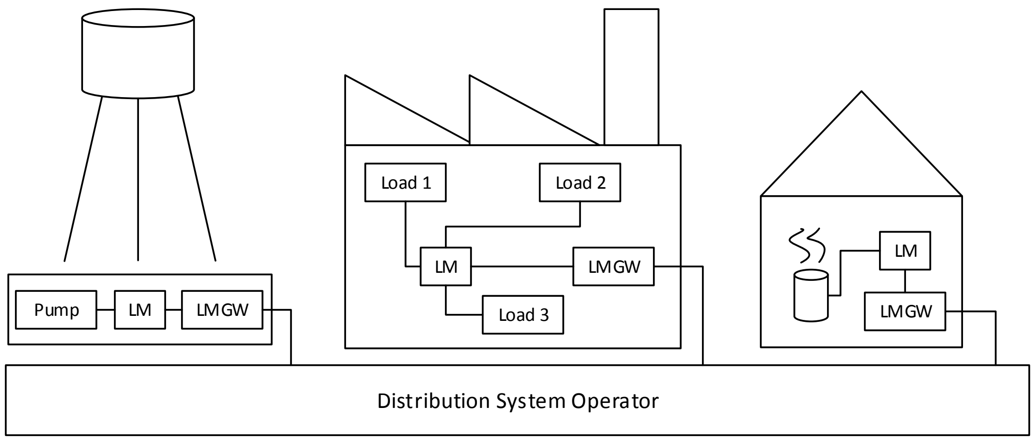

Figure 1: Overview of how our gateway can be used with different load management devices

duces additional security challenges and will be discussed later in this paper. As an example, such a solution is offered by ADS-Tec [2].

A new approach is the use of a custom secure communication channel designed specifically for load management systems, with regard to the requirements of today's IT security. The *load manager gateway* (LMGW) we develop is an embedded device that incorporates this concept. It is installed in company networks containing legacy LMs and connects to them. Unlike the LM, the LMGW is allowed to connect to the Internet.

Our scenario is as follows: A central management server, located at the *distribution system operator* (DSO), issues commands over an encrypted and authenticated channel to the connected LMGWs, which relay them to the LMs (Fig. 1). The response from the LM device is then sent back to the DSO over the same channel.

This paper focuses on the challenges of deploying a LMGW device in a corporate environment and the prototype implementation of the communication between the LMGW and DSO.

Related Work

The German Federal Office of Information Security has worked out a protection profile for a smart meter gateway [3]. The document describes the security objectives and the requirements for that. The gateway connects to the Internet and is connected to one or more smart metering devices. The goals of the gateway are protecting the privacy of the consumers, ensuring a reliable billing process and protecting the Smart Grid as a whole. As such it collects, processes, and transmits data from the connected meters. An important requirement is the usage of a security module (e.g. a smart card) [4] that provides various functions related to encryption and authentication. Another important premise required by the protection profile is that all devices communicating with the gateway have to use encryption and mutual authentication. This does not allow for legacy devices without such functionality to be incorporated into the Smart Grid. The scenario of controllable systems (in terms of load management) is mentioned but not discussed further.

The design of a secure access gateway for home area networks is considered in [5]. Their article focuses on secure, real-time remote monitoring and control of managed devices using a smartphone. The proposed system architecture also enables the managed devices to send alarms to the smartphone. The emphasis is on physical layer security of wireless networks (e.g. OFDM and GSM) and capacity challenges therein.

A system which controls connected loads based on prices is introduced in [6]. The authors use a laptop to monitor energy prices and use this information as a basis for a small localized demand response system.

In [7] the authors provide a threat analysis for advanced metering networks and formulate requirements based on those threats. In a prototype implementation they use a Trusted Platform Module (TPM) for attestation and the Xen Hypervisor [8] to allow multiple virtual machines on a host to isolate different applications from each other. Their focus lies in the isolation of different applications running simultaneously on an advanced meter to preserve their integrity and confidentiality and the attestation of the software running on it.

Lots of papers can be found about cyber security in Smart Grids which discuss security-related issues and technologies that can be used. But they do not actually specify how to apply them in practice or describe a prototype implementation.

