Choice of Application Layer Protocols for Next Generation Video Surveillance Using Internet of Video Things

Article *in* IEEE Access · March 2019

DOI: 10.1109/ACCESS.2019.2907525

CITATIONS
2

READS
51

2 authors:

Tanin Sultana
University of Saskatchewan
11 PUBLICATIONS **19** CITATIONS
SEE PROFILE

Khabou Wahid
Institution of Agricultural Research and Higher Education - Tunisia
70 PUBLICATIONS **388** CITATIONS
SEE PROFILE

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2019.Doi Number

Choice of Application Layer Protocols for Next Generation Video Surveillance using Internet of Video Things

Tanin Sultana[1], Khan A. Wahid[2], Senior Member, IEEE

[1,2]Department of Electrical and Computer Engineering, University of Saskatchewan, Saskatoon S7N 5A9 SK, Canada

Corresponding author: Tanin Sultana (e-mail: tanin.sultana@usask.ca).

This work was supported in part by The Natural Sciences and Engineering Research Council of Canada (NSERC) and in part by The Canada First Research Excellence Fund (CFREF).

ABSTRACT Video surveillance has become ubiquitous due to the increasing security requirements in every sphere of life. The next generation video surveillance system (VSS) possesses great challenges in various applications, such as intelligent urban surveillance systems and smart cities. In these applications, we need to deal with the fast-growing number of surveillance nodes which introduce several constraints, e.g., high latency, high bandwidth, high energy consumption, and CPU and memory usage. To address these issues, the Internet of video things (IoVT), which is considered to be a part of the Internet of Things (IoT), can be a solution. The IoVT is composed of visual sensors (i.e., cameras) connected to the internet. Unlike conventional systems, the VSS under an IoVT framework provides multiple layers (i.e., edge, fog, cloud) of communication and decision making by capturing and analyzing rich contextual and behavioral information. Since an appropriate application layer protocol (ALP) can help in alleviating the challenges of future VSSs, the selection of ALPs is important for IoVT-based systems. Therefore, this paper presents a generic architecture of an IoVT-based VSS and a comparative analysis of several ALPs, such as MQTT, AMQP, HTTP, XMPP, CoAP, and DDS, with real-time experimentation. This analysis will assist the users to choose the appropriate ALPs in various surveillance applications and determine their suitability at different nodes of the IoVT framework.

INDEX TERMS Application layer protocols, Internet of Video Things (IoVT), Video Analytics, Video Surveillance.

I. INTRODUCTION

The use of *Video Surveillance Systems* (VSSs) has increased tremendously in the last decade, primarily to counter terrorism and to reduce the crime rate. It is also widely adopted in various cyber-physical applications including traffic analysis, healthcare, public safety, wildlife tracking, smart building, industrial automation, and environment/weather monitoring [1]. A typical VSS consists of image and video data. The number of surveillance cameras is also increasing around us. The advancement of technology has reached a level where mounting a camera for monitoring is cheap but finding sufficient human resources to sit, operate, and monitor the captured image is expensive. To deal with the increasing number of surveillance cameras, video surveillance that does not need human assistance for its functionality is becoming more popular. The computer-assisted reproduction of video analysis is called *video analytics* (VA) [2]. Its ability to detect or identify, monitor or locate, and analyze action or interaction in order to interpret the activity of a scene without human intervention is known as *intelligent video surveillance* (IVS) [8]. There have been over 6000 research papers published since 1971 in video systems design, tracking, modeling, behavior understanding, abnormality detection, real-time performance, and practical implementation of IVS and VA [9]. All of these works show the growing interest of researchers in video surveillance-based applications.

The Internet of things (IoT) is the inter-networking of physical devices, vehicles, buildings, and other items embedded with electronics, software, sensors, actuators, and network connectivity that enables these objects to collect and exchange data. The IoT melds together physical objects, virtual objects, living beings, user interfaces, and analytics that are all interconnected over an Internet-based infrastructure. IHS Markit forecasted that over 130 million surveillance cameras would be shipped globally in 2018 [3]. In another study [10], the authors estimated that the number of cameras would rise to 13 billion by 2030. Therefore, there is a need to combine visual sensors and surveillance data

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2907525, IEEE Access

within the framework of the IoT that will require higher bandwidth and power for transmission and computational ability [11]-[13]. The conventional cloud-based architecture alone is not capable of dealing with the issues that are related to the fast-growing number of visual IoT nodes in the IoVT paradigm which has led to the introduction of fog and edge computing. This IoT-integrated edge-fog-cloud framework has the potential to bring the services closer to the edge by a decentralizing cloud. Among many factors, application layer communication, which depends on selected communication protocols, is one that does influence the performance of such an integrated system [4].

The application layer provides services and ensures effective communication between application programs in a network [62]. The selection of an appropriate application layer protocol (ALP) is a prerequisite for a better understanding of a specific IoT application and its data-sharing demands [14]. The traditional web applications depend on HTTP, but unlike them, IoVT applications cannot depend on one protocol for all its requisites. However, hundreds of emerging protocols are available which can be chosen, depending upon the requirements of an IoT-based application. Consequently, the future of IoVT systems largely depends on the pros and cons of these emerging ALPs of the IoT to determine their best-fit scenarios [14]. Hence, this paper focuses on analyzing several widely used IoT protocols to fulfill the communication requirements of the IoVT. The contributions of this work are summarized as follows:

- We first discuss the edge-fog-cloud integrated generic architecture and functional requirements of the next generation IoVT-based VSSs.
- Then we present a comparative analysis of the characteristics of IoT ALPs and evaluate them in terms of performance parameters, such as latency, energy and bandwidth consumption, and network throughput based on real-time experimentation. This work provides information about their relative strengths and limitations which will be useful to system architects and protocol designers for determining the best fit and the potential of a protocol in an IoVT-integrated edge-fog-cloud-based VSS environment.

II. RELATED WORK

This section presents relevant research works that have proposed novel architectures, communication protocols and deployments of smart surveillance systems using IoT solutions. In [24], the authors describe a novel, real-time, wireless, multisensory, surveillance system with 3D-HEVC features and deployed a network-adaptive transmission protocol with adaptive packet frame grouping (APFG), adaptive quantization to maximize the quality-of-experience (QoE). The authors describe real-time, high update rate, super media data transfer over the internet of things [25]. Plageras *et al.* proposed an IoT-based surveillance system for ubiquitous healthcare monitoring [26]. In [27], the authors described an innovative topology paradigm offering better use of IoT technology in video surveillance systems. Memos *et al.* [28] proposed an efficient algorithm using Wireless Sensor Network (WSN) packet routing and high-efficiency video coding (HEVC) for a media-based surveillance system (EAMSuS) in an IoT network for a smart-city framework.

Traditionally, each surveillance camera/node directly sends the video data to the cloud in video surveillance applications. Cloud-based architectures are used for processing and storing essential data. Examples of the cloud-based IoT solutions are [5], [6], [7], [15], and [16]. Pflanzner *et al.* [45] conducted a detailed analysis of properties for IoT cloud providers. In [17], the authors present the contributions of both mobile cloud computing (MCC) and the IoT to the technology of big data. Plageras *et al.* [18] performed an analytic study of IoT technology, cloud computing, and large-scale data to resolve various issues facing the health sector. All these studies reveal the capability of cloud computing to satisfy many IoT requirements (e.g., monitoring, sensor stream processing, and visualization tasks). However, IoT-cloud architecture has issues regarding bandwidth and latency-sensitive video surveillance applications, which require surveillance nodes in the vicinity to meet their delay requirements.

On the other hand, fog-based solutions are considered suitable to address real-time processing, fast data response, and latency issues, thus extending the cloud computing and services closer to the edge of the network [19]-[22]. Fog, however, can be distinguished from the cloud by its proximity to the end users, the geographical distribution, and its mobility support [23]. Edge computing provides simple processing at the constrained IoT devices [61]. The opportunities, research challenges [54], detail architectural insights, features, and versatile roles in IoT applications show the promising future of fog computing [56]. Long *et al.* [57] propose an edge computing framework for cooperative video processing in multimedia IoT systems. Fog computing and efficient resource management methodology to balance the content generation rate of cameras in an IoVT environment are discussed by [58]. Skrbic *et al.* [61] present design requirements for a heterogeneous and collaborative edge-fog- cloud computing network.

Though a variety of protocols is available for the edge, fog, and cloud domains, there is no standard protocol to follow for a specific IoT system. Thus, one of the primary issues is the choice of the appropriate ALP/s for an IoVT system, while levering the advances in edge, fog and cloud computing. Though several works [65], [66] and surveys [7], [46]-[51] cover various aspects of the IoT architectures, along with its research, simulators, and testbeds [55], the specific issues of communication protocols in the application layer for any VSS are yet to be addressed by researchers. To fulfill the protocol requirements, the system designer should consider the devices (which range from the resource-constrained IoT edge-nodes to the resourceful clouds), data (produced at the IoT edge, fog, and cloud layers), and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2907525, IEEE Access

communication architecture of the system. An efficient VSS architecture should be able to achieve all these functionalities.

III. VSS REQUIREMENTS

In this section, we discuss the functional requirements of the next generation VSS. First, an intelligent VSS must perform the basic operation of surveillance, i.e., it should take images and videos from the visual sensor node. These visual data will help the system to detect and classify objects utilizing standard techniques such as frame difference, optical flow, and background subtraction [29]. The system can also track persons using video arrays [30], point tracking, kernel tracking, and silhouettes [29]. Automatic interpretation of human motion helps to identify abnormal behavior [31]. Behavior detection and event recognition are also crucial in this regard to detect intrusion [32] and movement disorder [33]. Learning and classification are other powerful ways for object detection and event recognition [34]. Bio-inspired adaptive hyperspectral imaging for real-time target tracking [35] and a brain-inspired neural-cognitive approach for thermal image analysis [36] are some examples. In addition, a VSS often needs precise real-time operation during crimes, such as breaking doors or locks of a highly secured area, or in critical situations, such as a stampede, or fire. The detection time of these types of incidents should be a few milliseconds.

