https://doi.org/10.2352/ISSN.2470-1173.2017.6.MOBMU-311

A Study of IoT MQTT Control Packet Behavior and its Effect on Communication Delays

Brian Bendele, David Akopian; University of Texas at San Antonio; San Antonio, Tx/USA

Abstract

An important aspect of enabling Smart Grid and other similar technologies is the development of communication networks that efficiently gather information from remote locations and reliably deliver the data to a world of connected devices. In this context, the concept of Internet of Things (IoT) has emerged and led to the development of several lightweight communications protocols optimized for various environments. This study will focus on the Message Queue Telemetry Transport (MQTT) protocol as used in a remote sensor network setting with the goal of characterizing delay patterns to improve reliability of large scale sensor networks in a publish/subscribe communication environment. Revealing the connection between data size, data collection intervals, network traffic, and delay, an approach for modeling an MQTT network design based on experimentation and inspection of behavior at the packet level is presented, and statistical distributions are explored to develop a method for real-time analysis.

Introduction

Utilities around the nation are increasing their focus on Smart Grid technologies and methodologies to better understand and control the flow of energy across the grid. It is needed to optimize its distribution and predict real time energy needs through an integrated information network [1]. The goal of using IoT in Energy production and distribution chains is to enable utilities to monitor operational and environmental variations that effect grid viability allowing improved stability, efficiency, and durability [1].

A few obstacles to the realization of a smart grid network include interoperability and communication across a multitude of devices and software applications [2]. For this reason, an Open Field Message Bus (OpenFMB) standardization process is launched with the goal of permitting devices and systems to easily communicate with one another in a shared contemporary framework that will utilize publisher/subscriber type of IoT communication and IP networking protocols [3]. OpenFMB spawned from a recent effort by Duke Energy and several working partners to combine IoT technologies with advanced interoperability for the power grid. The result was the development of a framework for a distributed intelligence platform to enable Smart Grid technologies [2]. The Smart Grid Interoperability Panel (SGIP) has taken on the responsibility of further evolving and standardizing the specifications of OpenFMB to offer power systems field devices the ability to exercise a non-proprietary and standards based architecture consisting of Internet Protocol (IP) networking and IoT communication protocols [3].

Many utilities and test beds are participating in OpenFMB simulations, tests, and interoperability demonstrations including San Antonio based CPS Energy as noted in their 2014 public request for partners to test the "grid of the future" [4]. In coordination with CPS Energy, the Texas Sustainable Energy Research Institute (TSERI) and the University of Texas San Antonio (UTSA) have agreed to test multiple smart grid applications and interoperability demonstrations to better understand and optimize implementations of power systems field devices that communicate on a common schematic definition and a scalable publish/subscribe architecture.

Numerous publish-subscribe communication protocols have been developed to enable IoT communications with multiple detailed surveys performed [5-6]. Many studies have compared these protocols in head to head trials with mixed results [7-11]. Available protocols tend to excel in divergent, specific situations, with no single protocol considered ideal for every situation [5]. Message Queue Telemetry Transport (MQTT) protocol offers a publish-subscribe method of communication supported by the TCP/IP protocol [12]. "Due to its light-weight, simple design, MQTT has become a popular protocol for IoT communications" [13].

Utilizing multiple platform trials, this paper will focus on detailed packet inspection to characterize delay patterns found within MQTT enabled, IoT communications. Delay in MQTT communications has been studied using a closed private network [14]; however, publishing frequency, packet size, and the role TCP plays in the characterization of delay was not considered. A correlation analysis of delay and packet loss to Quality of Service levels offered by MQTT has been measured with mixed results [9]. In a comparison of MQTT to CoAP, it was shown that MQTT's performance is dependent on differing network conditions when experiencing controlled packet loss [11]. Still, more detailed packet level analysis has not been performed on the variations of communication delay patterns which may indicate on issues with integrity, and security, as well as provide a generalized observation of the overall health of the connection.

This paper offers multiple novel contributions including: (i) a deep packet inspection of MQTT to better understand the behavioral relationship to TCP; (ii) integration of a real-time IoT server using Amazon Web Services to deliver brokered communications; (iii) multiple publisher-subscriber nodes operating from geographically separated locations to deliver real world network traffic conditions; and (iv) a methodology for establishing a baseline delay pattern in various environments that could prove useful for monitoring connection health.

The subsequent sections of this paper are organized as such: A more detailed examination of the MQTT protocol is presented and directly followed by a description of the experimental setup used to test and evaluate the protocol, including the communication architecture of the system. The final sections of this paper present the experimental procedures with results, conclusions, and future work.

Message Queue Telemetry Transport (MQTT) Overview

The following section briefly examines the basic connection and transmission methods of MQTT. All information regarding

specifics of the protocol were taken directly from the MQTT Standard, Version 3.1.1.

Why MQTT?

MQTT is described as a lightweight, open, simple publish/subscribe messaging transport protocol [13]. The protocol is ideal for use in situations where a code footprint is small and network bandwidth is scarce [12]. MQTT was chosen as the test protocol for this study due to its compatibility with the process of collecting data from large sensor networks, and organizing the data at one central location. This is an ideal scenario for the collection of sensor data at solar, wind, and other remote energy smart grid applications.

The protocol was originally designed to operate over TCP/IP providing "ordered and lossless bidirectional connections." [12]. The publish/subscribe method is enabled by a central server, often called a broker. The broker sorts incoming messages by topic and forwards the traffic to subscribers of the topic. MQTT provides three options for quality of service (QOS), as well as "small transport overhead" [12]. The MQTT protocol follows a basic flow of communication that only adds minor complexity with changes to QOS.

Connection Method

To begin an MQTT session one must first establish some connection parameters with a broker. Every control packet has a fixed header that gives instructions to the server to describe what the client wants to do, and a variable header (Fig. 1) to stablish the parameters that characterize the action type allowed by the protocol.

A basic MQTT session connection scenario would follow a two-packet request/response format. Once a connection is established a client can then either send a PUBLISH request, a SUBSCRIBE request, or any other control packet.

Publish Messages and QoS Levels

A PUBLISH request contains a payload with a topic/label attached to it. This topic is what the broker will use to transfer data to the appropriate destination on the subscription side. If a QOS level of zero is selected, then the publisher is in "fire and forget" mode as there will be no acknowledgement packet from the MQTT server [12]. If the QoS level is greater than 0, the acknowledgement scenario changes.

QOS level one requires that after the publish message is received and processed, the server will return a PUBACK message [12]. This is nothing more than an acknowledgement for the publisher to log that the message was processed [12]. If this acknowledgement is not verified, the packet will be resent with the DUP flag set to 1 to indicate that the packet may be a redelivery [12].