Challenges

When thinking about implementing an LMGW, the most important aspect is the communication with the DSO (Fig. 2). In such a communication an LMGW acts as an intermediary between an LM and a DSO. A DSO sends commands to an LMGW, the LMGW then gets or sets the state of the LM. The response from the LM is sent to the DSO via the LMGW.

We use the following five requirements to drive our evaluation. They are chosen to allow for seamless adoption into a corporate environment.

1. Secure communication channel using TLS (Transport Layer Security) with end-to-end encryption and mutual authentication.
2. No obligation for any special firewall setup, especially not opening a port to allow Internet traffic into the company's network.
3. The ability to pass through intermediary proxy servers.
4. Performance, because several messages may be sent to a LMGW every second.
5. A persistent bidirectional connection (optional).

In the following paragraphs, the requirements are explained further with their respective problems to our solution domain.

To protect the data exchange between an LMGW and a DSO the connection needs to be encrypted. That means no device between an LMGW and a

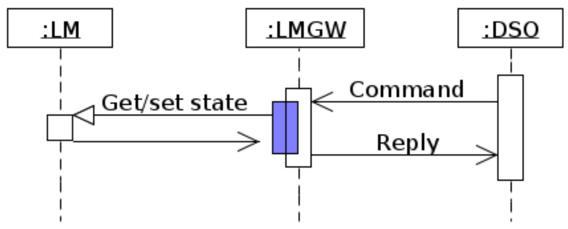

Figure 2: Communication between the DSO and a LM using the LMGW as an intermediary

DSO should be able to decrypt the traffic (end-to-end encryption).

Another important aspect of a system like a DSO is that the operator needs to be able to see who is connected. Using a proper method of authentication each LMGW has its own identity. This also allows revoking access from a specific LMGW and ensures the continuity of the system as a whole without the revoked device.

Many companies employ restrictive policies regarding network security. This means that they are not willing to lessen their security policy just for one device. Therefore, a LMGW has to adapt to this situation and has to offer a way to play nice with a restrictive network security policy. Such companies commonly use at least a firewall which by default blocks all inbound traffic from the Internet. More restrictive firewalls may even be configured to block most or all TCP/UDP ports from the company's network to the Internet, except for those used in the Web (HTTP, HTTPS). This means that a DSO cannot be the initiator of the connection.

Larger companies utilize proxies to cache, restrict and filter access to the Internet. They are located between the client and the server and may cause problems when communication protocols other than HTTP are being used, for which the proxies have not been designed.

Some companies use proxy servers featuring *deep packet inspection* (DPI), which inspect even encrypted traffic by decrypting, analyzing and re-encrypting the packets prior to forwarding. They provide a custom *certification authority* (CA) that is trusted by the clients in the company network, and instead of presenting clients the correct server certificate when they open a secure web site they present a self-signed certificate with the same server name. Because the clients trust the CA of a proxy, they accept its certificate. Communication between a client and a proxy is encrypted, but the proxy is able to decrypt the data. After analyzing the request of a client, the proxy then forwards it to the actual server, now using the correct server certificate for encryption. What the DPI system actually does qualifies as a man-in-the-middle attack. This is particularly a problem if we require end-to-end encryption for a connection.

Evaluation of Transport Protocols

When talking about the transport protocol, we mean the ISO/OSI application layer protocol used to transport the application protocol from one endpoint to another. It provides a higher level API to communicate using arbitrary messages between the two communication partners. Choosing an appropriate transport protocol for the communication between a DSO and an LMGW (Fig. 2) is crucial. We therefore analyzed some well-known approaches. We were looking for an efficient protocol which can be used without compromising the security of the company that deploys our solution. In the following sections, we discuss several protocols and their applicability to our problem domain. We compare them in terms of efficiency and how well they meet the previously discussed challenges. An overview of the discussed protocols and their fulfillment of the requirements can be found in Table 1.

HTTP Polling: HTTP is a request-response based protocol. We are looking at two approaches usually employed to communicate with a server when asking it for information. *HTTP Polling* continuously sends requests to the server in a fixed time interval. The server immediately responds either with new information or an empty response (Fig. 3a). The second approach is *HTTP Long Polling*, which is just a slight deviation from *HTTP Polling* (Fig. 3b). The difference is that *HTTP Long Polling* does not send empty responses back to the client, but instead keeps the connection open until the request can be answered with new information.

