Design and Development of an IoT Gateway for Smart Building Applications

Aditya Nugur[1], Manisa Pipattanasomporn[2], *Senior Member, IEEE*, Murat Kuzlu[3], *Senior Member, IEEE* and Saifur Rahman[4], *Life Fellow, IEEE*

***Abstract*—Due to the depletion of energy resources and increased energy demand, there is an increased focus on the energy consumption and management in buildings. Many building energy management (BEM) software platforms are commercially available to monitor and control energy consumption. These platforms are hosted on the physical hardware within the building, due to which the hardware specifications limit their performance. To address this limitation, cloud technology emerged which facilitates software to be deployed at a remote location that has scalable hardware resources. Conventional BEM software can leverage such a cloud platform to offer scalable and maintenance-free installation. Once hosted on a remote cloud platform, BEM software lacks direct connectivity to building sensors/controllers, hence requires a device to support remote accessibility. Most devices are bound to a local area network and therefore need an additional functional layer on top of its communication stack to perform Network Address Translation (NAT)-Traversal. This functionality is implemented on a scalable software which connects to the devices in the network and acts as a gateway for cloud-based BEM software to access devices in the local area network. Any message sent to this gateway is translated to a respective device protocol. This paper describes the design and implementation of such an Internet of Things (IoT) gateway for a cloud-based BEM system that requires support for BACnet, Modbus, and HTTP RESTful interface devices.**

Keywords*—Building Energy Management, Fog Computing, IoT gateway, Cloud computing, Internet of things.

I. Introduction

With increasing global energy requirements, it is necessary to monitor and manage energy consumption. Residential and commercial customers contribute to 40% of the total power consumption in 2016 [1]. Hence, efficient load management at the level of building scale would lead to significant overall energy saving. Thus, smart building loads which can communicate their power consumption are being developed.

Advancements in recent communication technologies promoted smooth development of such smart devices giving birth to a new paradigm of connected things called the Internet of things (IoT). It is speculated that by 2020, IoT will comprise up to 26 billion interconnected devices [2]. Due to this growing number of smart devices, there is an increasing need for the development of a unified Building Energy Management (BEM) platform. This platform encompasses loads of different protocols and applications, thereby, making it agnostic to the device protocol. Conventional BEM systems are hosted physically on hardware resource within a building vicinity. Due to this deployment, its performance is always limited by the installed hardware capability and cannot be considered scalable. Such hardware constraints defined problems are a prevalent problem in the domain of software engineering. For supporting scalability, a software solution must have access to scalable physical hardware resources along with high maintenance standards. This led to the advent of a cloud computing technology in which computation and software maintenance is taken care at a remote facility from where the data is retrieved. Leveraging on this technology, hosting BEM solution on a cloud service upgrades conventional BEM software in following aspects

1) *Scalability*- Hosting on cloud platform, BEM solution has scalable access to hardware resources. BEM software can monitor and manage any number of devices in building vicinity by scaling on demand. Hence, a single instance of cloud-based BEM solution can support multiple buildings.
2) *Remote accessibility*- When opted to host on public cloud platform, hosted BEM solution offers accessibility to dashboard multiple buildings onto single screen which can be administered from anywhere on planet.
3) *Ease of deployment*- A cloud based platform can be deployed without any physical intrusion. Support to new device can be added by modifying source code on cloud. Additionally, once deployed, software availability and maintenance is taken care by cloud provider.
4) *Disaster recovery*- With growing software vulnerability concerns, by hosting BEM software on cloud platform, it can harness highest quality of security expertise and provides direct access to the source code to manage software updates and rectify vulnerabilities.
5) *Data storage and analysis*- energy consumption statistics are crucial to determine faulty environment or

[1] Aditya Nugur received his M.S. degree in Electrical and Computer Engineering from Virginia Tech, VA, USA. He is now with KE2therm solutions Inc., MO 63090 USA (e-mail: aditya32@vt.edu).
[2] Manisa Pipattanasomporn is with Smart Grid Research Unit (SGRU), Department of Electricity Engineering, Chulalongkorn University, Thailand, and an adjunct professor at Virginia Tech – Advanced Research Institute, Arlington, VA 22203 USA (e-mail: mpipatta@vt.edu).
[3]Murat Kuzlu is with Old Dominion University - Department of Engineering Technology, Norfolk, VA 23529 USA (e-mails: mkuzlu@odu.edu).
[4]Saifur Rahman is with Virginia Tech – Advanced Research Institute, Arlington, VA 22203 USA (e-mail: srahman@vt.edu).

optimize building energy consumption. This requires aggregation of huge amount of datasets (in Gigabytes) and application of resource intensive algorithms. Cloud services being data centers with large storage spaces are necessary to facilitate this advantage. Additionally, advancements in machine learning domain enable real time control over building load profiles. As these algorithms get fine-tuned, data collection and analysis may become a mandatory need.

6) *Smarter grid*- Being capable of hosting multiple buildings on a single platform, a cloud-based BEM solution can have varied data inputs from an entire locality, giving greater insights to power generating plants and hence can efficiently manage demand and promote smarter pricing scale. This feedback loop on large scale paves way towards smart cities.

Hence, a cloud-based BEM software can extend the horizon of conventional BEM system offerings. Although the cloud architecture could streamline BEM software adoption, it still suffers from a fundamental flaw, i.e., the feasibility of the cloud solution. Although recently developed smart devices are adaptive to the cloud architecture by supporting remote accessibility, the majority of legacy devices are bounded within a local area network.

For a cloud-based BEM software to have comprehensive control on all devices in a building, each device in the building automation space requires implementing NAT-Traversal as an additional layer on top its communication stack. Featuring this functionality, proxy software can be utilized which runs on a local area network and performs NAT-Traversal on behalf of all local devices. Once traversed through the router, the proxy software connects to the cloud-based BEM software and serves its requests. Based on the received request, this software performs protocol translation and relays the request to the device in the local area network. Hence, this proxy software performs gateway functionality and can be recognized as an IoT gateway. This IoT gateway must be scalable to support any number of devices concurrently, and it is required to support a varied number of protocols.