For a QOS of level 2, a PUBREC packet is sent back to the publisher as the second packet in the four-way handshake. with the PUBREL being the third packet sent back to the server by the publisher to let the server know the second packet was received [12]. The PUBCOMP is the fourth and final packet in the QOS 2 protocol exchange. The packet is sent by the server to the publisher to let the publisher know that the server is ready to receive the next packet of published data. Figure 2 generalizes the communication process for QOS levels one and two.

Asynchronous Publication QoS-1

A sliding window can be defined by the designer to permit a specified number of published packets through to the Server before the Client receives an acknowledgement. This type of asynchronous publication behavior results in the possibility of duplicate packets received after one of its successor messages. This can be avoided by narrowing the window to size one packet,

	Description	7	6	5	4	3	2	1	0
Protocol Name									
byte 1	Length MSB (0)	0	0	0	0	0	0	0	0
byte 2	Length LSB (4)	0	0	0	0	0	1	0	0
byte 3	"M"	0	1	0	0	1	1	0	1
byte 4	"Q"	0	1	0	1	0	0	0	1
byte 5	"T"	0	1	0	0	1	1	0	1
byte 6	"T"	0	1	0	1	0	1	0	0
Protocol Level									
	Description								
byte 7	Level(4)	0	0	0	0	0	0	0	0
Connect Flags									
byte 8	User Name Flag (1) Password Flag (1) Will Retain (0) Will QoS (01) Clean Session (1) Reserved (0)	1	1	0	0	1	1	1	0
Keep Alive									
byte 9	Keep Alive MSB (0)	0	0	0	0	0	0	0	0
byte 10	Keep Alive LSB (10)	0	0	0	0	1	0	1	0

Figure 1 CONNECT Packet, Variable Header Example

resulting in messages not being sent until its predecessor has been acknowledged [12].

Subscribe, Unsubscribe, and Other Processes

The subscribe side of the protocol operates in a slightly different manner than that of the publish side. A SUBSCRIBE request contains a fixed control type header with a variable header that contains desired topic name, wildcards, and QOS level. A SUBSCRIBE request is always followed by a SUBACK that specifies whether a subscription was successful as well as the granted QOS level [12].

Other control options include UNSUBSCRIBE which allows clients to unsubscribe from topics. This type of request requires the sever to send an unsubscribe acknowledgement (UNSUBACK), complete any QOS 1 or QOS 2 messages transactions, and then delete the client subscription. PINGREQ/PINGRESP are packets used to test the liveliness of a connection, and the DISCONNECT packet is an indication from the client to the server that the client is requesting a clean disconnection [12]. The newest version of the protocol, MQTT v3.1.1 allows for message transport across TLS and Websocket protocols as well as the TCP/IP protocol.

Experiment setup

The Texas Sustainable Energy Research Institute has multiple locations utilizing National Instruments products [15] designed for wired and wireless sensor networks. This experiment will leverage these pre-existing setups to test data collected from these locations. The distribution of collected data is handled by a modified version of a publishing virtual instrument (VI) currently operating in a remote location. An established LabVIEW MQTT

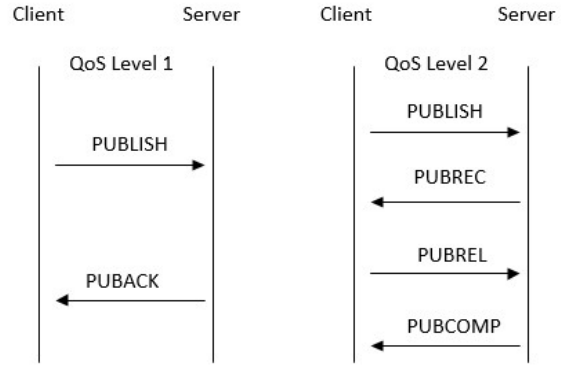

Figure 2 QoS Level 1 & 2 Flow Diagram

API [16] was used to implement the publisher side of the protocol, while a Linux based Mosquitto™ Broker [17] was deployed on an Amazon Web Server [18] to enable live network traffic conditions. Multiple subscribers were used to collect the data from the Mosquitto™ Broker. These subscribers reside on Windows 7 and Windows 10 based systems running a LabVIEW subscriber VI, while a third publisher/subscriber exists on a Linux based Raspberry Pi [19] utilizing the Eclipse PAHO project's, open source Python version of the MQTT protocol [20]. The network traffic was monitored at the server/broker, as well as the publisher, by means of a popular packet capture software package called Wireshark and Tshark [21, 22]. The collected data was then imported into R for statistical analysis [23].

LabVIEW Development Environment

LabVIEW is a virtual instrument (VI) engineering workbench with a visual programming language platform [24]. Publisher and Subscriber VIs were developed in LabVIEW for use on the Windows based systems. Implementation fragments are shown in Fig. 3. The Subscriber developed for this experiment is a simple state machine VI that contains the following states: CONNECT, SUBSCRIBE, UNSUBSCRIBE, DISCONNECT. The goal of this VI was simply to establish a connection with the broker to simulate typical use in a live environment.

The Publisher developed for this experiment is a modification of a Publisher currently deployed in the field at a TSERI/CPS solar energy site. The modifications include the ability to adjust the byte size of the data to be published, a modification to change the data publishing interval, the ability to add multiple topics, and the capability to reset the TCP connection; all while the publisher is in operation. Comparable to the Subscriber VI, this code is designed as a simple state machine with the following states: INITIALIZE, CONNECT, PUBLISH, DISCONNECT, EXIT. The INITIALIZE stage incorporates user selected session attributes in preparation for the connection. In the CONNECT state, a TCP connection is established, followed by the MQTT CONNECT request. If the CONNECT state is successful, the code moves into the PUBLISH state. The PUBLISH state publishes the data to the server under the selected topic. In this state the user can control the message size, and the interval in which the message is published. This allows the evaluation of delay based on the publishing interval during packet inspection. From this state the user may also reset the TCP connection. This is useful in establishing a new initial round trip time which is used in the delay calculation. This code continually attempts to establish a connection in the event of a disconnect due to network issues. In the event of a user initiated exit, the code will end the MQTT session with the broker and signal to terminate the TCP connection during the DISCONNECT state. The EXIT state exits the program.

LabVIEW MQTT API

LabVIEW Client API in native LabVIEW is a community project that implements MQTT v 3.1.1 standard protocol using the LabVIEW 2013 or later development environment [16]. A thorough examination of this API was performed including header and packet formation and disassembly on the publishing and subscribing sides. Compatibility tests were performed to ensure seamless performance with the server/broker. It should be noted that this API does not provide the method for establishing a sliding window for QoS > 0. This is instead left up to the designer/engineer as suggested by the MQTT standard.