All these tasks require the system to make quick decisions automatically by analyzing a large amount of visual data. Hence, local processing and decision making are essential in this regard. The act of fast decision making can speed up the process of initiating proper actions (e.g., turning the alarm on, shutting down a lift, sending SMS (Short Message Service) or calling an emergency number. Interconnectivity with other systems is another requirement for a VSS. To enable all these functional requirements, a VSS requires proper design architecture.

IV. GENERIC ARCHITECTURE OF AN IOVT-BASED VSS

In this section, we describe the generic architecture of an IoVT-based VSS with the aim of covering wide spectra of smart surveillance applications in various real-life domains, including healthcare, public safety in transportation, smart-home security, smart-city surveillance, environmental disaster, and weather monitoring. We specify the logical design considerations of data structures, nodes, and communication architectures. Traditional cloud-based systems consist of only clouds and the device nodes, but the combined edge-fog-cloud-based VSS introduces new abstractions between them [4]. Fig. 1 illustrates a typical edge-fog-cloud-integrated IoVT based VSS. This framework describes the possible devices and the communication architecture from edge to fog and fog to cloud using the ALPs. The edge-node devices are usually positioned to send visual surveillance data to a more capable computing system in the fog layer in order to offload larger computational tasks that require low-latency. Cloud computing provides storage resources and performs the largest amount of computation and analysis.

A. DATA STRUCTURE

The authors of [24] classify the different visual and sensory media data of a smart surveillance system into three categories: image and video, audio, and multisensory data. Unlike mainstream IoT systems (where the nature of the data is discontinuous and small in size), an IoVT-based VSS deals with a large volume of data [11]. The streaming of such data also takes a long time because of the transmission of many packets [12]. Consequently, processing and transmission of IoVT-based VSS data bring challenges for devices and applications, causing high latency and consuming high bandwidth and energy [11].

B. EDGE-NODE ARCHITECTURE

IoT nodes are smart devices equipped with sensors or actuators that provide information on real-world observations [37]. The regular IoT nodes have limited resources in computing, communication capacity, and energy. The image sensors and cameras are the key components for acquiring the real-world image and video data in the IoVT. The design challenges of these visual sensor-equipped IoVT edge nodes are high-energy consumption and high-transmission bandwidth requirement, as well as a high-power consumption for data transmission and communication [13]. Though video and image compression techniques can significantly reduce the required bandwidth, it is still high when compared to the lightweight requirement of common IoT nodes [13], [37] (e.g., 10 Mbps for 1080p over H264 vs. 1 byte of a temperature sensor reading). Hence, an IoVT edge node may require relatively higher energy resources, computational power, and communication capability. Kokkonis *et al*. explain quality of service (QoS) requirements for a super media application in [25], and state that each packet frame of video data should be equal to the maximum transmission unit (MTU) size of the physical layer, which requires a high data rate of about 2500-40000 kbps. The required throughput of a raw 1920×1080@30fps video can reach up to 1.49 Gbps. Therefore, the key criteria of an IoVT edge node are energy-efficient processing, transmission with high throughput, and relatively higher computational ability [11], [12]. Fig. 1 shows camera-connected possible edge-node platforms of an IoVT, including Arduino, Raspberry Pi, and BeagleBone.

C. FOG-CLOUD ARCHITECTURE

There will be many visual nodes in an IoVT system. Hence, local edge-node computation alone will not be able to solve all the potential issues. For instance, multi-camera object tracking may share data beyond single cameras [10]. Therefore, offloading some of the image-video compressions and coding to a fog-computing unit can be another attractive approach to include in an IoVT-based VSS platform [11]. Fog paradigm can also be helpful in designing a vendor-independent system (e.g., ONVIF [Open Network Video Interface Forum] created such a standard for IP products [44]). A visual fog paradigm can also add more efficiency, lower bandwidth, node-end power consumption, and latency [1]. Therefore, in an edge-fog-cloud platform (as shown in Fig. 1), intra- prediction compression and light computations

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2907525, IEEE Access

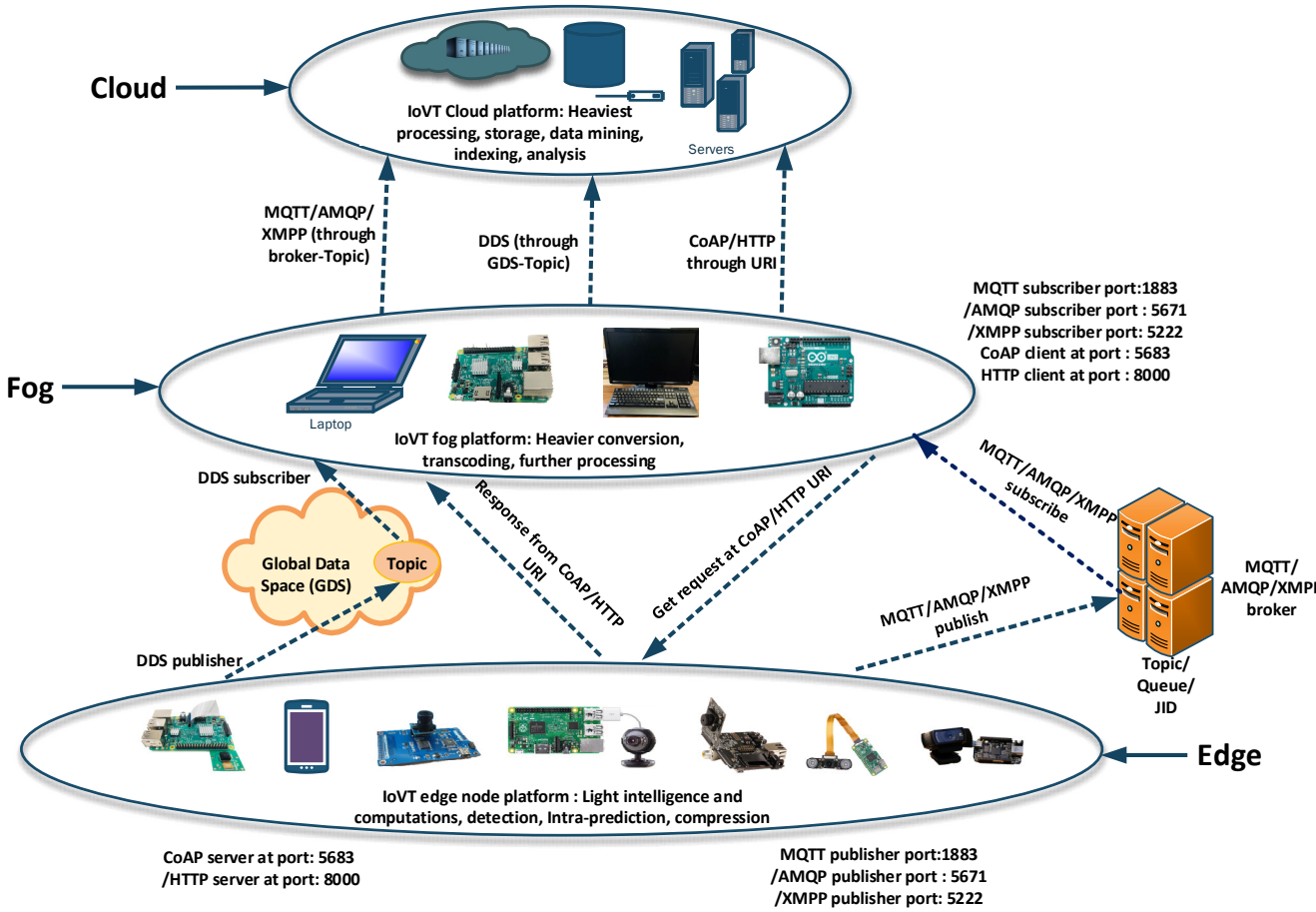

FIGURE 1. Generic architecture of IoVT-based VSS

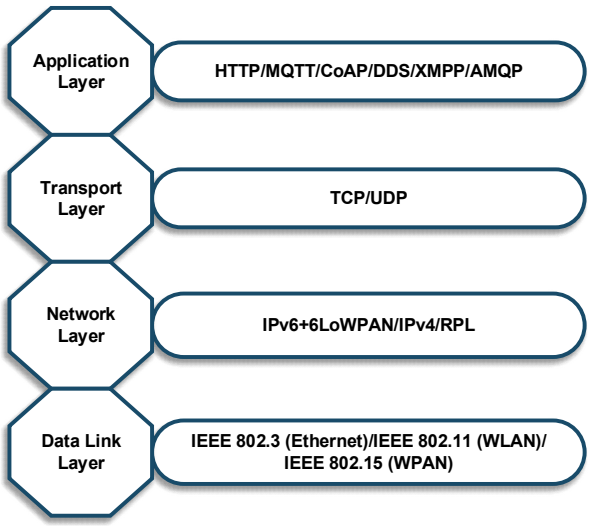

FIGURE 2. The general IoVT protocol stack

can be done at the edge-node side, whereas fog-computing units can accomplish the heavier conversions, transcoding, and further processing, and finally, the cloud unit can handle the heaviest processing [10], [11]. The authors in [1] propose an efficient framework for visual fog-computing, describing key features such as reusability, efficiency, and configurability. Examples of possible fog-node platforms can be personal computers (PCs), laptops, and Raspberry Pis. Examples of cloud services are the servers of Apple iCLOUD, Amazon Web Services (AWS), IBM cloud, Microsoft Azure, VMware, Google, and Digital Ocean.