A number of gateway-related papers [3-20] were published which focused on the gateway just as a protocol translator within the network are published. A brief overview of an IoT gateway and its applications in terms of commercial deployment were published in [3]. This paper discussed various message patterns in the IoT gateway context. The gateway acts as middleware to perform communication with multiple devices in the network. In [4] authors architected a smart grid BEM solution which can monitor, analyze, and control devices. Authors in [5, 6] proposed various IoT gateways to integrate different protocols, such as ZigBee, GPRS, Bluetooth. Authors in [7] suggested an architecture for micro service-based middleware aimed at connecting heterogeneous IoT devices. However, all the mentioned IoT gateways in [5, 6, 7] were not flexible and customizable for different applications. To address this issue, many multi-protocol scalable integration architectures were developed [8, 9]. However, all these architectures were not targeted to work in conjunction with cloud-based BEM software.

With the advent of cloud computing, significant changes took place in the way software architecture is designed. These approaches were applied in the development of BEM software as well. In [10], an IoT architecture based on OPC.NET specification was developed. However, it addressed only devices which follow OPC.NET specifications and do not support RESTful API interfaced devices.

In [11] a detailed study was presented which explains how an IoT gateway performs by fitting in fog computing architecture (fog computing is the approach where a part of cloud computation occurs at the edge of the network). In [12] IoT gateways were demonstrated as distributed nodes which perform machine-to-machine communications for discovery and management of connected vehicles. Authors in [13, 14, 15, 16, 17] proposed how fog computing is a more stable architecture than cloud computing.

Today, there are also a number of hardware, software and end-to-end vendors who commercially deploy IoT gateways. Companies such as Dell, Cisco, Intel, etc., provide commercial-scale smart IoT gateways. These IoT gateway products [18] [19] [20] have a wide range of physical input modules from serial connections to wireless connections. However, these IoT gateways require the acquisition of devices made explicitly for the proprietary IoT gateway and targets sensor devices. Additionally, these IoT gateway devices are compatible with only the cloud IoT platforms dictated by the vendor.

In summary, all literature focused on proposing standards and architectural philosophies to integrate either recently developed smart sensor devices or industrial standard specific devices into an IoT platform but not all kinds of devices in the building automation domain. Although some literature proposed such platforms, they were industrially targeted requiring devices to conform industrial standards. Additionally, to the best of authors' knowledge, there was no academic literature on an IoT gateway platform which can work in conjunction with any cloud-based BEM software to have access to devices adhering both legacy protocols and modern plug-n-play protocols.

Addressing this critical knowledge gap, this paper presents the design and development of a commercial scale IoT gateway, targeting integration of legacy protocols along with RESTful API into a cloud-based BEM software architecture. This device acts as a master device polling non-cloud devices in the network and pushes the received data to the cloud-based BEM software. The architecture of developed IoT gateway is scalable in terms of number of protocols supported. Additionally, developed IoT gateway is compatible with any cloud-based BEM software which can host a Web Socket server.

II. IOT GATEWAY FUNCTIONALITIES

Being independently capable of performing device management, an IoT gateway interfaces with cloud-based BEM software to address user's requests. Utilizing such a gateway software, any cloud-based BEM software can connect to supported local devices. Hence, the features an IoT gateway need to support to assist a cloud-based BEM software are:

A. *NAT-Traversal & Secure Communications*

Upon initiation, an IoT gateway must implement NAT-Traversal and maintain a persistent connection with the cloud-based BEM software. This connection must be secure to protect the privacy of building-related information. For secure communications between an IoT gateway and cloud-based BEM software, the following two essential security considerations must be undertaken:

Authentication: Authentication checking implies the communicating devices cross-checking the genuineness of the connected peer. Once authenticated, the entire communications between authenticated pairs is encrypted at the socket level thereby securing any sensitive device monitoring data.

Authorization: Once peers are authenticated, an additional security layer is necessary, where peers cross-check if the connected peer has sufficient rights to access the information for its request.

B. *Handling of Cloud-based BEM Software Requests*

BACnet and Modbus are two widely adopted legacy protocols which are deeply penetrated into commercial building automation space. Recently, a large number of next-generation smart devices are deployed basing on the HTTP RESTful architecture. To ensure comprehensive access to all ranges of building loads, cloud-based BEM software must support these protocols through the designed IoT gateway software. Leveraging on these protocol stacks, IoT gateway software must be capable of addressing following requests from the cloud-based BEM software:

- Discover: The IoT gateway should make protocol specific discovery requests in the network to discover devices once cloud-based BEM software makes a ‘Discover’ request.
- Approve: Polling of discovered devices should be initiated once cloud-based BEM software makes an ‘Approve’ request.
- Control: An IoT gateway should make a necessary control action and return ‘Success’ or ‘Failure’ response whenever it receives a control request from cloud-based BEM software.
- Authorize: Certain devices require physical authentication of devices by pressing a button on the device or by any other means.

Fig. 1 shows the conceptual architecture of the proposed cloud-based BEM software along with an IoT gateway and other cloud devices.

III. SOFTWARE ARCHITECTURE

A software architecture defines the layout or skeleton of the software and thereby, gives the highest level of abstraction of a system. The entire software development is based on Python 3.6.2 as a programming language. Python has been chosen for software implementation because it takes less development time and supports all recent smart devices API. The primary purpose of IoT gateway software is to perform NAT-Traversal and to support communication with devices adhering to different communication protocols.

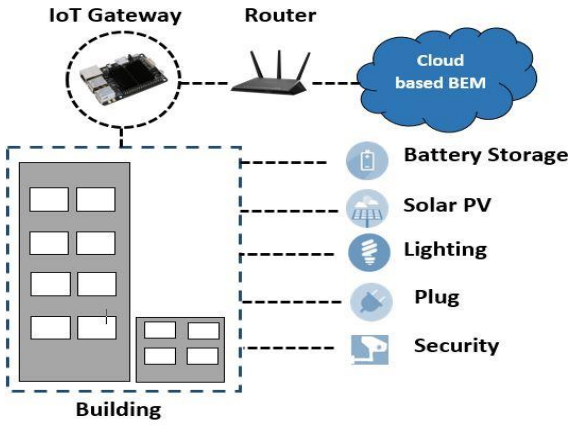

Fig. 1. Layout of cloud-based BEM software along with an IoT gateway.

Fig. 2 shows the architectural layout of cloud-based BEM software along with an IoT gateway. A cloud-based BEM interfaces with the IoT gateway through a web socket server to access devices within building area network. Similarly, it interfaces with cloud devices through their APIs.