Eclipse Mosquitto™ Broker

Eclipse Mosquitto™ is an open source (EPL/EDL licensed) message broker that implements the MQTT protocol versions 3.1 and 3.1.1 [17]. This API is installed on the Ubuntu 16.04 Server to act as the broker between publish and subscribe client requests.

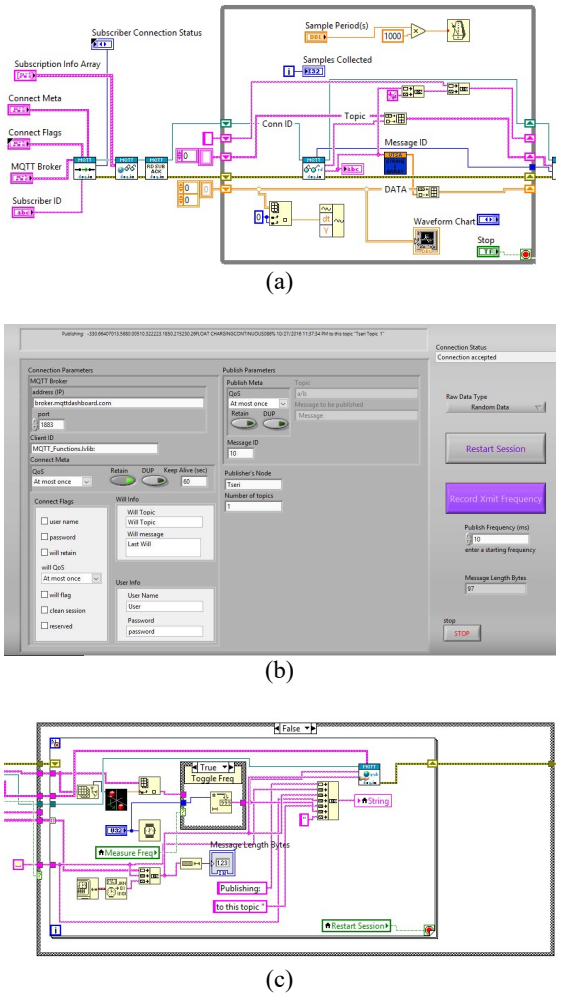

Figure 3 LabVIEW Code Sample of Subscriber State Machine (a), Publisher User Interface (b), Publisher Code Fragment (c)

The default settings were used in this study with the exception of adjustments made to the sliding window size in QoS-1 communication.

Ubuntu Server 16.04

Ubuntu Server is a Debian-based Linux operating system developed to run network servers [25]. This is an open source operating system, and it is offered as an option with Amazon Web Services compute package.

Packet Capture Software

Wireshark is an open source network protocol analyzer [21]. Wireshark provides live capture and offline analysis of the TCP/IP protocols as well as support for MQTT traffic. Wireshark was used on all Windows platforms for capture and analysis.

TShark offers a network protocol analyzer that allows the same capture ability of Wireshark, but for console based systems [22]. Tshark was used on the Ubuntu server to capture network traffic at the Mosquitto™ Broker. The created pcap files were then transferred to a Windows based system for analysis in Wireshark.

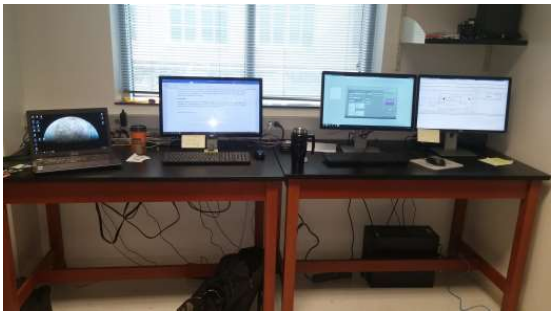

Figure 4 TSERI Labs Publish/Subscribe Nodes

Python MQTT

Python is open source software utilized by the Eclipse PAHO project to implement MQTT protocols [20]. In this experiment, Python code was used to publish and subscribe to topics from a Raspberry Pi.

Elastic Compute Cloud (EC2)

Amazon Web Services (AWS) offers a cloud based compute platform called Elastic Compute Cloud (EC2) [18] which is used as the server for this project. This compute instance was launched with Ubuntu Sever 16.04 and hosts the Eclipse Mosquitto™ broker used to implement the MQTT v3.1.1 protocol over a TCP connection.

Windows Platform Systems

Four Windows based systems were used to perform publishing and subscribing duties. A Windows 7 platform computer referred to as TSERI Labs is located on campus at UTSA, while two Windows 10 platform desktops, referred to as the pink box and the black box, are geographically separated from the TSERI Lab. A fourth Windows 10 laptop is used to access the Mosquitto™ Broker for coordinating TShark network traffic captures. All the Windows based systems contain the LabVIEW Publisher and Subscriber VIs designed for this experiment, as well as Wireshark for network traffic analysis.

Raspberry Pi

A Raspberry Pi is used as a publisher and subscriber [19], implementing the Python based MQTT API created by the Eclipse PAHO project [20].

Vaisala Weather Transmitter

Vaisala WXT520 Weather Transmitter offers six weather parameters including wind speed and direction, precipitation, atmospheric pressure, temperature and relative humidity [26]. The WXT520 outputs to serial data collected by the myRIO using RS-232 interface.

Pyranometer

Kipp and Zonen CMP11 Pyranometer is a "radiometer designed for measuring short-wave irradiance on a plane surface" [27]. This pyranometer's micro-Volt output is amplified to a 4-20mA signal. Using a translation table, the analog signal is converted to irradiance prior to publishing. The analog signal is captured and processed by the myRIO.

myRIO

National Instruments myRIO is an embedded hardware device with multiple, configurable, digital and analog IO [28]. This device gathers and formats all the sensor network data prior to publishing. Sample data gathered in the field by this device is used in the experiment.

System Architecture

The system architecture consists of a collection of hardware and software that perform two main functions, gathering data and processing data. A sensor network consisting of the above-mentioned hardware was placed at a solar array site to collect relevant data. A MQTT network was also built to process this data in a manner consistent with live operations. Fig. 5 and Fig. 6 generalize the arrangements of the MQTT network and sensor array respectively.

MQTT Network

The MQTT network was designed with flexibility in mind. The layout enabled multiple publisher and subscriber nodes at geographically separated locations to connect simultaneously to the AWS EC2 Mosquito Broker. The system operated in a live network traffic environment to achieve an accurate representation of real world conditions. Wireshark collected TCP and MQTT traffic at the client nodes, while TSHARK collected the same type of traffic at the broker.