D. COMMUNICATION ARCHITECTURE

A reliable communication protocol is indispensable for an IoVT-based VSS [12]. Fig. 2 describes the IoVT protocol stack [14], [38]. It consists of the Data link (physical layer), Network, Transport, and Application layers. The physical layer is the first layer of the IoVT protocol stack where physical devices, like IoVT visual sensors or cameras, will work. The IEEE provides widely used physical layer standards, such as IEEE 802.3 (Ethernet), IEEE 802.11 (Wireless Local Area Network/WLAN), and IEEE 802.15.4 (Wireless Personal Area Network/WPAN). A popular communication protocol in this layer for standard IoT-based systems is IEEE 802.15.4 [38]. It is designed for low power, lower data transmission, less complex modulation, lower frame overhead, and a low data rate (max. 250 kb/s) [52],[53]. 6LoWPAN is a protocol defined to enable Internet Protocol version (IPv) 6 over the IEEE 802.15.4 standard [52]. The maximum size of the media access control (MAC) layer (MAC is a sublayer of the data link layer) frame in IEEE 802.15.4 is 127 bytes which leaves only 102 bytes for an IPv6 packet, even though IPv6 requires an MTU size of 1280 bytes for operation. Therefore, 6LoWPAN provides the necessary adaption between IPv6 and IEEE 802.15.4

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2907525, IEEE Access

[52]. Again, the IPv6 Routing Protocol for Low-power and Lossy Networks (RPL) is another network layer convention which provides the facility for devices to perform point-to-point or multipoint communication [38].

IoVT systems require strictly sustained information delivery. Any lost packet in the video coding leads to severe error propagation between two consecutive frames [12]. The packet loss for video transmission should be ≤ 1% as Kokkonis *et al.* describe in [25]. Because of the large MTU size, high data rate, and almost no packet loss requirements of the IoVT node, 6LoWPAN and RPL over IEEE 802.15.4 for large visual data transmission cannot be used. On the other hand, IEEE 802.3 and IEEE 802.11 permit large MTU sizes of 1518 bytes and 2304 bytes, respectively. Hence, IPv6 or IPv4 over 802.11 or 802.3 will be an efficient choice for IoVT-based VSS applications. The choice of transport layer protocols (i.e., UDP [User Datagram Protocol] or TCP [Transmission Control Protocol]) depends on the ALPs like CoAP, MQTT, AMQP, and HTTP [38]. Also, the choice of proper ALPs for an IoVT-based VSS application depends on their best suitability at different node levels (edge, fog, and cloud, as shown in Fig. 1), which is to be evaluated based on their performances, as described in the following sections.

V. APPLICATION LAYER PROTOCOLS

Based on the application developers' perspective, the design alternatives for developing IoT-based real-time applications are communication protocols, message encoding format, and the Web platform [37]. Hence, this section presents some widely accepted and emerging communication protocols for IoVT-based smart surveillance systems: MQTT, HTTP, CoAP, AMQP, DDS, and XMPP. Table 1 shows a comparison based on the general criteria of these protocols.

A. MQTT (MESSAGE QUEUING TELEMETRY TRANSPORT)

MQTT, designed in 1999, is a lightweight machine to machine (M2M) communication protocol that supports the publish/subscribe architecture with minimal bandwidth requirements, power consumption, and message data overhead [14]. An MQTT client publishes messages to a broker through an address known as Topic. Another client can receive the messages by subscribing to that Topic. Clients can subscribe to multiple topics [38]. MQTT provides three levels of quality of services (QoS) to ensure reliability. QoS 0 mode sends a packet one time only without requiring confirmation messages or ACK (Acknowledgement). QoS 1 mode delivers the messages at least once by requiring an ACK. QoS 2 mode guarantees that the message is delivered exactly once [38]. MQTT has a facility for a variable length header. It does not provide any MQTT ACK response, but its default transport protocol (TCP) provides TCP ACK for each packet sent [14], [39].

B. HTTP (HYPER TEXT TRANSFER PROTOCOL)

HTTP is a globally accepted web messaging standard which offers several features such as persistent connections, request pipelining, and chunked transfer encoding. HTTP supports request/response RESTful Web architecture. Unlike MQTT, HTTP uses the Universal Resource Identifier (URI) instead of topics. It is a text-based protocol that does not define the size of header and message payloads. It depends instead on the web server or the programming technology [14].

C. COAP (CONSTRAINED APPLICATION PROTOCOL)

CoAP which is another lightweight M2M protocol was standardized in 2014. It provides a request-reply interaction model like REST (Representational State Transfer) to constrained devices and environments. CoAP was developed to interoperate with HTTP and the RESTful Web through simple proxies. It supports both request/response and resource/observe architectures. Like HTTP, CoAP uses URI. The server sends data through the URI, and the client receives data from a specific resource indicated by that URI [14]. Its payload size should be small, and the maximum size of the payload depends on the web server or the programming technology. CoAP uses confirmable (CON) and non-confirmable (NON) messages to provide two different levels of QoS. The receiver acknowledges CON messages with an ACK packet, unlike non-confirmable messages that do not need ACK [14], [38], [39].

D. AMQP (ADVANCED MESSAGE QUEUING PROTOCOL)

AMQP, developed in 2003, is a lightweight M2M protocol designed for reliability, security, provisioning, and interoperability. This protocol supports both request/response and publish/subscribe architectures. In AMQP (version 0.9.1) clients publish messages to a broker. The broker stores the messages in queues as long as the subscribers to those queues have not received the messages. Depending on the exchange type, there are four possible ways of routing messages between publishers and consumers: directly, in fanout form, by topic, or based on headers [14], [37].

E. DDS (DATA DISTRIBUTION SERVICE)

DDS uses a data-centric publish-subscribe (DCPS) model for real-time M2M communications and was developed by Object Management Group (OMG) in 2004. Applications use Data-Writer of the given data type to publish data objects to a topic over Publishers component. Data-Reader of the given data type receives data objects over the Subscriber component. The publishers discover the subscribers dynamically, using matching topics and data types [37]. In contrast to other publish-subscribe protocols like MQTT or AMQP, DDS relies upon broker less architecture and uses multicasting. It supports 23 QoS policies providing various communication criteria like reliability, priority, urgency, and security [40]. DDS specifies the use of multicast UDP within LAN (Local Area Network) and TCP transport for communication over WAN (Wide Area Network) [37]. The two chief stakeholders in the DDS market proposed the two main implementations of DDS. One is OpenSplice by PrismTech, and the other is Connext DDS by Real-Time Innovation (RTI) [42].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2907525, IEEE Access

TABLE 1. COMPARISION AMONG APPLICATION LAYER PROTOCOLS

Parameter	MQTT[a]	HTTP[b]	CoAP[c]	AMQP[d]	DDS[e]	XMPP[f]
Header size	2 Byte	No limit/ Undefined	4 Byte	8 Byte	Typically, large overhead, varies with situation and configuration	Very large with no limit (Varies with data size)
Maximum payload per packet	256 MB	Defined by web server/ programming technology	64 KB (for block-wise UDP transfer) 1280 Bytes (for IPv6 to avoid IP fragmentation) 60-80 Bytes (to avoid adaptation layer fragmentation for 6LoWPAN)	Negotiable and undefined (depends on the broker/ server and programming technology)	Large data transmission is supported, it can be limited by transport	10000 Bytes (Max. stanza size depends on the server configuration)
Encoding format	Binary/Text	Binary/Text/file	Binary/Text	Binary/Text	Efficient Binary	Text (XML) based and EXI (Efficient binary)
Transport protocol	TCP	TCP	UDP	TCP	UDP, TCP	TCP
Quality of service (QoS)	3 QoS: QoS 0 – at most once QoS 1 – at least once QoS 2 – at exactly once	Limited	CON – must be acknowledged NON – acknowledgement is not required	Settle – at most once Unsettle – at least once	23 QoS	Till no QoS is supported
Security	TLS/SSL	TLS/SSL	DTLS	TLS/SSL	TLS/SSL/DTLS	TLS/SSL
Resource locator	Topic name	URI	URI	Exchange-queue	Topic	JID (Jabber Identifier)
Default port no.	1883	80/8000	5683	5672	-	5222
RESTful methods	Absent	present	Present	Absent	Absent	Absent

a. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html
b. https://tools.ietf.org/html/rfc2616
c. https://tools.ietf.org/html/rfc7252
d. https://www.rabbitmq.com/resources/specs/amqp0-9-1.pdf
e. https://www.omg.org/spec/DDS/1.4/
f. https://tools.ietf.org/html/rfc6120

F. XMPP (EXTENSIBLE MESSAGING AND PRESENCE PROTOCOL)

XMPP is an Instant Messaging (IM) standard used for real-time messaging with XML (eXtensible Markup Language) streaming technology at its core. It connects a client to an XMPP server using a stream of XML messages called stanzas, and uses TCP as the default transport protocol and TLS/SSL for security. XMPP serves various purposes including publish/subscribe messaging, multi-user chatting, and sensor data exchange [37], [40].

VI. EXPERIMENT SETUP

According to the generic architecture described in section IV, test video surveillance applications were set up to transmit real-time images and video through the IoVT network. In this case, an IoVT source node was used to capture images and video and send them to the IoVT sink nodes, utilizing the described protocols in section V. The source node will acts either as an edge or a fog, while the sink node serves either as the fog or the cloud, depending on the role of the source node. Hence, this section describes the detailed technical specifications of these nodes along with the specifications of the software packages. Fig. 3 shows the experimental setup.

[1,2]www.raspberrypi.org

A. SOURCE NODE

Raspberry Pi[1] 3 model B (RPi-B) is one of the most common devices for emulating an IoT source node [11]. Its quad-core 64-bit ARM Cortex A53 operating Raspbian stretch (clocked at 1.2 GHz) has a built-in 802.11 n Wireless LAN and 400 MHz Video Core IV. These features enable RPi-B to encode/decode visual data in H264 format and implement ALPs in real time. The Pi camera[2] board plugs directly into a dedicated 15 pin MIPI (Mobile Industry Processor Interface) Camera Serial Interface (CSI) on the RPi-B through a 15-pin Ribbon cable. This 5 MP (Mega Pixel) camera sensor is capable of capturing 2592×1944 resolution static images, as well as supporting the 1080@30fps, 720@60 fps, and 640×480@60/90 fps video recordings. Hence, the Pi camera connected with RPi-B will serve as a visual sensor node for this experiment.

B. SINK NODE

The Intel(R) Core (TM) i7 processor, clocked at 3.40GHz, with a 24GB RAM and 64-bit Windows 7 operating system works as a data sink node for this experiment. Another Raspberry pi 3 model B board is also used as a sink node.