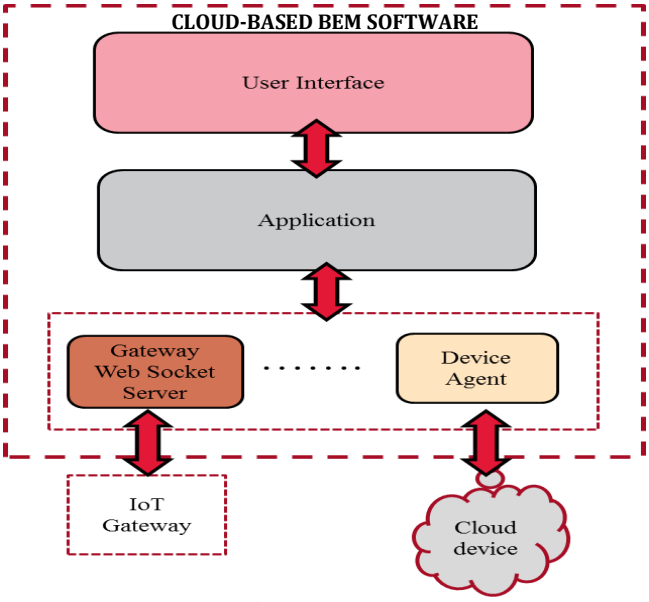

Fig. 2. Cloud-based BEM software architecture along with IoT gateway.

A. *Choosing NAT-Traversal Mechanism*

Since an IoT gateway application is primarily deployed in a network with legacy devices which are behind age-old NAT devices, a NAT-Traversal technique implemented by an IoT gateway must not rely on router’s functionality. Hence, an Internet Gateway Device protocol is not a suitable NAT-Traversal mechanism. On the other hand, TURN, STUN NAT-Traversal techniques relied on an additional server and employed for many to many communication systems. Since cloud-based BEM software and an IoT gateway act as a peer-to-peer communicating devices, a hole punching technique is employed to perform NAT-Traversal. IoT gateway makes an outbound request to the cloud platform and maintains a persistent connection by periodically sending keep-alive messages to ensure the router does not intercept the connection.

B. Concurrency

Under its hood, IoT gateway software should provide services to handle requests from cloud-based BEM software and periodically perform communication with local devices. Requests received by the IoT gateway should be handled without blocking any incoming messages. To address non-blocking request handling performance, IoT gateway software must be architected to perform with concurrency. Concurrency refers to the execution strategy where execution of one part of software does not block execution of another part of software execution. Ideally, parallelism is a much better approach to deal with such unblocked software executions, but this requires more than one processor. In a single processor scenario, concurrency is achieved by scheduling execution of code in a cooperative manner. To support concurrency, software must provide for multiple threads of control. This is accomplished by:

- Using Multiprocessing
- Using Multithreading
- Using User Space threads

Table I compares IoT gateway requirements considering the above three concurrency patterns.

TABLE I
COMPARISON OF VARIOUS CONCURRENCY PATTERNS

IoT Gateway Requirement	Multiprocessing	Multithreading	User Space threads
Optimum waiting time	Moderate	Best favored	Least favored
Uses all CPU cores	Yes	No(Python GIL)	No
Scalable	at least 32 processes for 2 GB RAM	Up to 100's	Up to thousands
Support standard OS blocking calls	Yes	Yes	No, requires monkey patching
No GIL effect	No	Yes	Yes
No deadlocks and race conditions	Yes	No	No

Based on the comparison in Table I, it is implicit that an adequately designed multiprocessing approach would lead to the best architectural design in IoT gateway application. It can make use of dual-core, or four core hardware supplied to it and performs robustly without any deadlock or race conditions. Since the scalability level is up to 30 processes, the software can be designed to run each process as a communication protocol. With this approach, typically on a 1 GB RAM infrastructure, IoT gateway can easily support up to 30 communication protocols and a large number of devices adhering to these 30 protocols. Hence, IoT gateway is developed with a multiprocessing architecture.

C. Actor-based Model

The design of a concurrent system requires choosing reliable techniques for coordinating process execution, data exchange, memory allocation so that the processes execute in minimize response time and maximize throughput. An Actor-based model is one such a concurrency model which defines general rules mentioning how system components behave and interact with each other.

An Actor-based model follows the philosophy of "Everything is an Actor" where an Actor is the primitive unit of computation which can receive a message to its mailbox. When an Actor receives a message, it can perform one of these three functions [21]:

- Create Actors.
- Send messages asynchronously to other Actors.
- Act as per the next message.

Each Actor maintains a private state that can never be changed directly by another Actor. All Actors run on an Actor system platform which takes care of creating the Actors, handling message delivery, managing Actor life cycles and provides appropriate scheduling and concurrency between different Actors. Thespian –an actor based python library– is used for this project [22], which has following advantages [23]:

- *Highly scalable*: Actors can be easily created (and destroyed) as needed.
- *Fault tolerant*: The developed Actor-based model follows "let it crash" philosophy. Every code run inside an Actor (process) is completely isolated, thereby its state never influences any other process. The supervisor, which can be another Actor or Actor system, will be notified when the supervised process crashes and it tries to put it in a consistent state again.
- *Message-Queuing*: Each Actor has a mailbox and a queue with it, where it can receive multiple messages simultaneously, thereby, ensuring the quality of service and non-blocking concurrency.
- *Extensible*: Since Actors are programmed to do specialized tasks, more modules can be added to increase the extensibility of the system. In IoT gateway design, just addition of new Actor adds support to the new protocol.
- *Location independent*: By using socket library for interprocess communication, Actors can be physically distributed throughout the network, but still contribute to the overall functionality.
- *A Higher level of abstraction*: Aspects which are relevant to concurrency such as multi-processing, inter-process communications, scalability, fault tolerance, and recovery, etc., are all handled by the centralized Actor System, thereby promoting an environment to build application code.

IV. SOFTWARE DESIGN

Fig. 3 shows the entire software design of the developed IoT gateway. Architecture as a whole is Actor-based. To support cloud-based BEM, IoT gateway software has been designed to run at least two Actor interfaces. While one is to perform NAT-Traversal and receive requests from the cloud-based BEM, the other is to handle the requests and make local communications. Hence, software design of the IoT gateway considers the efficient design of two interfaces: (a) Client Actor Interface and (b) Handler Actor Interface. These are explained below.