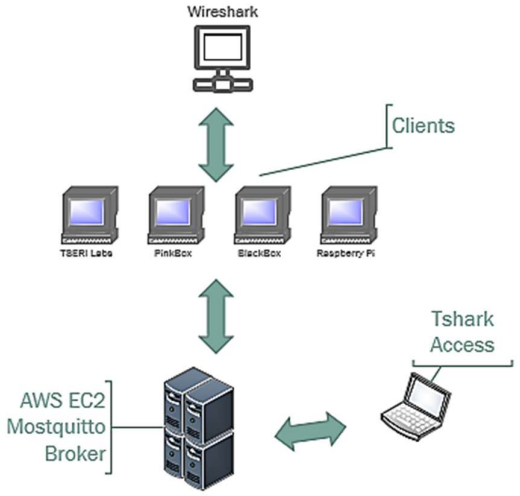

Figure 5 MQTT Network Architecture

Sensor Network

The sensor network gathered environmental information and reported the data to the myRIO. From that point, the publisher software residing on the myRIO organized and published the data to which clients could then subscribe. This data was captured and stored for use with this study. The VIs created to run on the Windows systems used in this study, accessed the data captured from the sensor network and published to the MQTT Network.

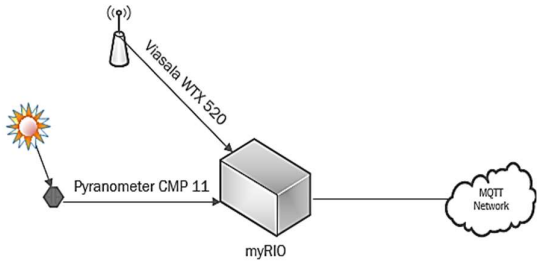

Figure 6 Sensor Network Architecture

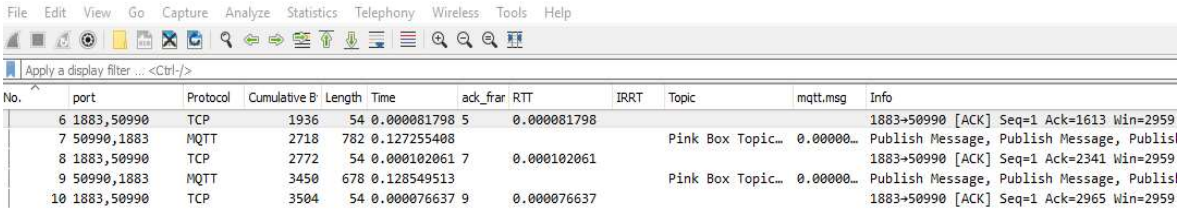

Figure 7 WireShark Display Example

Experiment

The following experimental procedures were informed by initial observation and packet inspection. The process of identifying and separating TCP behavior from MQTT behavior helped establish the conditions under which the delay trials were performed.

Packet Inspection

Over the course of this study, packet inspection remained an integral process to the development of delay analysis. From packet inspection, behavioral characteristics of TCP and MQTT can be separated and controlled, permitting a more accurate approach to understanding MQTT delay characteristics. The control process developed is detailed throughout the subsequent sections.

The process of packet inspection was performed in two steps.

i. Capture the packets with TShark and/or Wireshark
ii. Analyze the packets by means of Wireshark's user interface.

Capture filters were used to isolate the network traffic to the nodes of concern. An example of one such filter is as follows:

tshark -c 10000 -f"(host ip address) and port not 22"

This TShark code will capture 10k packets (-c 10000) of all traffic from a host defined by an ip address (-f"(host ip address) and port not 22"). Port 22 is ignored to eliminate the capture of SSH protocol packets. Inspection of the packets in Wireshark is performed with display filters, and column filters were used to configure exported .csv files used for statistical analysis. Figure 7 demonstrates results of a typical capture session.

Delay in General Terms

End-to-end delay in a network connection is made up of various events that can be expressed in typical situations by Equation 1. For any given node, there exists delay due to processing, queuing, transmission, and propagation [29]. Processing, queuing, and transmission delays occur at the node, while propagation delays occur between node. In this study, some of these variables are found to be controllable at times, while others are merely defined by a mean or median value through repeated experimentation.

$$D_{node} = D_{processing} + D_{queuing} + D_{transmission} + D_{propagation}$$

Equation 1 End to End Delay Equation

Transmission Delay, QoS-0 and QoS-1 with Asynchronous Publication

Special attention must be paid to transmission delay when aggregate data from multiple sensor networks is distributed through a single node. Transmission delay can be defined by Equation 2 [29]. Publishing intervals can influence transmission delay inversely, which has a consequential effect on queuing delays. When the TCP initiates its 3-way handshake between communicating nodes, an initial round trip time (IRTT) is

$$D_{transmission\ (sec)} = Segment\ Length_{(bits)}/Rate_{(bits/sec)}$$

Equation 2 Transmission Delay Equation

recorded by the packet inspection software. This interval represents an estimate for the latency of the connection between the TCPs. From the perspective of the node, $D_{propagation}$ can be approximated by IRTT/2. When MQTT data is published at intervals at least two times the IRTT, the publishing node's TCP will forward the single publish request at a rate roughly equivalent to the publishing interval minus the IRTT; however, if the publish interval is reduced below the IRTT threshold, the TCP will begin to queue published MQTT messages and push them as a single segment causing an increase in the packet length transmitted. Figure 8 demonstrates the TCP's behavior in relation to the IRTT threshold. An inverse variation begins to takes shape as the publish interval breaches the IRTT threshold. This relationship between the publishing interval and the segment length can be estimated by Equation 3, where k is an integer >= 1.

$$k^{-1}(IRTT) \propto k\ (\min\ [Segment\ Length])$$

Equation 3 Relationship Below IRTT Threshold

The increase in packet length is limited by an established Maximum Segment Size (MSS) regulated by the TCP. Once the MSS is reached, the TCP begins segmenting the packets into smaller portions to keep up with the transmission requirements. Increased Packet Length has a direct relationship with $D_{transmission}$, which in turn directly effects $D_{queuing}$ (Equation 4). It is important to understand and control this behavior to facilitate accurate MQTT delay calculations without TCP interference while using