C. SOFTWARE AND COMMUNICATION PROTOCOL

IEEE 802.11(WLAN) was set as the physical layer and IPv4 as the network layer protocols. The server/client side and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2907525, IEEE Access

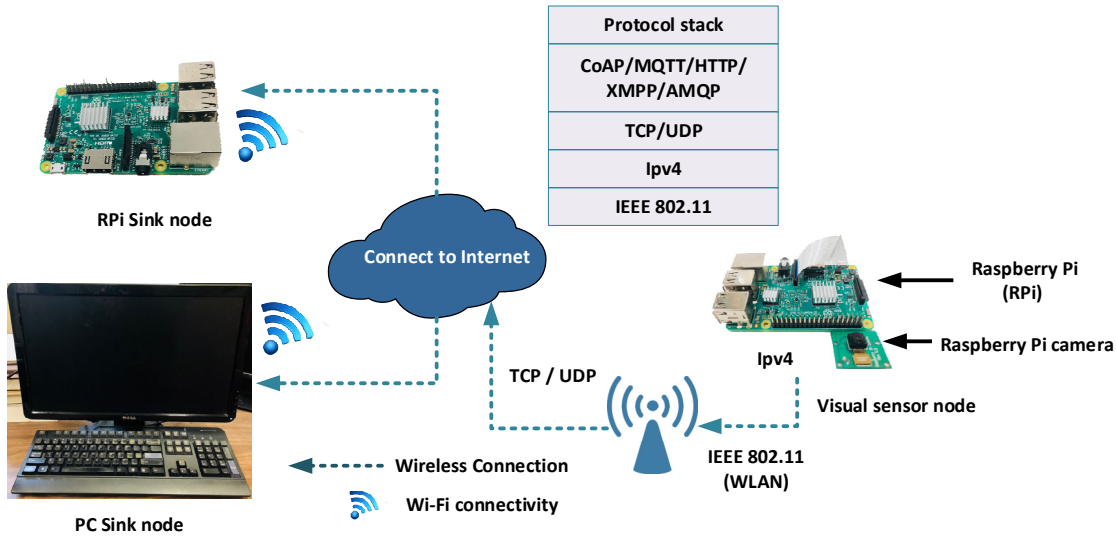

FIGURE 3. Experimental Setup

publisher/subscriber side scripts for test applications are all written using Python[3] version 2.7.13. Wireshark8[4] (version 2.6) and tcpdump[5] were used to monitor and analyze the generated network traffic between the source and the sink. The test parameters, like average latency (including data processing, decoding and transferring time), throughput, memory and CPU usage, overhead, and energy and bandwidth consumption were calculated with the help of these software programs.

VII. TEST PARAMETER MEASUREMENT

This section explains the details about the methods and experimental procedures taken to measure all the test parameters of interest.

For MQTT and AMQP protocols, the source node publishes visual data to the broker/server, and then the broker forwards data to the client running at the sink node. An Eclipse mosquitto[6] message broker has been used as the MQTT broker, while AMQP used a CloudAMQP[7]-provided RabbitMQ[8] server. XMPP used a Jabber[9] server for communication between two clients. The HTTP and CoAP servers were set at the source node (RPi-B), and the clients accessed them by sending GET requests and using specific URI and port numbers.

Unlike MQTT and AMQP, DDS does not require a broker. It uses Global Data Space (GDS) instead. The Publishers forward their data into the common GDS, and the DDS support propagates the data to all interested Subscribers having common topics. The underlying DDS data distribution is decentralized and adopts a peer-to-peer model, and therefore requires no centralized broker [42]. RTI uses the standard RTPS (Real-time Publish-Subscribe) data delivery protocol by default. OpenSplice DCPS (Data Centric Publish Subscribe) implementation supports an optimized proprietary data delivery protocol called RTNetworking. It also supports RTPS for interoperability with other DDS solutions [42]. Connext DDS by RTI was used for the experiment. The methods used to calculate the test parameters are described in the following sub-sections.

A. LATENCY

Network latency[10] is defined as the amount of time it takes for a packet to reach a destination device from its source device. Latency can be measured in several ways, including one way and round trip. Synchronizing the clocks at both ends and then subtracting the packet transmission time from the packet arrival time at its destination gives the latency in one-way measurement. In the case of the round-trip method, latency from the sender to the receiver and back is measured, and it is assumed that the end-to-end latency is half of this result. This method measures latency from the same device, which removes the necessity of the clock synchronization. In this experiment, the one-way method for measuring the latency for both full image/video and per packet data is used, taking the average of multiple sessions for each.

B. THROUGHPUT

Throughput is the quantity of data being sent or received per unit time. The maximum possible throughput measures the bandwidth of the communication link. The throughput was calculated using only the payload-containing packets per second (packets/s). The maximum transmission unit (MTU) defines the maximum Ethernet frame size (i.e.,1514 bytes), which cannot be exceeded in this network configuration.

C. CPU AND MEMORY USAGE

The percentage CPU usage denotes how much of the processor's capacity is in use, and can be calculated by dividing the CPU usage time by the total process running time. The ratio of the process's resident size to the physical memory on the machine is the percentage memory usage. These parameters are calculated using the Linux command "top" in the Raspberry Pi command terminal.

[3]https://www.python.org
[4]https://www.wireshark.org/
[5]http://www.tcpdump.org/
[6]https://mosquitto.org/
[7]https://www.cloudamqp.com
[8]https://www.rabbitmq.com/
[9]https://xmpp.org/software/servers.html
[10]http://smutz.us/techtips/NetworkLatency.html

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2907525, IEEE Access

D. PACKET LOSS

Packet loss is the number of packets lost per 100 packets sent from a source. Analyzing the captured packets stored in the pcap[11] files with the help of Wireshark, we assessed the number of lost packets for each transmission.

E. AVERAGE BANDWIDTH (BW) CONSUMPTION

Using the Wireshark packet analysis tool we measured the average BW consumption for each protocol. The captured packets were saved in a pcap file by the "tcpdump" command in the RPi-B node. Wireshark analyzed the pcap file to calculate the BW.

F. OVERHEAD

The network overhead generated for transmitting the data packet was calculated utilizing the pcap files.

G. ENERGY CONSUMPTION

We calculated the energy consumption by measuring the total Power (Voltage × Current) drawn by each protocol process during data transmission. First, the transmission latency was measured, then the latency was multiplied by the Power to find out the consumed energy for each protocol.

VIII. RESULT AND DISCUSSION

First, we analyzed the performance of different protocols using color images of different resolutions and sizes. Then, we checked the performance regarding video sequence transmission. Finally, we performed multiple IoVT node-based analysis. The analyses can be categorized into three parts: analysis on single-node image data, analysis on single-node video data, and analysis on multi-node data. Table 2 shows the details of the visual data samples.

TABLE 2: VISUAL DATA USED IN THE EXPERIMENT

	Resolution	Payload size (kB)	Format
Image	320×240	76.6	JPG
	640×480	196.8	JPG
	960×720	418.1	JPG
	1280×960	699.9	JPG
Video (23.979 fps)	1280×720	871	H264

A. ANALYSIS ON SINGLE-NODE IMAGE DATA

The analysis performed on single node image data of different resolutions to measure the test parameters (described in section VII) on the protocols is presented in this section.

1) LATENCY ANALYSIS

1.1) MQTT

The first experiment was performed to transmit color images of various resolutions from the RPi-B source node to the RPi-B sink node using the MQTT protocol. We used an MQTT Python client library called Eclipse Paho[12] to create applications for the image publisher. It connects to the mosquitto broker to route the visual data towards the Subscriber, using a topic. The MQTT protocol is an M2M connectivity protocol. Hence, the source node encodes the image data into binary arrays before transmission. The receiving node decodes these byte arrays and saves the image into JPG format. We have calculated the encoding, transmitting, and decoding time of the payload. Total latency (T) is calculated using equation (1). We repeated each test five times to calculate the average latency. Fig. 4 shows the outcomes. It is evident that Latency (T) increased proportionally with the quality of the image.

$$Latency\ (T) = Encode\ time\ (T_e) + Transmit\ time\ (T_t) + Decode\ time(T_d) \quad (1)$$

Later the same experiment was done from the RPi-B source node to a PC sink node as shown in Fig. 5, the results indicate that changing the sink node does not impact T_t significantly, although there is a slight change in T_e and T_d. Because of the hardware configuration, the results may vary slightly from machine to machine. In addition, the 1280x960 image seems to require more transmission time in the PC sink node. We examined the receiving packets for both the PC and RPi-B and found a variation in the TCP receiver window size (Win) [63]. The Windows operating-system-based PC node used smaller Win than the Linux-based RPi-B [64]. The transmission latency can be shorter if the device uses a larger Win size with the help of the Win auto-scaling factor.

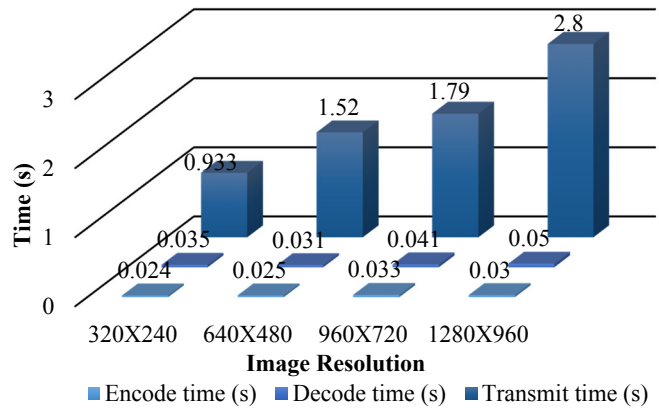

FIGURE 4. Latency of MQTT protocol with varied image resolutions measured from RPi-B source to RPi-B sink

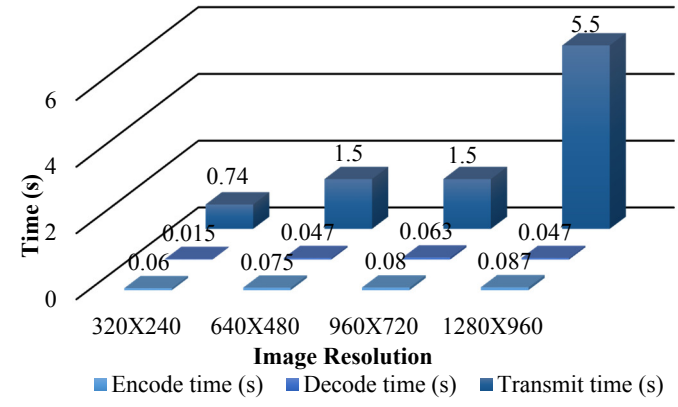

FIGURE 5. Latency of MQTT protocol with varied image resolutions measured from RPi-B source to PC sink node

[11]https://en.wikipedia.org/wiki/Pcap
[12]https://pypi.org/project/paho-mqtt/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2907525, IEEE Access

1.2) AMQP

A similar experiment was done using AMQP, utilizing both RPi and PC sink nodes. We used the Pika[13] python client library and RabbitMQ server, which is an implementation of AMQP version 0-9-1. The AMQP Publisher runs in RPi-B, and the Subscriber receives the image data through a specific topic and queue defined by the Publisher. The RabbitMQ server routed the data successfully. We repeated each test five times to calculate the average latency. Fig. 6 and Fig. 7 show the results. The results indicate that the transmission time significantly increased with the increasing image quality. Because of the receiver window size variation, the transmission time varied from the PC to RPi nodes.