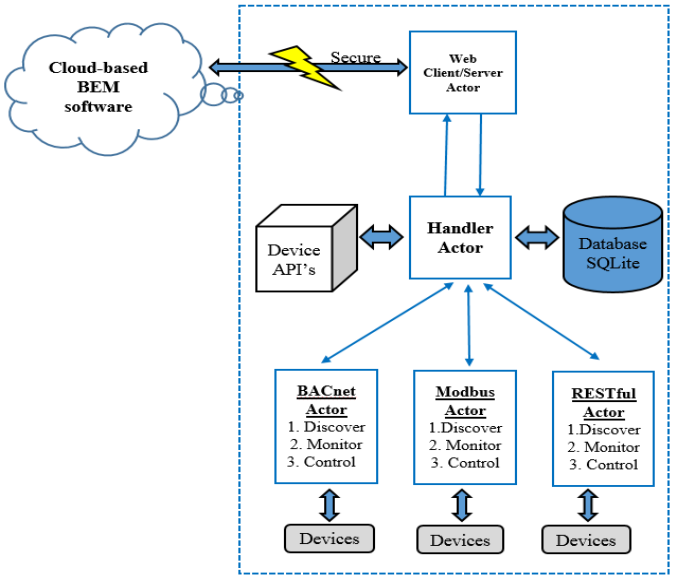

Fig. 3. Software design of the developed IoT gateway.

A. *Client Actor Interface*

This interface requires to perform NAT-Traversal and maintain a persistent connection with the cloud-based BEM interface. Hence, it should implement a cloud protocol. Any communication protocol requires implementation of following interconnection layers: Transport Layer, Session Layer and Application Layer.

1) Transport Layer

The Transport layer defines protocols which are responsible for the reliability and state of connectivity between both peer devices. TCP and UDP are predominantly used transport layer protocols. Selection of the transport layer is application specific. IoT gateway software requires a point-to-point, reliable, long-term connection with the cloud-based BEM software. The TCP protocol is a connection-based protocol which maintains a state of the connection. The UDP protocol is a "fire and forget" protocol, where a sender and a receiver do not establish a connection. Based on IoT gateway software requirements TCP connection is preferred as the transport layer.

2) Session Layer

This layer along with the presentation layer serves as the means to encrypt and secure the communications between peer devices. In addition to encryption, the IoT gateway needs to authenticate that it is connected to a genuine cloud-based BEM. Historically, various versions of SSL/TLS are developed to solve this authentication and encryption problem. TLS 1.2 is most recent security standard, is required to be implemented on the communication between the IoT gateway and cloud-based BEM software. Fig. 4 demonstrates how the developed IoT gateway and a cloud-based BEM implement TLS.

The IoT gateway sends the initial Hello message along with its supported cipher suites, compression, etc., Cloud-based BEM software selects appropriate cipher suites for the session. It sends its selection and its public key. IoT gateway software authenticates the public key of the cloud-based server based on the CA certification. Once authenticated, it generates a new symmetric key which is used for encryption of messages. This key is encrypted using the public key of the cloud-based BEM software and sent to it. Since, it is encrypted using its public key, only the connected cloud-based BEM software can decrypt the message with its private key to retrieve the symmetric key. From now, entire communication is encrypted at both ends using the shared symmetric key.

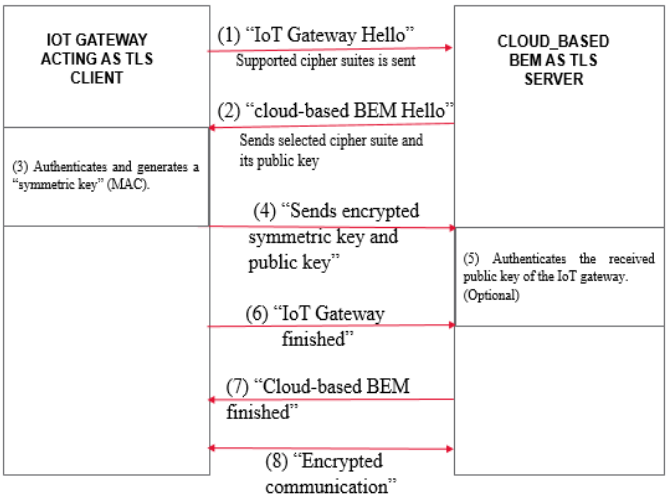

Fig. 4. TLS session between IoT Gateway and cloud-based BEM software.

3) Application Layer

This layer deals with the communication protocol between two services, i.e., the IoT gateway service and cloud-based BEM software. A number of communication protocols emerged recently in IoT domain to interconnect smart devices with cloud IoT platforms. These protocols perform hole-punching and maintain persistent connectivity with cloud platforms. To perform hole-punching, IoT gateway makes an outbound request to the hosted cloud-based BEM software. This request is transferred by the building router which maintains the routing history on a routing table. This request entry on the routing table acts as a temporary unprotected open port in the firewall rules of the router. Once cloud-based BEM software responds to the request, its response, is transferred back to the device based on routing table entry. From now, cloud-based BEM has a bidirectional tunnel with IoT gateway as long as an entry exists on routing table. To ensure IoT gateway is available for any request, it periodically sends keep-alive messages to the cloud-based BEM software. Table II compares various popularly used cloud platform protocols [24].

Protocols can be distinguished based on the architectural pattern, i.e., publish-subscribe pattern and request-response pattern. Publish-Subscribe is prominently used for many-to-many connectivities and requires additional protocol specifications such as topics, hierarchies, etc. Since cloud-based BEM software and the developed IoT gateway service talk to each other as peer devices, request-response architecture suits the application. According to Table II, there are four popularly used request-response protocols, of which three most popularly used protocols are: (1) Restful API; (2) Web Socket; and (3) Constrained Application Protocol (CoAP). These are compared in Table III.