$$D_{queuing} = D_{transmission} * (\text{Length of queue})$$

Equation 4 Queuing Delay

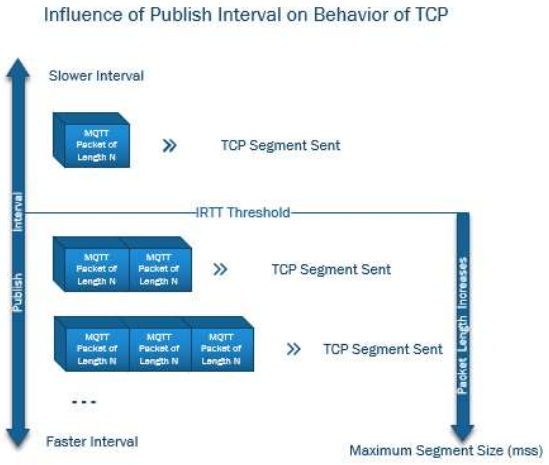

Figure 8 TCP Behavior at IRTT Threshold

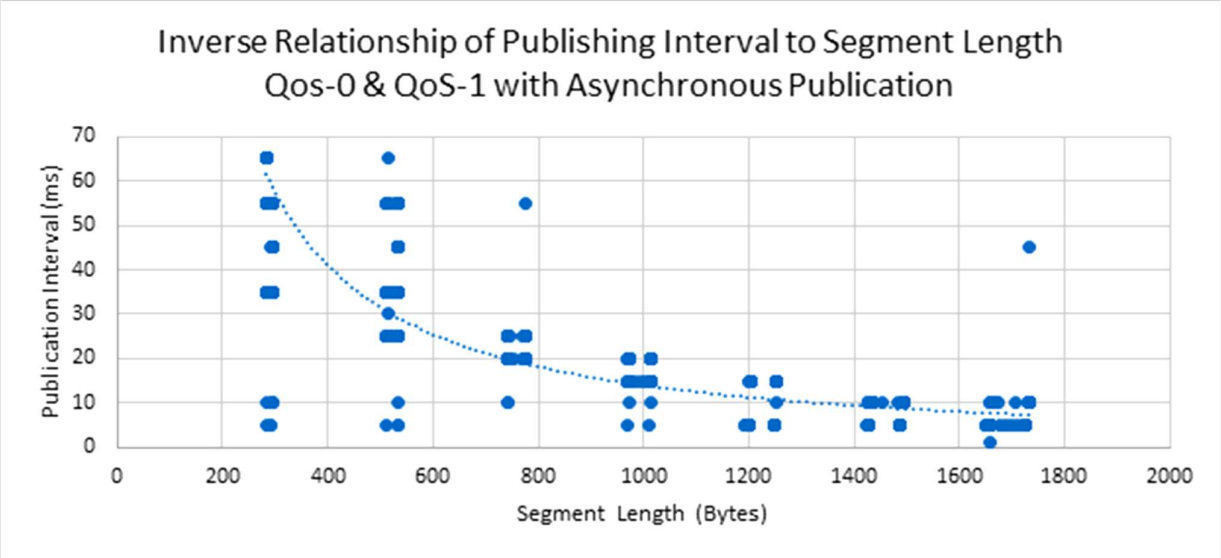

Figure 9 Publish Interval vs. TCP Segment Length

QoS-0 or Qos-1 with asynchronous publication. Figure 9 demonstrates this behavior as the published data was held constant at 238 Bytes, while the publishing interval was decreased from 65ms to 5ms. The IRTT was recorded at 62.5ms and the Maximum Segment Size (MSS) negotiated by the TCP was 1460 Bytes. It is shown that the segment length is broken up at 10ms due to the segment exceeding the MSS.

Measuring Delay

Visually, the measured delay involved in the process of sending and receiving packets at the Client and Server is shown in Fig. 10. The packet capture software is used will capture two packets per publish interval in normal network conditions. These packets include the published data and either the MQTT or TCP acknowledgement. In the special case of QoS-2 multiple packets will be sent in the exchange between the client and server as discussed in section 2.3. The calculated delay will be split up into two sections; delay at the node, and round trip time. From the perspective of the sending node, RTT includes the delay at the receiving node as well as the propagation delay to send data and receive the corresponding acknowledgement. Having described the publishing behavior for QoS-0 and Qos-1 with a sliding window, the remaining experiments include an examination of IRTT to set a baseline delay pattern, testing the effects of packet length on delay, and an analysis of delay in the presence of heavy network traffic.

Initial Round Trip Time (IRTT) Test

As previously stated, the initial round trip time is recorded when the Client establishes a connection with the Server by way of the TCP three-way handshake. Through deep packet inspection, the IRTT has potential to be a good indication of connection latency and may be used to establish a baseline for measuring delay shifts during an MQTT session. The IRTT test