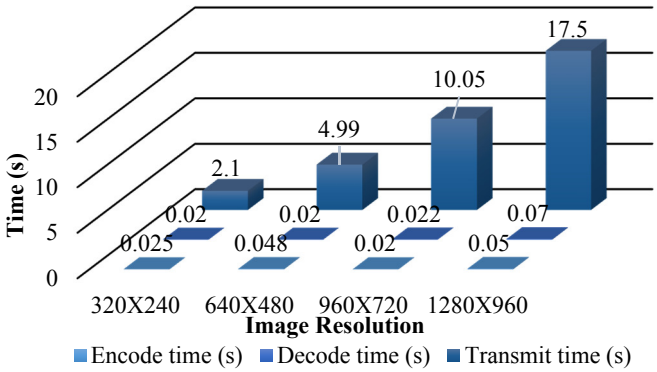

FIGURE 6. Latency of AMQP with varied image resolutions measured from RPi-B source to RPi-B sink

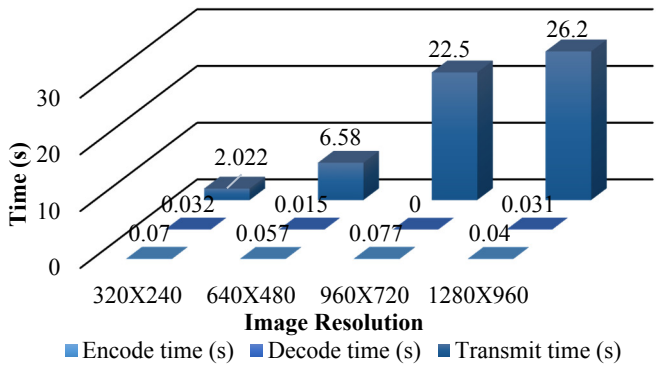

FIGURE 7. Latency of AMQP with varied image resolutions measured from RPi-B source to PC sink

1.3) HTTP

This experiment analyzed the performance of HTTP. A Simple[14] HTTP image server written in Python was set to run in the RPi-B at port 8000. The GET requests were made from the RPi-B and the PC sink node using both the python urllib module and chrome web browser. The HTTP content type was defined as the image in the server script, as well as in the binary format. We repeated each test five times and calculated the average time. The calculated T from RPi-B to RPi-B and RPi-B to PC are shown in Fig. 8 and Fig. 9, respectively. As HTTP is considered to provide a more reliable network condition, the PC node seemed to use a larger Win compared to RPi-B in this case.

[13]https://github.com/pika/pika
[14]https://docs.python.org/2/library/simplehttpserver.html

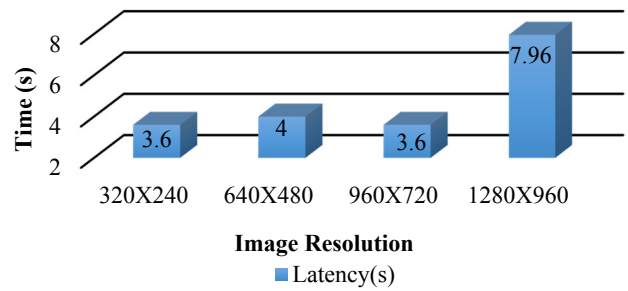

FIGURE 8. Latency of HTTP with varied image resolutions measured from RPi-B source to RPi-B sink

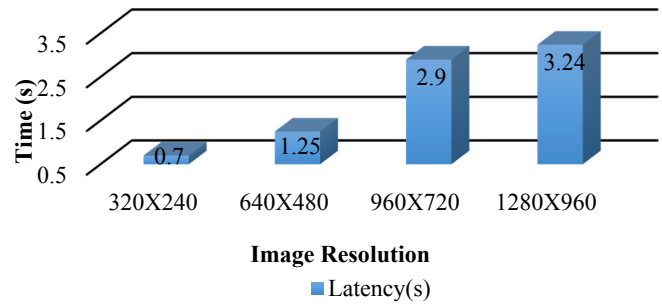

FIGURE 9. Latency of HTTP with varied image resolutions measured from RPi-B source to PC sink

1.4) XMPP

This protocol used Jabber domain and two Jabber IDs (JIDs) to perform the required tests. We utilized the Python SleekXMPP[15] library for implementing XMPP clients that can exchange image, presence, and ID in the form of XML stanza. As XMPP can transfer only small binary data, it is unable to transmit large image and video files. Therefore, its In-band[16] binary data transfer resulted in limited success, as shown in Fig.10. The results show that XMPP successfully transmitted the images of lower resolutions. The system froze while trying to transmit higher resolution images. The high-resolution images can use XMPP out-of-band[17]. Fig. 11 shows the XMPP out-of-band performance.

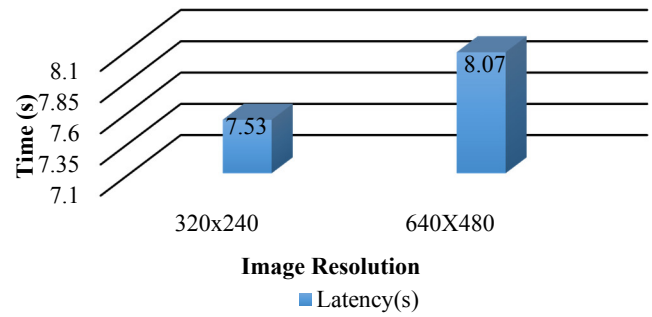

FIGURE 10. Latency of XMPP with varied image resolutions (In-band)

[15]https://github.com/fritzy/SleekXMPP
[16,17]https://xmpp.org/extensions/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2907525, IEEE Access

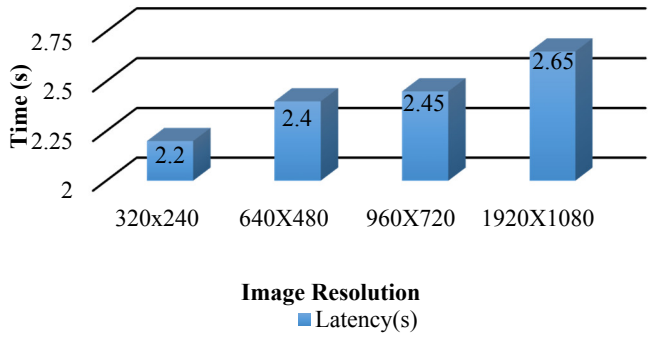

FIGURE 11. Latency of XMPP with varied image resolutions (Out-of-band)

1.5) CoAP

For CoAP, we used the txThings[18] library, which is a Python-implemented CoAP library for the twisted framework. The CoAP image server at the source node transmitted images according to the GET requests made by the client. It was a block-wise transfer with CON-type acknowledgments. CoAP successfully transmitted the images of low resolutions (320×240 and 640×480). However, unlike other protocols, the latency of CoAP seemed very large, and so we did not continue our experiments using high-resolution images (960×720, 1280×960). Fig. 12 and Fig. 13 show the results.

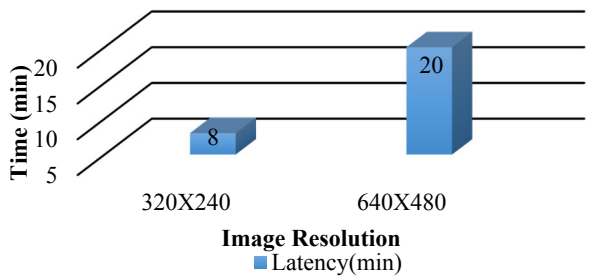

FIGURE 12. Latency of CoAP with varied image resolutions measured from RPi-B source to RPi-B sink

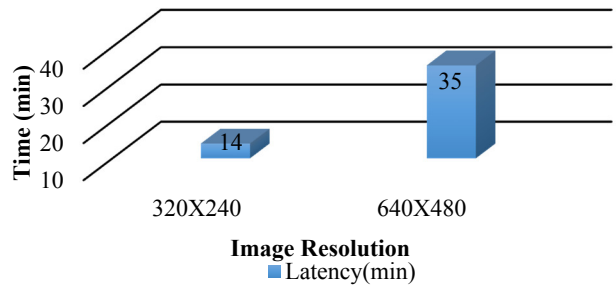

FIGURE 13. Latency of CoAP with varied image resolutions measured from RPi-B source to PC sink

[18]https://github.com/mwasilak/txThings

2) ANALYSIS OF THROUGHPUT AND PER PACKET LATENCY

In this experiment, we took the 320×240 resolution, 76.6 kB color image and analyzed the performances of various protocols by observing per-packet processing, transmission, and decoding time, as well as the throughput. We calculated the packet processing, transmission, and decoding time (per packet) as an approximate value based on the number of the payload-containing packets and their transmission delay. We also calculated the throughput (packets/s or bytes/s) using only the payload-containing packets. Table 3, Fig. 14, and Fig. 15 present the related outcomes.