TABLE II
COMPARISON OF VARIOUS CLOUD PROTOCOLS

Protocol	Transport	QoS	Architecture	Security
Message Queuing Telemetry Transport (MQTT)	TCP	Yes	Pub/Sub	TLS/SSL
Constrained Application Protocol (CoAP)	UDP	Yes	Req/Res	DTLS
Representational state transfer (RESTFUL)	HTTP	No	Req/Res	HTTPS
Advanced Message Queuing Protocol (AMQP)	TCP	Yes	Pub/Sub	TLS/SSL
Web Socket	TCP	No	Req/Res Pub/Sub	TLS/SSL
Data Distribution Service (DDS)	TCP/UDP	Yes	Pub/Sub	TLS/SSL
Secure MQTT (SMQTT)	TCP	Yes	Pub/Sub	Own security
Extensible Messaging and Presence Protocol (XMPP)	TCP	No	Req/Res Pub/Sub	TLS/SSL

TABLE III
COMPARISON OF REQUEST-RESPONSE ARCHITECTURE PROTOCOLS

IoT Gateway requirement	RESTful HTTP Protocol	Web Socket	CoAP
Lightweight	Heavy protocol	Lightweight protocol	Lightweight protocol
Maintains state between communication	Stateless protocol	Maintains state	Stateless protocol
Secure communication	Supports TLS 1.2	Supports TLS 1.2	Supports DLS
TCP based	Yes	Yes	No
Long-lived connection	No	Yes	No
Predominantly used for JSON	Predominantly used for hypertext transfer	Predominantly used for JSON, XML data transfer	Predominantly used for JSON, XML data transfer

According to Table III, the Web Socket protocol satisfies all the gateway required features. Unlike HTTP, the web socket connection stays "open" for communication indefinitely. Through this open connection, data is "pushed" to the client in real-time on demand. Hence, web sockets support low latency and full duplex with a single connection. Additionally, web sockets transfer header contents only during communication initiation, after which each payload sent via the socket are framed with only two bytes. This drastically cuts down the size of payload transfers between peer communications and saves bandwidth. Hence, web socket protocol is implemented as a Client Actor by the IoT gateway software. Requests and response payloads between applications are in JSON encoded strings.

B. Handler Actor Interface

Handler is the back-end of the software which handles any request from the cloud-based BEM software. Since any request is addressed to devices in a local network, the Handler should be a middleware between all supported non-cloud protocols and web socket client. To support heterogeneous protocols, IoT gateway can host multiple protocol clients to run concurrently. Each protocol client inherits protocol library to assemble protocol specific request. Alternate method to handle protocol heterogeneity is to generate protocol specific packet at the cloud-based BEM and encapsulate it as a payload to IoT gateway. Such an approach requires cloud-based BEM to support varied protocol libraries, perform encapsulation and specific encapsulation protocol. This limits developed IoT gateway to support specific cloud-based BEM solutions. Having protocol specific requests generated by the IoT gateway, its communication with cloud-based BEM software is structured to be in json format, thereby, promoting IoT gateway to be versatile with any cloud-based BEM solution. Additionally, IoT gateway couldn't handle protocol heterogeneity by encapsulating the packet since it requires devices in the network should support decapsulation.

The Handler must address requests received without blocking itself during request handling, thus, making itself available for any following request from the Client Actor. In this scenario, the Handler interface leverages the Actor-based model and initiate protocol Actors to transfer the request to respective Actors. The Handler hence maps client requests to device API to fetch configuration details and sends the request along with configuration data asynchronously to the appropriate protocol Actor. A protocol Actor upon receiving a request message from the Handler invokes its appropriate method to handle the request. These methods talk to the device by making protocol-specific communications. Currently, developed gateway supports following communication protocols and forks following three Actors:

1) BACnet Actor

The BACnet protocol is specifically designed for building automation and control due to which it is most adopted communication protocol at the building automation level. The BACnet protocol defines each data point in terms of Object Type and Index along with its address, and an identifier for every reference point to be monitored. A BACnet actor simulates a virtual BACnet server and acts as an interface to communicate with BACnet devices in the network. The open-source bacypypes library [25] is used for this purpose.

2) Modbus Actor

The Modbus protocol, being one of the legacy protocols, has very high adoption in VAV's, RTU's, sensors, power meters, thermostats, solar inverters, etc. The Modbus protocol defines requests to possess Function Code, register start address and end address for accessing the device. The Modbus actor simulates a virtual Modbus client and acts as an interface to communicate with Modbus devices in the network. The open-source pymodbus library [26] is used for this purpose.

3) HTTP RESTful Actor

The RESTful API interface has gained popularity lately due to its simple architecture. Many smart plug loads and other controllers developed during the current IoT era provide a RESTful API interface to their devices. These devices are based on the HTTP protocol and provide GET, POST, PUT and DELETE methods to access its resource. The HTTP RESTful Actor makes GET, POST requests on the local network

depending upon the request from cloud-based BEM software. The Python inbuilt requests library [27] is used for this purpose.

A BACnet Actor is developed as an asynchronous client by spawning threads for each request, whereas Modbus and RESTful Actor is designed as an asynchronous client implementing cooperative multitasking. Being cooperatively multitasking both Modbus and RESTful Actors can support any number of devices. Due to GIL on threading BACnet actor starves to run all threads in parallel and hence, the number of devices supported by BACnet actor is limited by monitoring interval. At a monitoring interval of one minute, the developed architecture can support 60 devices per one BACnet Actor.

C. Device Application Program Interfaces (APIs):

Since device-specific API abstracts device information, IoT gateway is developed with structured device APIs with device specific parameters pluggable from a CSV config file. Hence, gateway software supports self-generating API based on input config file fed in the form of CSV. BACnet and Modbus protocol translators take in Object type, property instance, property name, register number, offset, function code, multiplier as input CSV columns whereas RESTful actor takes in URL, restful method(GET/POST), params as input columns. Since, HTTP RESTful protocol is not standardized and have varied implementations among various vendors, leading to different parsing mechanisms, adding support to a RESTful device may require a device-specific python API file. Being standardized protocols, BACnet/Modbus device details can be fed in using a standard CSV template and hence support to any new BACnet/Modbus device can be added by developing a new CSV file with device-specific details structured in it.