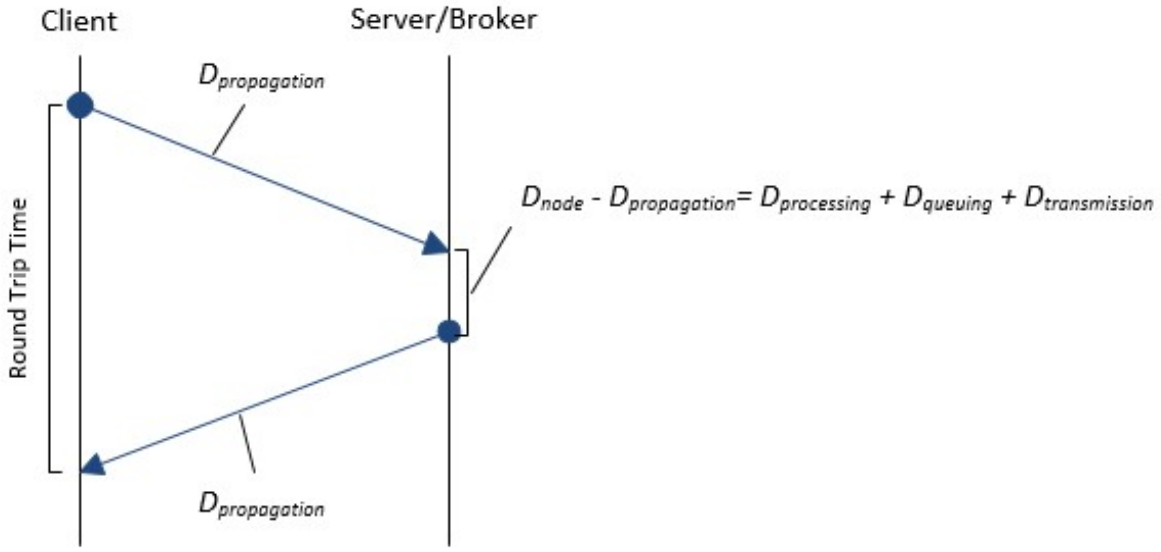

Figure 10 Components of Delay in RTT Timestamp

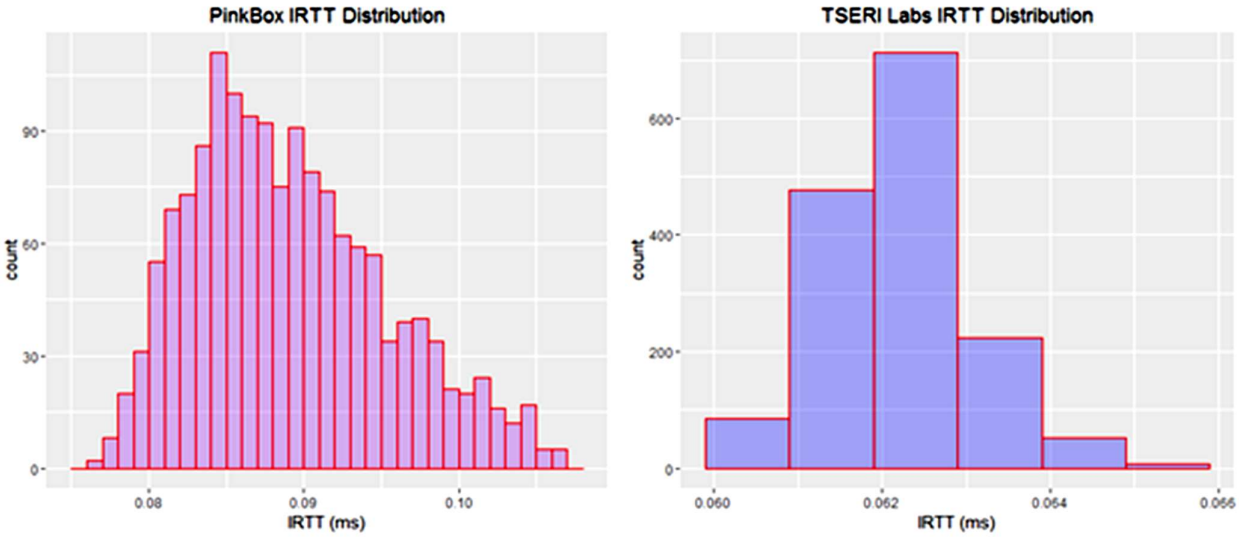

Figure 11 IRTT Distributions

performed for this study involved a TCP connection loop in which a connection was established and then terminated at two second intervals. Each connection and termination cycle required a 6-packet exchange. 10,000 packets were captured, yielding ≈ 1600 IRTT samples per test. The tests were performed at the TSERI labs and Pink Box locations to establish reference points for each node with the intention of illustrating that the propagation difference two geographically separated nodes might experience has a minimal effect on the distribution of the IRTT. Figure 11 establishes the IRTT distributions for both Clients. Outliers totaling less than 10% of the data sets were removed from both distributions using the five number summary method [30]. Though the propagation delay of IRTT is different in time at the nodes, the distribution over the median yields a similar curve from both locations; however, the spread of the data collected at the Pink Box location was slightly larger than that of the TSERI Labs location. A deviation calculation was performed on the data to justify the use of the TSERI Labs node over that of the Pink Box node for the remaining tests as a more stable connection could reduce error in delay interpretation and calculation. Median absolute deviation was used as it is considered a more robust method for handling outliers in a univariate data set [31]. Table 1 shows the median absolute deviation along with the median for each location.

TABLE I. Robust measure of variability in IRTT

Median and Deviation Table of IRTT		
Client	**Median IRTT (s)**	**Median Absolute Deviation**
Pink Box	0.08911979	0.007440069
TSERI Labs	0.06247810	0.000901661

Published Packet Length Trial

The packet length test involves changing the published packet size from 40 Bytes to roughly 9 KBytes. While the MQTT standard allows control packets of up to 256MB in length, the Ethernet standard breaks this down even further to a maximum transmission unit (MTU) at 1500 Octets [32]. The AWS Server allows Jumbo frames [33] with a MSS value of 8960 established by the Server. Packet inspection revealed that indeed the limits per publishing interval rest at roughly 8 Kbytes and breaching this limit leads to unpredictable behavior and data loss.

Determining a round trip time for QoS-0 proves difficult due to the lack of a MQTT acknowledgment from the Server. A TCP acknowledgment is sent from the Server however, and it provides a close approximation to the round-trip time when the publishing interval is greater than two times the IRTT. At publishing intervals near or below the IRTT threshold, these acknowledgments become less reliable. The Server, to maximize efficiency, begins to stack acknowledgements which increases the difficulty of accurately measuring delays. QoS-1 with asynchronous publication can yield similar results below the IRTT threshold, though reliable data can be gathered due to the MQTT acknowledgement requirement.

For this test, quality of service levels examined were published at an interval of 65ms. This interval was chosen for reasons previously discussed in sections 5.2.1 and 5.3.1. QoS-0 and QoS-1 with asynchronous publication display the stacking behavior detailed in section 5.2.1 due to contributory increases in both $D_{transmission}$ and $D_{queuing}$. In this state, instability increases producing variations in the delay of the system. This pattern continues until the TSERI Labs MSS is exceeded at which point the RTT stabilizes due to the delay in the system reaching a level of process equilibrium as seen in Figure 12.

QoS-1 with synchronous publication was held to the same publishing interval of 65ms, though publishing interval in this situation only sets the minimum as the next MQTT packet will not be published until the previous packet has been acknowledged. This eliminates the variable delay patterns caused by increases to $D_{queuing}$ since the queue never exceeds one MQTT packet.

In this circumstance, delay for QoS-1 with synchronous publication remains remarkably stable while the MQTT packet is kept below the MSS as seen by the RTT distribution in Fig. 13 (top). As the packet length exceeds the MSS, the distribution of delay shifts to a less predictable pattern shown in Fig. 13 (bottom). This is due in large part to the processing of the packets at the Server. While the MQTT packet length is held below the TSERI Lab's MSS, the TCP and MQTT acknowledgements are combined into a single ACK; however, packet inspection reveals a separation of the TCP and MQTT acknowledgements when the

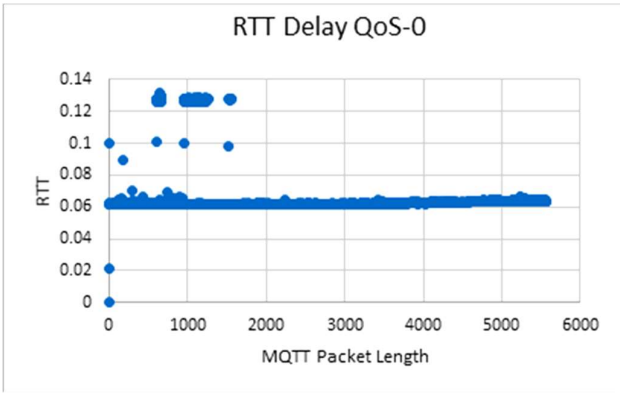

Figure 12 RTT(s) vs. MQTT Packet Length (bytes)

MQTT packet length exceeds the MSS. This is caused by the increased processing time required by the MQTT broker to examine and acknowledge the receipt of the larger MQTT packets. In this scenario, the TCP ACK is followed by the MQTT PUBACK reducing efficiency at the Server and increasing traffic in the channel.