TABLE 3: COMPARISON OF THROUGHPUT AND LATENCY PER PACKET

Protocols	MQTT	AMQP	HTTP	XMPP	CoAP
Packet size (Bytes)	1514	1514	1514	1514	118
Max. Payload size (Bytes)	1448	1448	1448	1448	64
Total packets	55	56	55	59	1225
Packet processing time (s)	.0009	.0005	.0014	0.0019	.0005
Packet transmit time (s)	.017	.038	.063	0.125	.2986
Packet decoding time (s)	.0005	.00093	.0009	.0009	.0009
Throughput (packets/s)	58.95	26.19	15.278	7.83	3.328
Throughput (Bytes/s)	85432	37924	22123	11338	213

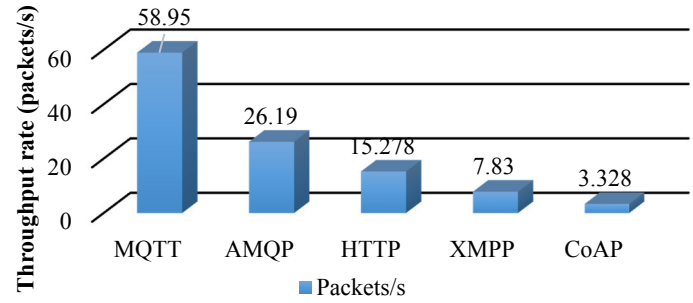

FIGURE 14. Throughput (Packets/s) comparison of different protocols

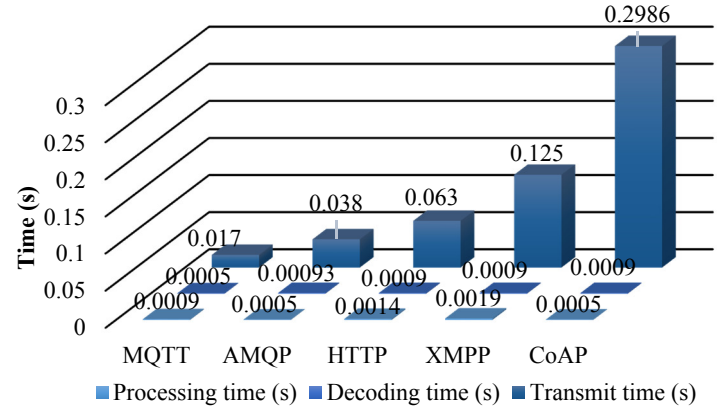

FIGURE 15. Per packet Latency of different protocols

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2907525, IEEE Access

3) MEMORY AND CPU USAGE vs. QUALITY

We performed this experiment to test the memory and CPU usage of the protocols. The "top" command in the Linux terminal helped to measure these parameters. We utilized images of various resolutions. Figs. 16, 17, and 18 show the related outcomes. Fig. 17 shows that image quality does not impact memory usage significantly. Again, memory usage for MQTT and HTTP is lower compared to the other protocols, followed by AMQP and CoAP. XMPP is the highest memory user among all the ALPs. On the other hand, the CPU usage of AMQP is highest (as shown in Fig. 18), followed by CoAP, MQTT, HTTP, and XMPP.

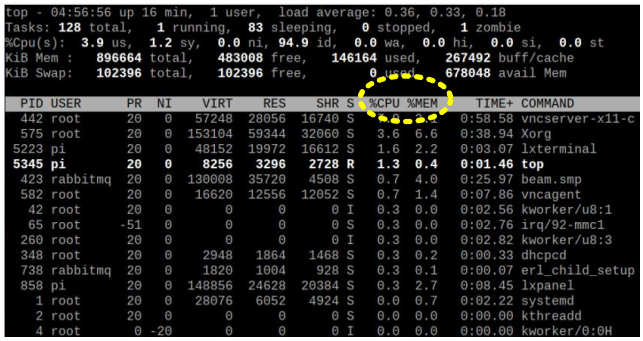

FIGURE 16. Memory and CPU usage monitoring

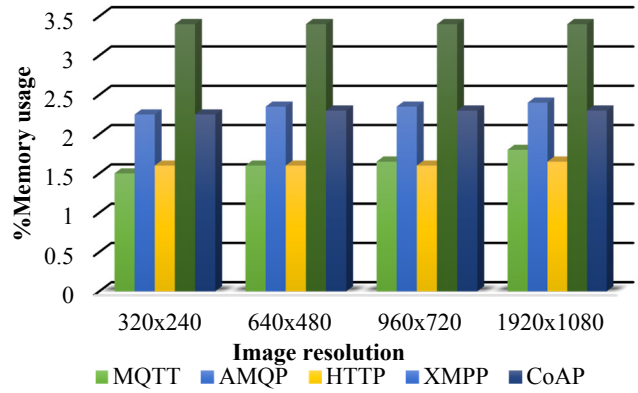

FIGURE 17. %Memory usage comparison of the protocols

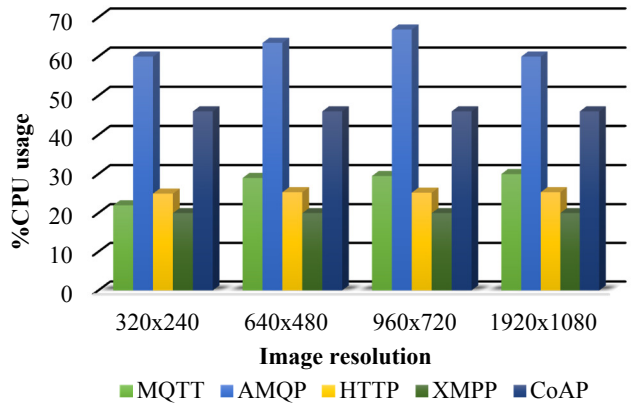

FIGURE 18. %CPU usage comparison of the protocols

4) BANDWIDTH CONSUMPTION

The experiment was also conducted to measure the average BW consumption by each protocol while transmitting the 320X240 resolution, 76.6 KB image. The maximum upload LAN speed was 7.3 Mbps. Table 4 shows the results.

TABLE 4: BANDWIDTH CONSUMPTION ANALYSIS

Protocol	MQTT	AMQP	HTTP	XMPP	CoAP
Highest BW (bits/s)	799K	279K	5840K	345K	1999
Lowest BW (bits/s)	39	68K	5496K	-	1916

5) OVERHEAD

The pcap files captured while transmitting the 76.6 KB color image were analyzed to find out total protocol overhead and other network layer overheads, as presented in Table 5 and Fig. 19. We observed the per-frame physical and network layer overheads as 14 bytes and 20 bytes respectively. The transport layer overhead of TCP (32 bytes) was different from UDP (8 bytes). The protocol overhead (ALP) includes all the exchanged data except the payload and other network overheads (including physical, network and transport layer).

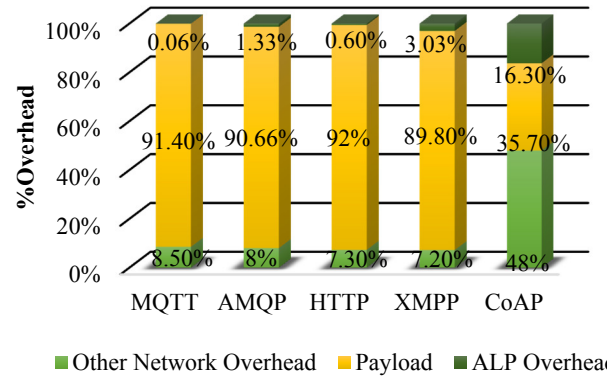

FIGURE 19. Overhead calculation

6) ENERGY CONSUMPTION

Fig. 20 shows the experimental setup to measure power and energy. We measured the voltage and current from a DC power supply (supply voltage: 5V, supply Current: 1A) while transmitting the 320X240 resolution, 76.6 KB image from the source node. Subsequently, we also calculated the power and energy consumption. Table 6 and Fig. 21 show the related results. The table shows that MQTT drew less power and consumed less energy during the process. On the other hand, XMPP was the highest power user among all the ALPs. In addition, the energy consumption of CoAP was highest due to its large transmission latency.

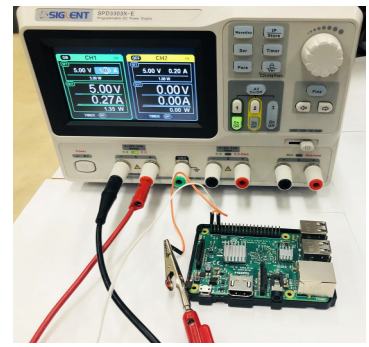

FIGURE 20. Energy measurement setup

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2907525, IEEE Access

TABLE 5: OVERHEAD CALCULATION

Protocol	MQTT	AMQP	HTTP	XMPP (In-band)	CoAP
Total No. of frames	111	105	94	95	41
Total Bytes exchange	85838	86544	85169	87384	3586
Total Payload (Bytes)	78464	78464	78464	78464	1280
Data link layer Overhead (Bytes)	1554	1470	1316	1330	574
Network layer Overhead (Bytes)	2220	2100	1880	1900	820
Transport layer Overhead (Bytes)	3552	3360	3008	3040	328
ALP Overhead (Bytes)	48	1150	501	2650	584
% Payload	91.4	90.66	92	89.8	35.7
% Other overhead	8.53	8	7.3	7.2	48
% ALP Overhead	0.056	1.33	0.6	3.03	16.3

TABLE 6: ENERGY CONSUMPTION MEASUREMENT

Protocols	MQTT	AMQP	HTTP	XMPP (In-band)	CoAP
Max Drawn[a] Current (A)	0.19	0.22	0.24	0.26	0.24
Power[b] (W)	0.95	1.1	1.2	1.3	1.2
Total latency(s)	0.992	2.145	3.6	7.53	480
Energy[b] (J=W*s)	0.942	2.3595	4.32	9.789	576
Energy per Byte (µJ)	12	30	55	124	7343
Energy per packet (mJ)	16.15	41.8	75.6	162.5	358.32

a. RPi-B Normal operating condition Current is 0.27 A at 5V.
b. Power (W) = Voltage (V) × Current (A), Energy (J) = Power (W) × Latency (s)