Server-Sent Events are currently being standardized as part of HTML5 by the W3C [9]. It offers a light-weight approach to push messages from the server to the client. The client initiates the connection which is basically an HTTP GET request with the *Content-Type* header set to *text/event-stream*. The server keeps the connection open and sends (pushes) multiple messages to the client until the connection is explicitly closed by the server or the client.

WebSocket is a bidirectional, full-duplex protocol using a single socket for communication (Figure 3c). It is standardized in RFC6455 [10] and is a W3C working draft [11]. A *WebSocket* client establishes a connection using the HTTP *upgrade*

Transport Protocol	efficiency	TLS	no firewall setup needed	proxy server pass-through	bidirectional
HTTP Polling	-	✓	✓	✓	-
HTTP Long Polling	-	✓	✓	✓	-
Server-Sent Events	-	✓	✓	(✓)	-
WebSocket	✓	✓	✓	(✓)	✓
Raw TCP Socket	✓	✓	(✓)	-	✓
VPN	(✓)	✓	(✓)	(✓)	✓

Table 1: Comparison of protocol capabilities

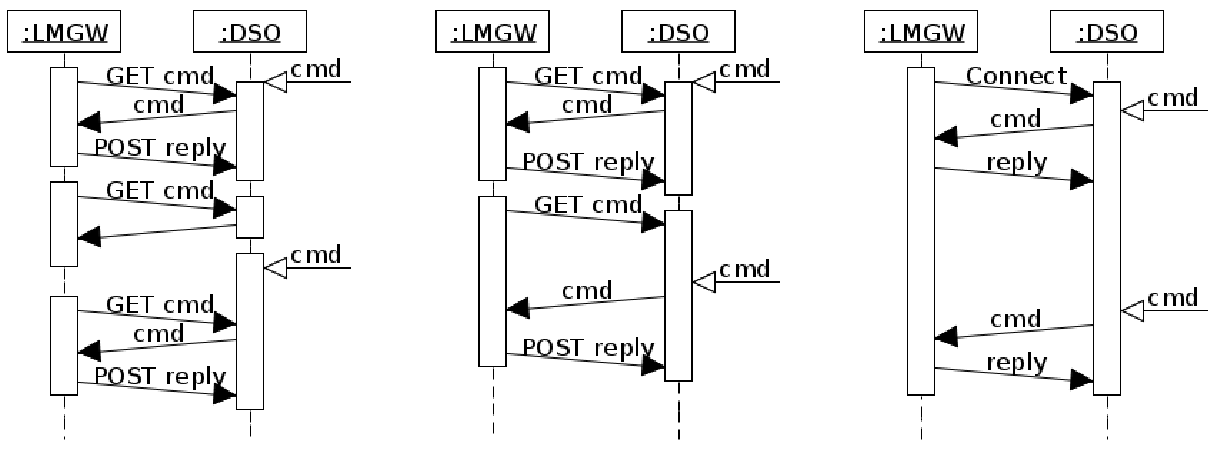

Figure 3: Communication between a distribution system operator (DSO) and a load manager gateway (LMGW).
a) HTTP Polling, b) HTTP Long Polling, c) Persistent connection

header during the initial handshake. The HTTP connection is subsequently upgraded to *WebSocket*. After that, it is no longer considered an HTTP connection. When established, the connection is kept open until a participant closes it explicitly.

Raw TCP Sockets are not a transport protocol as we defined it earlier, but we have included them in our evaluation. An established TCP connection provides a bidirectional full-duplex communication channel (Fig. 3c).

Virtual Private Networks (VPN) offer an easy way to interconnect remote networks through a bidirectional tunnel. All traffic between the two endpoints is encrypted and authenticated. A client connected to a *VPN* is logically in the same LAN as the server.

Transport Layer Security

The *Transport Layer Security* (TLS) protocol ensures privacy, data integrity, and authentication using public-key cryptography. By obfuscation of the transmitted data through encryption, privacy is guaranteed. *Message Authentication Codes* (MAC) are used to detect message tampering and forgery. All of the protocols in our evaluation support TLS since it is an additional layer on top of TCP and all our protocols use TCP under the hood.