The packet length test revealed a good candidate for establishing a predictable delay pattern that can be used to detect shifts in delay during live sessions. QoS-1 with synchronous publication (a sliding window set to size one) and a MQTT packet length set below the smallest established MSS results in a stable and controllable system that may allow for reactive monitoring of the delay.

Network Traffic Trial

Network traffic tests were performed using multiple nodes to generate traffic to the server. The test involved publishing a fixed payload to the Server from a single node, and then gradually adding traffic to the server. Packets were captured at the Client of interest, as well as the Server. This method allows a comparison of delay measured at the Client, to traffic (in Bytes) measured at the Server. Figure 14 (top) shows the distribution of the delay before the increased traffic, while Fig. 14 (bottom) displays the shifted delay distribution recorded as the network traffic was increased.

The results of the network traffic test are promising in that they display a perceived shift in the established delay pattern. Using a combination of MSS load balancing, IRTT, and expected deviations, one could conceivably create a reactive monitoring system used to report unexpected behavior based on an expected distribution.

Conclusion and future work

This paper presents an approach to establish an expected delay pattern for MQTT based communications which could be used to enable live reactive monitoring of an MQTT connection. Through deep packet inspection the relationships between data size, data collection intervals, network traffic, and its effect on delay were examined. It was shown that the initial round trip time (IRTT) established during the TCP three-way hand shake can give a reliable estimation for the propagation delay of the connection, which in turn can be used to establish a baseline for expected delay

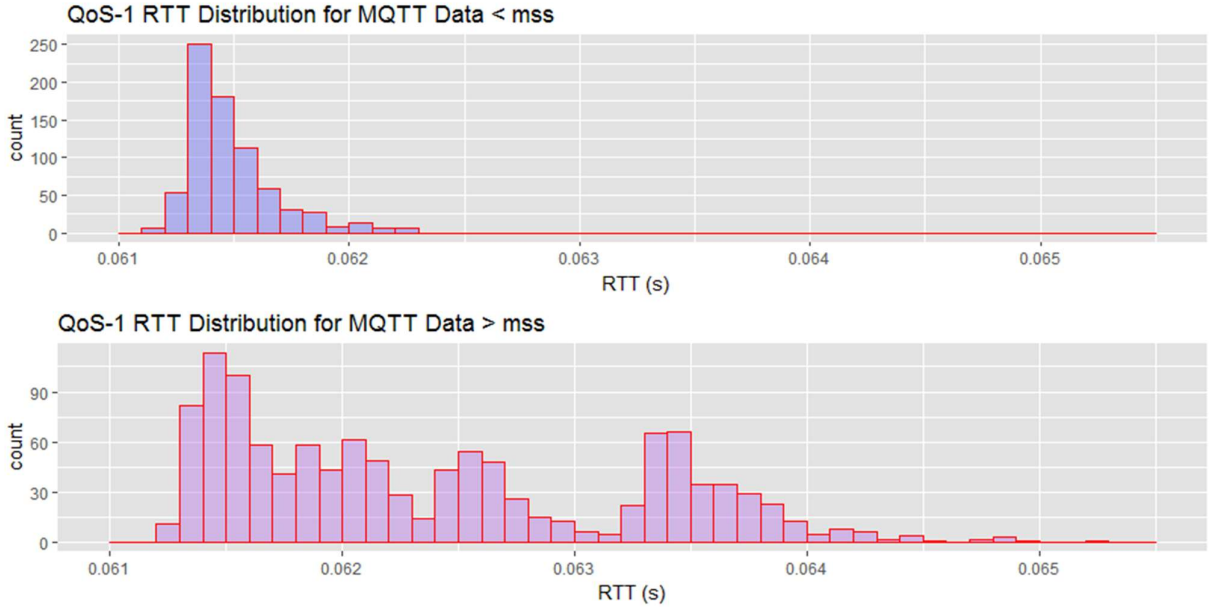

Figure 13 QoS-1 RTT for Synchronous Publication; Packet Length<MSS (top), Packet Length >MSS (bottom)

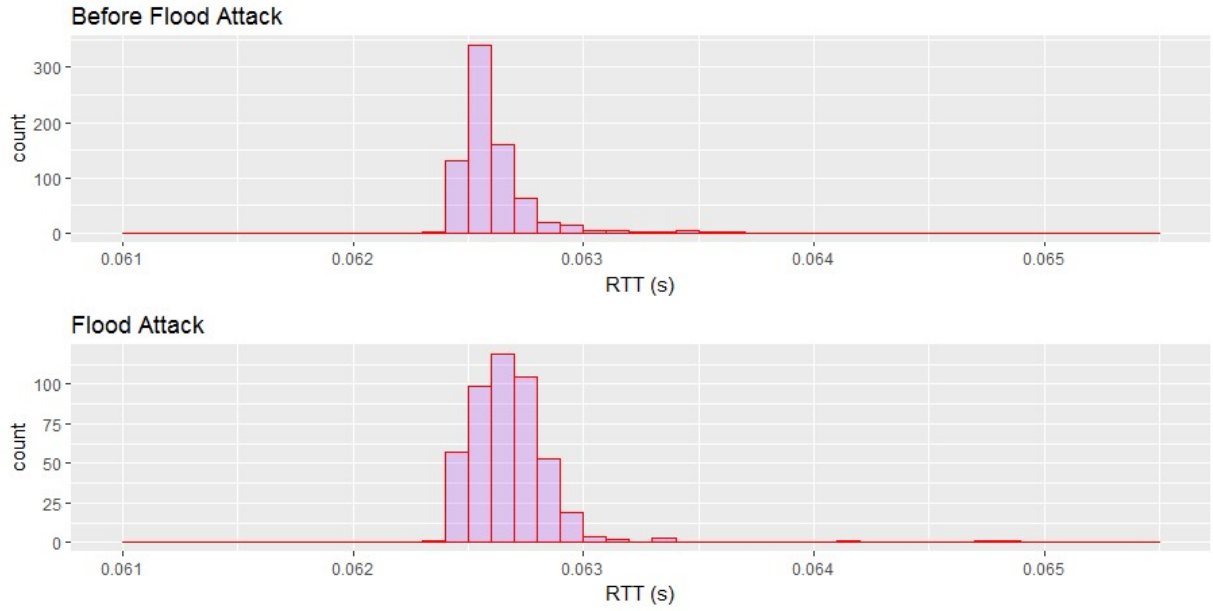

Figure 14 Measured Delay Shift Due to Network Traffic

in a connection. Through experimentation we demonstrated that shifts in the distribution of the expected delay are measurable using the combined application of MSS load balancing, IRTT, and expected deviations. These shifts may indicate session integrity, security, as well as generalize the overall health of the connection.