D. API Interface for Cloud-based BEM Software

To develop an IoT gateway which is compatible with any cloud-based BEM solution, it should have an application interface which cloud-based BEMs can use to communicate with the gateway. From the cloud-based BEM software's perspective, integrating IoT gateway is equivalent to adding a new device which has Application Programming Interface (API) accessibility to discover, approve, control and authorize, variety of devices located behind the IoT gateway. Hence, IoT gateway is developed with a JSON-encoded API. Any web socket communication from the cloud-based BEM software should have a body which has a JSON-encoded message containing 3 fields namely TYPE, PAYLOAD and AUTHORIZATION_CODE. The content of these fields dictates what the IoT gateway must perform on the local network. The TYPE field can take "Discover", "Approve", "Control" and "Authorize" as entries. Accordingly, the PAYLOAD field has device model names to discover in the case of discover message or unique deviceID on which IoT gateway has to operate in the case of approve, control or Authorize request. Depending on the value of TYPE and PAYLOAD, Handler actor transfer request to respective functions in the protocol handlers. Additionally, AUTHORIZATION_CODE field acts as an authentication mechanism between IoT gateway and the cloud-based BEM software. This JSON defined communication format form as API guidelines of the IoT gateway. Since any cloud-based BEM software is designed to support new devices, IoT gateway can be integrated easily, similar to any other device which has API accessibility. As IoT gateway contains an interface to talk to multiple devices, it can be categorized as device type HUB and unique deviceID in the payload dictates which device to address. Using these guidelines, developed IoT gateway is integrated to widely used open source home automation software- homeassistant, URL: https://www.home-assistant.io/ by adding an ENTITY class of type hub. This entity class hosts a web socket server to which developed IoT gateway connects on boot and provides API interface to devices present in its network. In addition, to further demonstrate the heterogeneity, the developed gateway can also be integrated with pytomation URL: http://www.pytomation.com/ by adding a device a web socket server as an instance.

V. Performance Tests

The developed software was set up in a laboratory environment at Virginia Tech Advanced Research Institute to demonstrate its compatibility with a cloud-based BEM software platform and its performance is evaluated. Since any cloud-based BEM software acts as a client and do not affect the performance of IOT gateway, these performance tests are done with one cloud-based BEM solution. As a proof of concept, the cloud-based BEM platform tested was Building Energy Management Open Source Software (BEMOSS)," URL: www.bemoss.org. IoT gateway software is connected to four (4) kinds of BACnet devices, i.e., a power meter, a lighting load controller, a plug load controller and an ambient light sensor; two (2) kinds of Modbus devices, i.e., a rooftop unit controller and a variable air volume controller; as well as five (5) kinds of RESTful HTTP devices, i.e., three smart plugs and two smart light switches.

A. RAM utilization

The amount of resources consumed by the software under stress determine the performance of the software. Hence, RAM usage of the developed IoT gateway along with CPU utilization is measured in different scenarios. This performance is benchmarked on Ubuntu 16.04 LTS, Intel Celeron® CPUJ1800 @ 2.41GHz with two GB RAM specifications.

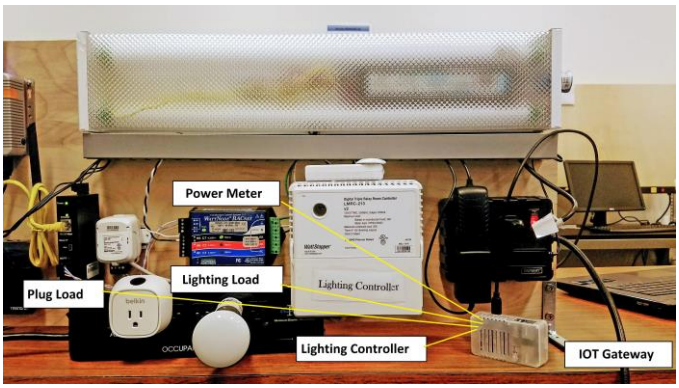

Fig. 6. Part of laboratory test bed at Virginia Tech Advanced Research Institute.

Fig. 6 demonstrates the laboratory test bed. A total of 40 real devices is used and to simulate a large number of these devices, virtual devices are created. The developed software only spawns six (6) processes in total. These processes are:

1) Actor System- This process acts as the platform and parent which spawns remaining processes.
2) Client actor- This process is the Web Socket client which connects to the cloud-based BEM software and maintains a persistent connection.
3) Handler actor- This process takes care of all request handling. Based on request parameters it spawns more protocol actors and transfers request to these actors. Any responses from protocol actors are transferred back to Client actors.
4) BACnet actor- This process runs as a BACnet device and addresses any BACnet specific communications.
5) Modbus actor- This process runs as a Modbus master and addresses any Modbus specific communications.
6) RESTful actor- This process makes device specific local HTTP GET/POST requests.

The process memory usage on a Linux operating system is a complex parameter since it involves virtual memory, transient storage, shared memory, etc. Each process can be identified by a unique number called process Identifier (PID). For analyzing the memory usage by a process, this PID is used to map the process with its memory and CPU consumption details. For this paper, to measure the memory consumption of the developed software, a python package named psutil [28] is used. Table IV shows the RAM consumption of the IoT gateway software at various stress levels. As noticed the RAM consumption remains almost constant even with growth in a number of devices monitored. In certain cases, RAM consumption decreases with devices since operating system optimizes memory usage per process. Additionally, RAM consumption is noticed to have a step increase depending upon the number of protocols supported. Hence, it can be considered that the RAM usage is independent of the number of devices monitored by IoT gateway software. This makes the developed IoT gateway highly scalable in terms of devices it monitors. Considering 35 MB per process, developed IoT gateway software is found to use a maximum of 250 MB of RAM. As the operating system requires additional RAM to run other functional processes, the IoT gateway can robustly run on a 500 MB RAM constrained commercially available hardware such as raspberry pi series.

TABLE IV
RAM CONSUMPTION OF IOT GATEWAY UNDER VARIOUS STRESS TESTS

Gateway Status	Total RAM (MB)trail 1	Total RAM (MB) trail 2	Total RAM (MB) trail 3
Idle	99.65	99.14	94.94
1 Protocol actor running with 10 devices	147.0628	146.523	147.22
1 Protocol actor running with 40 devices	135.8	128.744	132.45
2 Protocol actors running with 10 devices each	168.07	166.099	166.92
2 Protocol actor running with 40 devices each	173.71	166.88	170.64
3 Protocol actor running with 10 devices each	196.28	204.23	215.2
3 Protocol actor running with 40 devices each	197.34	193.05	196.63

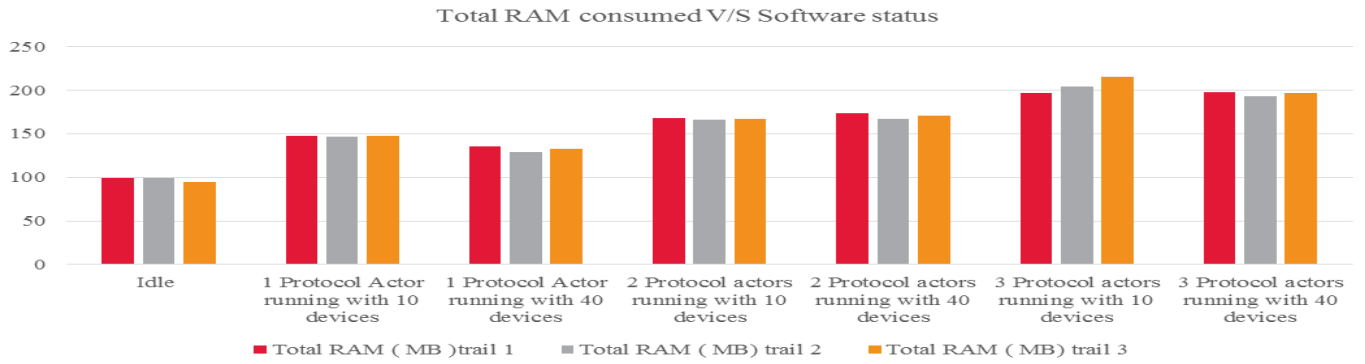

Fig. 5. Comparison of RAM consumption levels in various scenarios.