Firewall

Assuming that a firewall allows outgoing connections on all ports from the company's network to the Internet, none of the described protocols will have problems. In contrast, if the firewall is restricted to only allow Internet access over HTTP ports (TCP port 80 and 443), which is likely in many company networks, a raw TCP socket configured for a different port cannot be used. The other transport protocols use the standard HTTP ports by default, thus it is not required to open any additional ports on the firewall. If a protocol uses these ports but is in fact not based on HTTP, it might be blocked by a packet-inspecting firewall. The *WebSocket* protocol circumvents this problem by using a HTTP request to initiate the connection and subsequently uses HTTP Upgrade to switch the actual protocol.

SSL-based VPNs like OpenVPN can operate on port 443, but other technologies like IPsec require a dedicated port to be opened on a firewall and sometimes even enabling a „VPN pass-through" option (violates requirement 2).

Proxy Servers

Figure 4 shows that using *WebSocket* without encryption is likely to fail if a proxy server is involved, especially since a proxy server may close long-lived connections to avoid keeping a connection to an unresponsive HTTP server open [12]. By using TLS the intermediary proxy should not interfere with the connection.

By using TLS with an explicit proxy, the protocols using HTTP (*Server-Sent Events* and *WebSocket*) will issue an HTTP CONNECT. This method establishes a tunnel through the proxy. If the proxy allows the CONNECT, a connection can be established. Because a *raw TCP socket* and all *VPN* solutions do not issue an HTTP CONNECT, they will not get through an explicit proxy server.

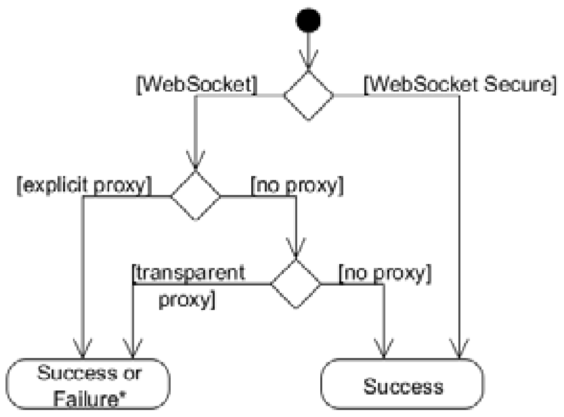

Figure 4: Shortened proxy server traversal decision tree for WebSocket [12] (*depends on the proxy's behavior and configuration)

Performance

To communicate using *HTTP Polling* in compliance with requirement 2, an LMGW would continually poll the DSO with HTTP GET requests for available messages. When a message is available on the DSO, it is included in the body of the HTTP response for the LMGW. The subsequent response from a LM is sent back to the DSO with a HTTP POST message. Another drawback of using HTTP Polling is that a message waits on the DSO to be fetched in the time between the poll requests.

When using *Server-Sent Events*, a DSO can push messages to an LMGW, because there is a persistent uni-directional connection from the DSO to the LMGW after the LMGW has initiated the connection. Nevertheless, the LMGW has to use additional HTTP POST messages to send data back to the DSO, since *Server-Sent Events* do not offer a bidirectional channel.

Using *HTTP Polling*, every POST request (i.e. polling attempt) requires establishing a new connection over TLS. A connection secured with TLS requires two additional roundtrips between the sender and receiver to exchange the certificates and negotiate the encryption. By using *Server-Sent Events* the overhead can be reduced but still requires creating a new TLS connection for every POST request. Both protocols have considerable overhead compared to a protocol offering a persistent bidirectional connection (requirement 4).

The *WebSocket* protocol and *raw TCP sockets* both offer a bidirectional, full-duplex and persistent communication channel. Therefore, both protocols use the available bandwidth responsibly, given the fact that the TLS overhead is concentrated at the time of the connection initiation.

A *VPN* connection is persistent but introduces additional traffic because it establishes a virtual network over the Internet. By using SSL-based solutions like OpenVPN all TCP/IP packets are encapsulated by the *VPN* for the transport over the Internet. The *VPN* package has a TCP and IP header and encapsulated inside is the actual package, again with TCP and IP headers.