Use of this proposed method to improve sensor network designs in a publish/subscribe communication environment was a goal of this project. We hope that this research provides some insight into the development of Smart Grid IoT communications in situations where MQTT and sensor networks play an important role.

Further research into nodal behavior regarding the processing of MQTT packets and delay effects caused by transport protocol behavior at the node would be a future goal. This requires a more detailed investigation of individual MQTT packets at both the Client and the Server nodes. We believe a thorough study on recovering from packet loss, and the impact the recovery has on delay and throughput would provide a useful comparison of MQTT to other IoT protocols for use in large-scale senor networks. We would also like to further investigate modeling ideal, scalable, remote sensor network designs for the MQTT protocol to optimize performance and reduce data loss.

Acknowledgment

This project and the preparation of this paper were funded in part by monies provided by CPS Energy through an agreement with the University of Texas at San Antonio.

Support for this project was also provided in part by Texas Sustainable Energy Research Institute.

References

[1] Goel, S., Bush, S., Bakken, D, "IEEE Vision for Smart Grid Communications: 2030 and Beyond." 2013.

[2] Laval, S. and B. Godwin, "Distributed Intelligence Platform (DIP) Reference Architecture, in Vision Overview", S. Laval, Editor. 2015, Duke Energy.

[3] OpenFMB. 2016 [cited 2016 02 November]; Available from: http://www.sgip.org/openfmb/

[4] Idell Hamilton, T. "CPS Energy seeking partners to test "grid of the future" and bring more value to customers." 2014 [cited 2016 02 November]; Available from: http://newsroom.cpsenergy.com/grid-of-the-future/.

[5] Al-Fuqaha, A., Mohsen, G., Mohammadi, M., "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications." IEEE Communications Surveys & Tutorials, 2015. 17(4): p. 2347-2376.

[6] Al-Fuqaha, A., Mohsen, G., Mohammadi, M., "Toward better horizontal integration among IoT services." IEEE Communications Magazine, 2015. 53(9): p. 72-79.

[7] Beckmann, K. and O. Dedi. "sDDS: A portable data distribution service implementation for WSN and IoT platforms." in Intelligent Solutions in Embedded Systems (WISES), 2015 12th International Workshop on. 2015.

[8] Fernandes, J.L., Lopes, I.C., Rodrigues, J., Ullah, S., "Performance evaluation of RESTful web services and AMQP protocol." 2013 Fifth International Conference on Ubiquitous and Future Networks (ICUFN), 2013: p. 810-815.

[9] Lee, S., Kim, H., Hong, D., Ju, H., "Correlation analysis of MQTT loss and delay according to QoS level." The International Conference on Information Networking 2013 (ICOIN), 2013: p. 714-717.

[10] Luzuriaga, J.E., Perez, M., Boronat, P., Cano, J.C., Calafate, C., Manzoni, P., "A comparative evaluation of AMQP and MQTT protocols over unstable and mobile networks." 2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC), 2015: p. 931-936.

[11] Thangavel, D., Ma, X., Valera, A., Tan, H.X., Tan, C.K.Y., "Performance evaluation of MQTT and CoAP via a common middleware." Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014 IEEE Ninth International Conference on, 2014: p. 1-6.

[12] Banks, A. and R. Gupta. MQTT Version 3.1.1. 2014 29 October [cited 2016 02 November]; Available from: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html.

[13] OASIS Message Queuing Telemetry Transport (MQTT) TC. November 02, 2016 [cited 2016 02 November]; Available from: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=mqtt.

[14] Ahmed, T., D. Akopian, and R. Vega, "Measurement and Characterization of Communication Delays for Internet of Things." 2015, UTSA, Texas Sustainable Energy Research Institute(TSERI).

[15] National Instruments, A Global Leader in Test, Measurement, and Control Solutions. [cited 2016 04 September]; Available from: http://www.ni.com/en-us.html.

[16] Peter-daq.io. Community: MQTT Client API in native LabVIEW - National Instruments. 2016 Aug 9, 2016 [cited 2016 05 September]; Version 12:[Available from: https://decibel.ni.com/content/docs/DOC-32539.

[17] An Open Source MQTT v3.1 Broker. 2016 [cited 2016 02 November]; Available from: http://mosquitto.org/.

[18] Ami. Elastic Compute Cloud (EC2) Cloud Server & Hosting – AWS. [cited 2016 05 September]; Available from: https://aws.amazon.com/ec2/?nc2=h_m1.

[19] Raspberrypi. [cited 2016 04 September]; Available from: https://www.raspberrypi.org/.

[20] Paho - Open Source messaging for M2M. [cited 2016 05 September]; Available from: http://www.eclipse.org/paho/downloads.php.

[21] WIRESHARK Go Deep. [cited 2016 05 September]; Available from: https://www.wireshark.org/.

[22] tshark - Dump and analyze network traffic. [cited 2016 05 September]; Available from: https://www.wireshark.org/docs/man-pages/tshark.html.

[23] R Studio. [cited 2016 02 November]; Available from: https://www.rstudio.com/.

[24] National Instruments. 2016 31 October 2016 [cited 2016 2 November]; Available from: https://en.wikipedia.org/wiki/National_Instruments.

[25] adconrad. Ubuntu 16.04.1 LTS ReleaseNotes. 2016 [cited 2016 02 November]; Available from: https://wiki.ubuntu.com/XenialXerus/ReleaseNotes.

[26] Multi-Weather Sensor WXT530. [cited 2016 02 November]; Available from: http://www.vaisala.com/en/products/multiweathersensors/Pages/WXT530.aspx.

[27] CMP Series Pyranometer Instrunction Manual. 2006, KIPP & ZONEN.

[28] myRIO - National Instruments. [cited 2016 05 September]; Available from: http://www.ni.com/myrio/.

[29] Network Delays and Losses. [cited 2016 08 November]; Available from: http://www.d.umn.edu/~gshute/net/delays-losses.xhtml.

[30] Kerns, G.J., Introduction to Probability and Statistics Using R.

[31] Median absolute deviation. [cited 2016 13 November]; Available from: https://en.wikipedia.org/wiki/Median_absolute_deviation.

[32] Association, I.S., IEEE Standard for Ethernet, in IEEE Std 802.3™-2015 (Revision of IEEE Std 802.3-2012). 2015, The Institute of Electrical and Electronics Engineers, Inc.

[33] Jumbo frame. [cited 2016 18 November]; Available from: https://en.wikipedia.org/wiki/Jumbo_frame.

Author Biography

Brian Bendele received his BFA from Texas State University (2000) and spent time in Los Angeles working in post-production for film and TV. Since receiving his BS in Electrical Engineering from the University of Texas San Antonio (2014) and his MS in Electrical Engineering from the University of Texas San Antonio (2016) he has worked designing test equipment for data collection, with a focus on sensor network design, wireless, and RF communications.