B. Bandwidth utilization

Bandwidth utilization is dictated by the amount of traffic generated by the IoT gateway software on the hosted network. As a rule of thumb, bandwidth consumption is proportional to amount of data queried and the frequency of polling. Hence, to evaluate this, traffic is generated by the IoT gateway software for reading one data point at a given monitoring period. Since, bandwidth is also consumed based on network environment dependent actions such as establishing the connection, acknowledging the packet transfer, and terminating the connection, for analyzing network independent bandwidth consumption statistics, performance is evaluated by focusing on actual protocol payload. Linux tool iftop and wireshark are used to analyze the actual flow of traffic and an estimate of network traffic is calculated based on each supported protocol.

1) Modbus protocol

Modbus protocol being master-slave topology requires both

request and response transactions for polling data. Number of requests and packet size of response varies based on the Modbus device. For a device which supports multiple registers read and has all required data encoded on consecutive registers, for given monitoring period, software generates a payload packet size of 6 bytes per request and (2n+5) bytes as response for n consecutive registers read. Entire packet size is observed to be approximately 130 bytes (65 request + 65 response) for reading a single holding register. For a device which as data split in non-consecutive register space, IoT gateway generates multiple requests to poll data from all registers. For example: In case of the VAV, status parameters are located at register 0, 159 and 200. Hence, IoT gateway requires 3 requests and corresponding 3 responses per given monitoring period.

2) BACnet protocol

BACnet protocol requires broadcasting of WHOIS message for discovering devices. This increases the network bandwidth for brief period of time. To minimize broadcast traffic, developed IoT gateway generates such broadcast discovery traffic only for 10 seconds upon receiving discover request from the cloud-based BEM software. Developed IoT gateway software utilizes ReadProperty and ReadPropertyMultiple services to poll for BACnet data. For example: Watt Stopper device supports ReadPropertyMultiple, hence a single request with ReadAccessSpecification fetches all Watt Stopper data. For a single ReadPropertyMultiple request, developed IoT gateway generates at least 60 bytes of request and 60 bytes of response traffic, If the querying device doesn't support ReadPropertyMultiple then "n" requests are to be generated to read "n" parameters thus, increasing bandwidth by 60*n bytes for given monitor period.

3) Restful API protocol

Restful protocol is based on standard HTTP request. Developed gateway discovers devices by broadcasting SSDP protocol in the network. The payload of SSDP protocol is vendor specific with minimum packet size of 110 bytes. For querying device parameters, the packet size per request by the IoT gateway is depends on vendor specific API. A basic http request has at least 400 bytes of packet size.

In building applications, the number of network hops is typically two or less and the IoT gateway operates with devices directly accessible within a router. Since all device communications are within the building vicinity, the response time delay is immune to latency caused by the distance between the gateway and devices. In the laboratory environment at 100Mbps with 40 real devices in a network, response delay is found to be less than 12 msec. This meets the requirements for monitoring and control applications in buildings [29]. Additionally, the response delay is found to be significant when the monitoring interval was less than 5 seconds due to which gateway is configured to operate at a minimum device monitor interval of 30 seconds. Hence, although any bandwidth utilized by the IoT gateway software increases the network congestion, the extent of aggravation depends entirely on the network environment dictated by number of devices in the network, traffic etc. Therefore, above estimate of network traffic generated by the IoT gateway benchmarks the bandwidth utilization in absolute terms and any conclusion on effects of bandwidth utilization can largely vary based on the deployed environment.

VI. Conclusion

A cloud-based BEM system is the only solution to address scalability and ease of use issues faced by conventional BEM software. A cloud-based BEM software in comparison with conventional BEM software has unlimited hardware resources and supports desired maintenance-free installation. Hence, migration of BEM software functionalities to the cloud would solve all drawbacks of traditional BEM systems. Although migration of BEM software to the cloud offers accessibility to recently developed smart devices, it has innate issues addressing devices. Hence, an IoT gateway is proposed as a solution to address this issue.

This paper describes the IoT gateway developed based on the most up-to-date technologies to achieve a high rate of concurrency at a scalable note. With the current setup, in total, a maximum of six processes, i.e., the Actor System, the Client Actor, the Handler Actor, as well as protocol Actors for Modbus, BACnet and RESTful, are executed at any given time to process cloud-based BEM software requests. Support to any device adds a bit to memory per process which is very efficient than forking a new process for every new device. Additionally, since each Actor is scheduled by the operating system, they can run on any number of hardware cores, thereby, achieve parallelism. Hence, a lightweight, highly concurrent, non-blocking, scalable framework is designed with a minimum amount of resource consumption.

Performance tests indicated that the amount of RAM usage does not vary even after each protocol supports 50 devices. RAM or resource usage of developed IoT gateway increases with the addition of new protocols as it spawns new process for each protocol. The modularity approach implemented in the design of the developed IoT gateway software facilitates compatibility with any cloud-based BEM software and provides provision to add support to any non-cloud protocol. In addition, support to any Modbus and BACnet device can be added by just writing a comma separated value (csv-based) configuration file. Hence, any Modbus/BACnet protocolled device can be integrated into the developed IoT gateway by building administrator with zero programming experience. These configuration files are built in as system files within the IoT gateway and thereby, act as API interface to support a new BACnet/Modbus device. Restful device APIs are written in the source code. Support to new devices can be added by updating the configuration files and restarting the IoT gateway in case of BACnet/Modbus device or by upgrading the software version in case of Restful API device. Although the developed software is targeted to deploy on single board computers such as odroid/raspberry pi series, etc., with up to 1 GB RAM space, it can be easily ported to embedded Linux operating systems such as OpenWRT/LEDE by writing it as opkg packages.