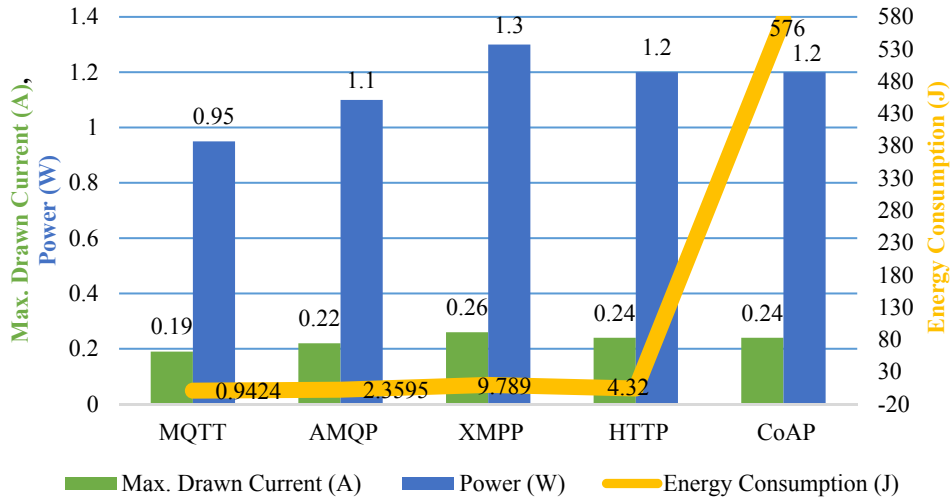

FIGURE 21. Comparison of energy consumption

7) ANALYSIS OF DDS

OpenSplice implementation of DDS performs better for small data at a high overhead cost. Moreover, it hinders implementation scalability for a large message and high data rate. On the other hand, RTI performs poorly for small data but exhibits good scalability if the message size increases. [42]. Hence, we have focused on the characteristics of DDS implementations by RTI in the LAN, where DDS uses UDP as its transport protocol. We used the rticonnextdds-connector library that enables Python to access DDS data. It works on XML Application Creation and Dynamic Data. As the architecture of DDS is complicated compared to other protocols, the tests were limited to the transmission of text data only. We transmitted text bytes of equal size to check its average latency, throughput, memory and CPU usage, bandwidth, and energy consumption. Table 7 shows the outcomes achieved from the Wireshark analysis. RTI DDS used RTPS as the data delivery protocol through UDP transport to transmit the text data. Each of the RTPS useful packets was 150 Bytes in length whereas the payload was 24 bytes.

We transmitted 10 RTPS packets and calculated the protocol overhead. The results show that DDS generated 8836 bytes (74%) of overhead while transmitting 240 bytes

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2907525, IEEE Access

(2%) of payload. BW and energy consumption also seemed high compared to the small payload (Table 7). This protocol also produced low throughput with a high per-packet latency of 2 sec., where the maximum per packet latency is 0.2986 sec., produced by CoAP. Due to this poor performance, we cannot consider DDS as a potential candidate for IoVT-based real-time video surveillance applications. Thus, the analyses on the DDS protocol is included here separately at the end of this subsection.

TABLE 7: PERFORMANCE ANALYSIS OF DDS

Latency per packet (sec)	Through-put (bytes/s)	Percentage Memory usage	Percent-age CPU usage	Energy per packet (J)	BW (bits/s)
2	56	3.4	40	2	7572

B. ANALYSIS ON SINGLE-NODE VIDEO DATA

The analysis performed on video data is described in this second category. We measured the latency of each of the protocols for video transmission. Due to the poor performance, we excluded CoAP from video data analysis. Fig. 22 shows the outcomes. MQTT performs well for video transmission. On the other hand, XMPP failed to transmit the video due to its in-band limitation. Hence, we utilized XMPP out-of-band transmission.

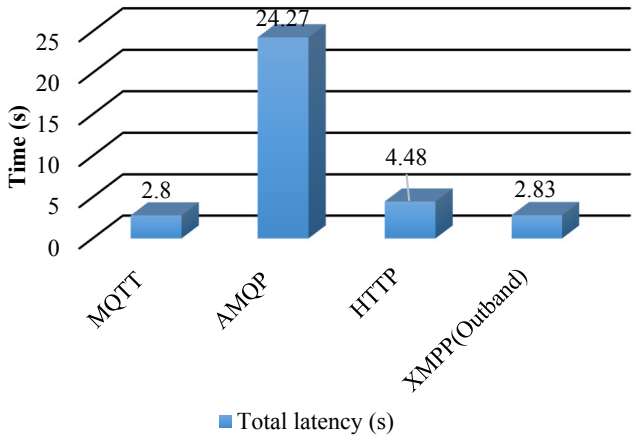

FIGURE 22. Video transmission performance comparison

C. ANALYSIS ON MULTI-NODE LATENCY

Previously, MQTT and AMQP performed well in terms of latency, throughput, and energy consumption. Hence, we conducted experiments utilizing multiple nodes to compare the delay between MQTT and AMQP. We took two color images (320×240, 76.6 kB each) and transmitted them from two different IoVT source nodes towards a common sink node. Fig. 23 and Fig. 24 show the outcomes. MQTT outperformed AMQP, and as the node number increased, latency significantly decreased for MQTT. It took only 0.065s on average for MQTT to transfer two images from two different nodes, while AMQP required 3.03s. Hence, for a double node, MQTT is 47 times faster than AMQP.

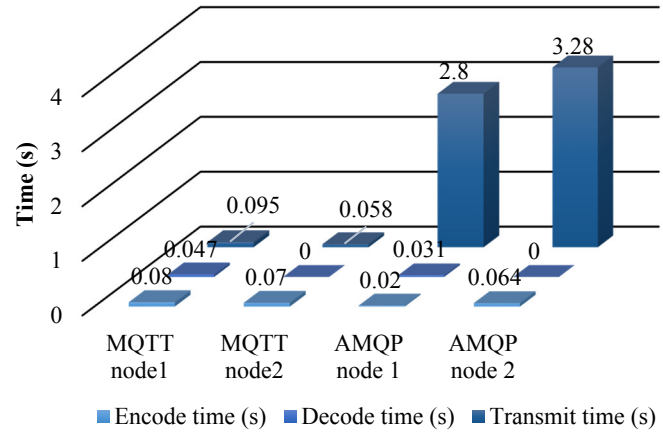

FIGURE 23. Multiple node comparison between AMQP and MQTT

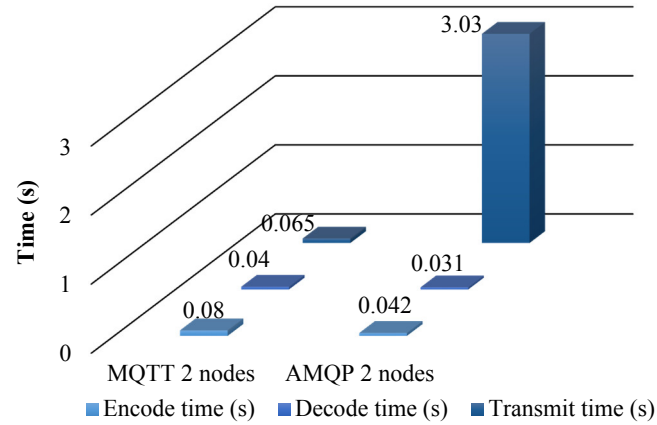

FIGURE 24: Latency comparison between AMQP and MQTT for multiple-node

IX. PERFORMANCE ANALYSIS OF MQTT PROTOCOL USING STANDARD DATASET

As MQTT outperformed other protocols, we applied this protocol to transmit various standard surveillance video sequences [41]. We also tested the transmission latency of the (320×240 resolution) color images of different formats using MQTT. We divided this analysis into two subsections, and described below.

A. TESTING OF IMAGE FORMATS

So far, we have utilized color images of JPG format only, but in this experiment, we tested the performance variation in different image formats. We chose the 320×240 resolution color image sample of various formats like JPG, TIFF, BMP, and PNG. Fig. 25 shows that JPG format achieves the lowest delay, followed by TIFF, PNG, and BMP.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2907525, IEEE Access

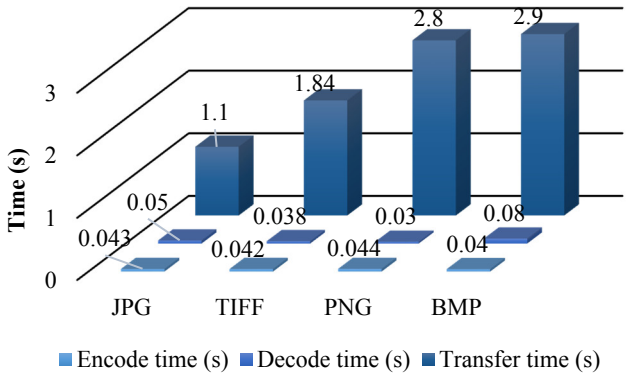

FIGURE 25. Transmission Latency of different image formats using MQTT

B. TESTING OF SURVEILLANCE VIDEO SEQUENCE

We have used standard video surveillance sequences from the online repository ViSOR and evaluated the performance of MQTT. The associated results of surveillance video testing, illustrated in Table 8 and Fig. 26, show that per byte energy consumption decreased as the video size increased.