An important aspect for the performance of a communication channel is latency. In [13] the authors compare *HTTP Polling*, *HTTP Long Polling* and *WebSocket* without TLS. The latency of polling is measured 2.3 to 4.5 times higher than with *WebSocket*. *HTTP Long Polling* has achieved both lower and higher latency in comparison to *WebSocket*, depending on the situation. Over the longest distance (Canada to Japan), the average latency of *WebSocket* is 3.8 to 4.0 times lower compared to HTTP Long Polling.

Another important performance indicator is throughput. In [14] the authors compare *HTTP Polling* with *WebSocket* in terms of overhead produced by the protocols used without TLS. The HTTP header they use is 871 bytes long, and a *WebSocket* message has an overhead of 2 bytes. They test 1000, 10000 and 100000 simultaneous requests per second to illustrate the difference. At 10000 requests per second *HTTP Polling* uses 66 Mbps, compared to 0.153 Mbps using *WebSocket*.

In a comparison between *HTTP Long Polling*, *WebSockets* and *raw TCP sockets*, the TCP socket was always faster and had more throughput than the others [15]. The larger the payload was, the larger the difference became.

Decision

Applying the requirements to the evaluated protocols, the *WebSocket* protocol fulfills the most of them as shown in Table 1. It is efficient and offers a bidirectional communications channel that the *HTTP*-based protocols do not. By using a *VPN*-based solution, the efficiency depends on the technology used. VPNs using IPSec are more efficient than SSL-based VPNs, but usually require that the firewall is configured to allow VPNs to pass through. SSL-based *VPN* solutions do not require the firewall to be configured, but they have a considerable overhead in bandwidth. In comparison to the *raw TCP socket*, *WebSocket* has the advantage to be able to pass through proxy servers and its higher-level API. A *WebSocket* implementation offers an API to send and receive messages. The *raw TCP socket* does not offer such high-level APIs. Besides establishing and closing the connection, even the low-level message has to be defined by the developer. *VPN*s also have another drawback which is not covered by Table 1: Linking whole networks with a bidirectional tunnel might include malicious traffic as happened in 2003 where the SQL slammer worm propagated through the *VPN* [16]. A similar situation could happen if a DSO's endpoint is taken over by a hacker, whereby he could gain access to all networks of connected LMGW operators.

Prototype Implementation

In a project funded by the Commission for Technology and Innovation (CTI)[1] we develop an LMGW, which is capable to securely communicate with a DSO developed by an industry partner. The LMGW device is based on an embedded Linux platform using an ARMv7 processor, similar to the Raspberry Pi. The communication based on our own XML-protocol is implemented in Java using the Jetty *WebSocket* library running on Java SE Embedded [17].

The *WebSocket* connection is secured with TLSv1.2 [18]. We use the mutual authentication feature of TLS, which is optional in the specification. This allows us to give each LMGW its own certificate that is used for authentication and authorization. The certificates are signed by a cus-

1 CTI Project "Virtuelles Kraftwerk Schweiz" - 15046.1 PFES-ES

tom Certificate Authority (CA) created solely for use in our system. Trusting our custom CA only, we can ensure that man-in-the-middle attacks are not possible because any other certificate presented by an attacker is rejected. All the certificates of the CA can be revoked. This gives us the possibility of denying access to the DSO by stolen devices. For the revocation mechanism we use a *Certificate Revocation List* (CRL) which is served to the LMGW devices and the DSO through a web server. The CRL contains a list with all revoked certificates of the CA. This mechanism adds a potential point of failure to the system because the LMGWs and the DSO need to access the CRL in order to verify the other endpoints certificate to establish the secure connection. The server that serves the CRL must be accessible over the Internet and is thus vulnerable to denial-of-service attacks.

In a period of more than six months we test the connection between one DSO and several LMGWs. The connection between the LMGWs and the DSO is stable and one LMGW connection uses only a combined mean bandwidth of 43.39 kbps measured for 70 hours during live operation. The DSO sends four messages per second to each LMGW device in average. The mean upload from one LMGW to the DSO is 25.66 kbps and the corresponding mean download is 17.73 kbps. The reason for the discrepancy between the upload and download bandwidth is the response from an LMGW contains more data than the request sent by a DSO.

The unsecured connection between the LMGW and legacy LM devices is a problem our solution cannot provide a solution for. That risk can be mitigated by the company itself by not allowing any connections form the Internet to an LM through the corporate firewall or even separate LMs into their own VLANs.