Future researchers may hence enhance features of this developed IoT gateway by targeting custom developed hardware with private key encrypted to the hardware. The

aspect of performance tests with several hundred or thousand devices in a real building can also be studied. Additionally, this work can be extended to architect distributed IoT gateway architecture where several IoT gateways communicate with each other to cover multiple floors in a building for reliability improvement. The data collected by the gateways can also be subjected to edge analytics to make local instantaneous actions.

VII. REFERENCES

[1] Energy Information Administration, "How much energy is consumed in U.S. residential and commercial buildings [Online]". Available: www.eia.gov/tools/faqs/faq.php?id=86&t=1. Retrieved: December 2017.

[2] Rivera, J., and Meulen, R., "Gartner says the internet of things installed base will grow to 26 billion units by 2020", Dec 2013 [Online]. Available: http://www.gartner.com/newsroom/id/2636073. Retrieved: December 2017.

[3] Sinha, S.R., and Park, Y., Building an effective IoT ecosystem for your Business. Springer, 2017.

[4] Jung, M., et al. "A transparent ipv6 multi-protocol gateway to integrate building automation systems in the internet of things." *Green Computing and Communications (GreenCom), 2012 IEEE International Conference on*. IEEE, 2012.

[5] Emara, K. A., Abdeen, M. and Hashem, M., "A gateway-based framework for transparent interconnection between WSN and IP network." *EUROCON 2009, EUROCON'09. IEEE*. IEEE, 2009.

[6] Zhu, Q., et al. "Iot gateway: Bringing wireless sensor networks into internet of things." *Embedded and Ubiquitous Computing (EUC), 2010 IEEE/IFIP 8th International Conference on*. IEEE, 2010.

[7] Vresk, T., and Čavrak, I., "Architecture of an interoperable IoT platform based on microservices." *Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2016 39th International Convention on*. IEEE, 2016.

[8] Park, K., et al. "Building energy management system based on smart grid." Telecommunications Energy Conference (INTELEC), 2011 IEEE 33rd International. IEEE, 2011.

[9] Desai, P., Sheth, A. and Anantharam, P., "Semantic gateway as a service architecture for IoT interoperability." *Mobile Services (MS), 2015 IEEE International Conference on*. IEEE, 2015.

[10] Ungurean, I., Gaitan, N.C. and Gaitan, V.G., "An IoT architecture for things from industrial environment." *Communications (COMM), 2014 10th International Conference on*. IEEE, 2014.

[11] Aazam, M., and Huh, E.N., "Fog computing and smart gateway based communication for cloud of things." *Future Internet of Things and Cloud (FiCloud), 2014 International Conference on*. IEEE, 2014.

[12] Datta, S.K., Bonnet, C., and Haerri, J., "Fog computing architecture to enable consumer-centric Internet of Things services." *Consumer Electronics (ISCE), 2015 IEEE International Symposium on*. IEEE, 2015.

[13] Perry, T.S., "What comes after the cloud? How about the fog?". IEEE Spectrum: Technology, Engineering, and Science News [Online]. Available:https://spectrum.ieee.org/tech-talk/computing/networks/what-comes-after-the-cloud-how-about-the-fog. Retrieved: Feb 2013

[14] Nikoloudakis, Y., et al. "A fog-based emergency system for smart enhanced living environments." IEEE Cloud Computing 3.6 (2016): 54-62.

[15] Zhu, J., et al. "Improving web sites performance using edge servers in fog computing architecture." *Service Oriented System Engineering (SOSE), 2013 IEEE 7th International Symposium on*. IEEE, 2013.

[16] Arkian, H.R., Diyanat, A. and Pourkhalili, A., "MIST: Fog-based data analytics scheme with cost-efficient resource provisioning for IoT crowdsensing applications." *Journal of Network and Computer Applications* 82 (2017): 152-165.

[17] Stojmenovic, I., "Fog computing: A cloud to the ground support for smart things and machine-to-machine networks." *Telecommunication Networks and Applications Conference (ATNAC), 2014 Australasian*. IEEE, 2014.

[18] Dell Edge Gateway 5000 Series manual [Online]. Available: http://i.dell.com/sites/doccontent/shared-content/data-sheets/en/Documents/Spec_Sheet_Dell_Edge_Gateway_5000_Series.pdf . Retrieved: June 2017

[19] Intel Corporation, "Developing solutions for the Internet of Things" [Online]. Available: http://www.intel.com/content/dam/www/public/us/en/ documents/white-papers/developing-solutions-for-iot.pdf. Retrieved: June 2014.

[20] Evans, D., "The Internet of Things how the next evolution of the Internet is changing everything", Whitepaper, Cisco Internet Business Solutions Group (IBSG), April 2011.

[21] Hewitt, C., Bishop, P. and Steiger, R. (1973). "A universal modular actor formalism for artificial intelligence", In Proc. IJCAI'73 Proceedings of the 3rd international joint conference on Artificial intelligence, pp. 235-245, Stanford, USA, August 1973

[22] Python Actor Model Library [Online]. Available: http://thespianpy.com/. Retrieved: January 2018.

[23] Thespian, "Thespian Python actor system: in-depth introduction", March 2017 [Online]. Available: http://thespianpy.com/doc/in_depth.pdf. Retrieved: January 2018.

[24] Asim, M. "A survey on application layer protocols for Internet of Things (IoT)." International Journal of Advanced Research in Computer Science, 8(3). (2017).

[25] Bender, J., BACpypes, BACnet Python GitHub repository [Online]. Available: https://github.com/JoelBender/bacpypes. Retrieved: December 2017.

[26] RiptideIO, pymodbus, Modbus python GitHub repository [Online]. Available: https://github.com/riptideio/pymodbus. Retrieved: December 2017.

[27] Python requests library [Online]. Available: http://docs.python-requests.org/en/master/. Retrieved: December 2017.

[28] Python ps utils library [Online]. Available: https://pypi.python.org/pypi/psutil. Retrieved: December 2017.

[29] Kuzlu, M. and Pipattanasomporn, M., "Assessment of Communication Technologies and Network Requirements for Different Smart Grid Applications," 2013 IEEE PES Innovative Smart Grid Technologies Conference (ISGT), Washington, DC, 2013, pp. 1-6.