Conclusions and Future Work

Our concept of using the *WebSocket* protocol in conjunction with encryption is a promising approach for both secure and efficient communication over the public Internet. Using Transport Layer Security (TLS) for end-to-end encryption in combination with revocable certificates for authentication, communication is secure against man-in-the-middle attacks, e.g. message tampering, data manipulation and eavesdropping.

There are only few remaining risks, such as the vulnerability to denial-of-service (DoS) attacks due to CRL checking during certificate validation. We plan to address these issues in future development, for example through the introduction of the *Online Certificate Status Protocol* (OCSP) to validate certificates. It uses a so called OSCP responder that knows the condition of all certificates in a CA. Communication usually takes place over HTTP, implementing the request-response paradigm. If an application wants to verify a certificate it asks an OCSP responder whether the certificate has been revoked. This reduces the overhead compared to the CRL approach, because the client only asks whether the presented certificate is valid.

References

[1] The Modbus organization. The Modbus standard specification. http://www.modbus.org

[2] ADS-Tec Big-Linx Remote Service Cloud using OpenVPN. http://www.ads-tec.de/industrial-it/cloud-big-linx/big-linx.html

[3] Federal Office for Information Security, Germany. Protection Profile for the Gateway of a Smart Metering System (Smart Meter Gateway PP). https://www.bsi.bund.de/DE/Themen/SmartMeter/Schutzprofil_Gateway/schutzprofil_smart_meter_gateway_node.html, Version 1.2, 2013.

[4] Federal Office for Information Security, Germany. Protection Profile for the Security Module of a Smart Meter Gateway (Security Module PP). https://www.bsi.bund.de/DE/Themen/SmartMeter/Schutzprofil_Security/security_module_node.html, Version 1.0, 18 March 2013.

[5] Tongtong Li, Jian Ren, and Xiaochen Tang, Michigan State University. Secure Wireless Monitoring and Control Systems for Smart Grid and Smart Home. IEEE Wireless Communications, June 2012.

[6] Michael LeMay, Rajesh Nelli, George Gross, and Carl A. Gunter, University of Illinois Urbana-Champaign. An Integrated Architecture for Demand Response Communications and Control. Proceedings of the 41st Hawaii International Conference on System Sciences, 2008.

[7] Michael LeMay, George Gross, Carl A. Gunter, Sanjam Garg, University of Illinois Urbana-Champaign. Unified Architecture for Large-Scale Attested Metering. Proceedings of the 40th Hawaii International Conference on System Sciences, 2007.

[8] Xen Project, http://www.xenproject.org/

[9] Ian Hickson. Server-Sent Events, W3C Candidate Recommendation. http://www.w3.org/TR/eventsource/, 2012.

[10] I. Fette, A. Melnikov. The WebSocket Protocol. http://tools.ietf.org/html/rfc6455, Dezember 2011.

[11] Ian Hickson. The WebSocket API, W3C Candidate Recommendation. http://www.w3.org/TR/websockets/, 20 September 2012.

[12] Peter Lubbers. How HTML5 Web Sockets Interact With Proxy Servers. http://www.infoq.com/articles/Web-Sockets-Proxy-Servers, 16 March 2010.

[13] Victoria Pimentel, Bradford G. Nickerson. Communicating and Displaying Real-Time Data with WebSocket. IEEE INTERNET COMPUTING, July/August 2012, pp. 45-53.

[14] Peter Lubbers, Frank Greco. HTML5 Web Sockets: A Quantum Leap in Scalability for the Web. http://soa.sys-con.com/node/1315473, March 2010.

[15] Sachin Agarwal , NEC Europe Ltd., NEC Europe Laboratories. Real-time Web Application Roadblock: Performance Penalty of HTML Sockets. IEEE ICC 2012 - Communication QoS, Reliability and Modeling Symposium, 2012, pp. 1225-1229.

[16] North American Electric Reliability Council. SQL slammer worm lessons learned for consideration by the electricity sector. http://www.utexas.edu/law/journals/tlr/sources/Issue 90.1/Thompson/NAERC Slammer Report.pdf, 2003.

[17] Jetty. http://eclipse.org/jetty/

[18] T. Dierks, E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. http://tools.ietf.org/html/rfc5246, August 2008.