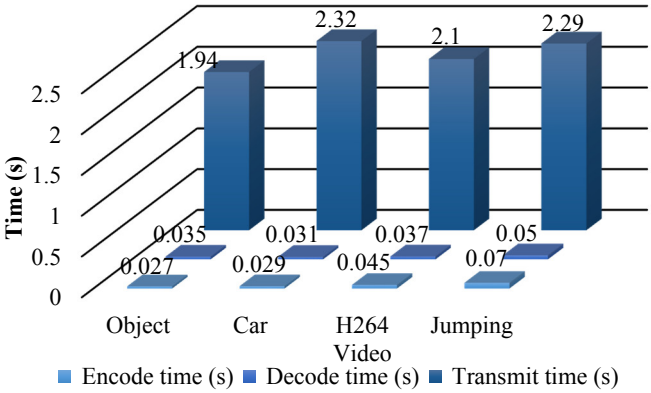

FIGURE 26. Latency of Surveillance Video sequence transmission using MQTT

TABLE 8: SURVEILLANCE VIDEO SEQUENCE [41] TESTING

Video sequence name[a]	Abandoned object 3	Entering a car	H264 video sample[b]	Jumping 1
Video format	MPEG1	AVI	H264	MPEG1
Resolution	320×256	640×480	1280×720	360×288
Frame rate (fps)	25	6.735	23.9798	25
Frame count	137	50	120	263
Video size (kB)	334	472	871	934
Encoding time (s)	0.027	0.029	0.045	0.07
Decoding time (s)	0.035	0.031	0.037	0.05
Transmit time (s)	1.94	2.32	2.1	2.29
Energy Consumption (J)	10	37.6	14.33	6.7
Energy Consumption per Byte (µJ)	4.1	3.9	1.93	1.85

a. http://www.openvisor.org
b. https://www.youtube.com/watch?v=JA-gGDXL4jk

X. COMPARATIVE DISCUSSION OF ALL PROTOCOLS

The performance analysis and comparison of ALPs for various IoVT layers are challenging in the IoT research community. Besides, MQTT, AMQP, HTTP, XMPP, CoAP, and DDS, other ALPs, like MQTT-SN, Web Socket, SMTP, and FTP, are also available in the world.

MQTT-SN is an emerging IoT protocol, but its service is limited for sensor-based networks with low data rates, which is why it is not suitable for large visual data-based surveillance applications[19]. In addition, only a few platforms support MQTT-SN implementation. Web Socket, another emerging protocol, is designed for the resource-intensive conditions which can be considered ideal for some IoT applications, but its work function is not entirely independent, in particular, it cannot interpret its handshake without an HTTP server)[20]. SMTP (Simple Mail Transfer Protocol), an old protocol, is the basis of electronic mail, and is already established and popular for mail messages. FTP (File Transfer Protocol) is another old protocol that is popular for on request file transfers on the internet. SMTP and FTP protocols were designed when there was no encryption mechanism (TLS/SSL/DTLS) available, and so, these protocols are vulnerable compared to the ALPs. Hence, we did not consider them for experimentation.

MQTT, AMQP, HTTP, XMPP, CoAP and DDS protocols were chosen instead as possible candidates for IoVT-based smart surveillance system in this work. We discussed and explained their suitability at different layers of an edge-fog-cloud-integrated IoVT based VSS. The task carried out based on their performances, as evaluated in the experiments.

From the latency analysis given in section VIIIA, it is clear that MQTT produced the shortest latency, followed by AMQP and HTTP. As the image size increased, HTTP eventually outperformed AMQP. XMPP was able to send only the lower resolutions of images, but at the cost of significant latency. Though, the experiments regarding higher resolutions of images failed due to its in-band binary data transfer limitation. XMPP overcame this utilizing out-of-band transfer. XMPP used the XEP-0066[21] plugin for this purpose. XEP-0066 allows large file sharing through HTTP or RTP links, and XMPP is used only as a signaling protocol.

Compared to others, CoAP performed poorly in terms of latency. Wireshark analysis shows that CoAP fragments the payload, which is only 64 bytes, in equal parts. Fragmentation of payload in equal parts is better for small size payloads, but worse when the payload size increases. Again, CoAP utilizes a timeout mechanism for retransmission, and so, it needs to wait for an acknowledgment (CON) for each packet. As the payload size increases, the number of fragments increases. As a result, timeouts, and delays increase. Due to this large latency for higher resolution images, it was illogical to measure the latency for them. Hence, we kept the measurements within the lower resolutions of images, ignoring the higher ones.

MQTT, HTTP, and AMQP fragment the payload based on its size. If the payload size increases, the fragment size also increases. Large fragment size results in a small number of

[19]http://www.mqtt.org/ 2013/12/mqtt-for-sensor-networks-mqtt-sn
[20]https://tools.ietf.org/html/rfc6455 [21]https://xmpp.org/extensions/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2907525, IEEE Access

acknowledgments and fewer request responses. Hence, they perform better in terms of transmission time. From Fig. 23 and Fig. 24, it is clear that MQTT performs better than AMQP, both for single and multiple nodes. It is interesting to observe that MQTT requires less time for transmission while the number of nodes is greater than one.

Although, XMPP out-of-band performs well with shorter latency, embedding visual data in the XML stanza structure produces the highest overhead compared to the other protocols. Wireshark analysis shows that XMPP costs 2650 bytes of protocol overhead out of a total of 87384 bytes while transmitting 78464 bytes (59 packets) of image payload. This high overhead causes the poor performance of XMPP, as shown in Table 5. We found that XMPP required 59 packets of data to transmit a color image, while HTTP and MQTT used only 55 packets of data.

Although the per-packet overhead of CoAP is only 4 bytes, the payload fragmentation method and timeout mechanism of CoAP increased its total number of packets (1225). Consequently, this block-wise data transfer and CON message per packet generated a substantial sum of overhead, which is 16.3% of the total bytes exchanged (Table 5).

HTTP is the proven worldwide protocol for the internet. But it has limited compatibility with the constrained communication environment when the application is running in a device like a smartphone or a Raspberry Pi. Here the additional overhead associated with request/response for this protocol can negatively affect the environment of a constrained network. For HTTP, if the number of connected devices increases, the protocol overhead becomes critical [43]. During the experiment, we ran HTTP image and video servers in the Raspberry Pi node and tried to request visual data from another Raspberry Pi and a PC node, first simultaneously, then one after the other. HTTP failed to process two simultaneous GET requests made from two different devices while it was running in a constrained environment like Raspberry Pi. In addition, it produced 501 bytes of additional application layer overheads to transmit 78464 bytes of image data.

AMQP, which produced a shorter per-packet overhead of 8 bytes appears as a promising solution for exchanging large visual data from publisher to subscriber. It induced 1150 bytes of ALP overhead. On the other hand, the per-packet overhead of MQTT is only 2 bytes. Consequently, while transmitting the same image payload, MQTT produced a total of 48 bytes of ALP overhead, which is only 0.056% of the total bytes exchanged (Table 5). Hence, MQTT is proven to be the most lightweight protocol of them all. It is useful for connections with remote locations where a small code footprint is required, and network bandwidth is at a premium.

In addition, MQTT took only 17ms to transmit a 1514 byte packet (Fig. 14 and Fig. 15). It is the lowest per-packet latency measured during the experiments. The highest throughput (packets/s) is also achieved by this protocol which is around 59 payload-containing packets/s. Compared to MQTT, AMQP also performed well. The throughput rate of CoAP and XMPP seem significantly lower compared to MQTT and AMQP.

MQTT is highly energy efficient, as shown in the energy consumption measurement table (Table 6), followed by AMQP and HTTP. XMPP consumes a bit more energy, while CoAP consumes the highest. The transmission latency substantially affected the total energy consumption of each protocol.

We also analyzed the protocols in terms of BW (Table 4). In addition, there was 0 packet loss during the experiments. The analysis shows that HTTP always consumed a high BW compared to other protocols. In the case of MQTT and AMQP, the BW consumption varied according to the network conditions. But for CoAP, the BW consumption was almost constant and efficiently low. For AMQP and XMPP, it is much better than HTTP. For MQTT, the range was wide. The BW consumption by MQTT was as high as 799K bits/s and as low as 39 bits/s. Hence, depending on the network situation, MQTT controls its BW consumption efficiently.

As AMQP and MQTT performed well for single-node analysis, we performed multiple-node analysis on these two protocols. The result shows that MQTT outperformed AMQP, being 47 times faster in terms of average latency (Fig. 23 and Fig. 24). The underlying TCP protocol ensured reliable transmission of MQTT even though the QoS was 0.

Finally, we chose the MQTT protocol to perform some more experiments based on image formats and standard surveillance video sequences. The test result shows that, among different available image formats, JPG format works best for low-latency IoVT applications. The surveillance video sequence test shows that MQTT can handle large visual data efficiently. As the video size increased, MQTT significantly reduced its transmission delay and per-byte energy consumption. These outcomes prove the suitability of MQTT in real-time surveillance applications. Table 9 summarizes the overall analysis of comparative performances.

TABLE 9: PERFORMANCE SUMMARY HIGH TO LOW (FROM LEFT TO RIGHT)

Performance:	**Highest** →				**Lowest**
Latency	CoAP	XMPP	HTTP	AMQP	MQTT
Throughput	MQTT	AMQP	HTTP	XMPP	CoAP
% protocol Overhead	CoAP	XMPP	AMQP	HTTP	MQTT
%CPU usage	AMQP	CoAP	MQTT	HTTP	XMPP
%Memory usage	XMPP	AMQP	CoAP	MQTT/ HTTP	MQTT/ HTTP
BW (bits/s) consumption	HTTP	XMPP	AMQP	MQTT	CoAP
Energy consumption (J)	CoAP	XMPP	HTTP	AMQP	MQTT

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2907525, IEEE Access

Security and privacy issues create a great challenge for IoT systems [59], and the research regarding these issues will continue to persist [56]. The evaluation of present and future critical security issues in the IoT world [59] and possible solutions regarding security threats in fog computing [56] and edge computing [60] encourage future research on this topic. Hence, in the future, we need to analyze privacy and threats, as well, especially the additional overhead they might bring. The security problem in IoT systems itself is a vast research area, and is beyond the scope of this present work.

XI. CONCLUSION

This paper presented an evaluation of several widely accepted ALPs for next-generation video surveillance systems using an IoVT framework. We implemented real-time surveillance test applications with image and video data and measured different communication performance testing parameters induced by these protocols, such as latency, throughput, BW and energy consumption, overhead, memory usage, and CPU usage. The overall analysis helped us rank the ALPs for different video surveillance scenarios. MQTT can become a matured platform for implementing IoVT based real-time video surveillance applications in a constrained environment. It can transmit visual data from an IoVT edge-node, consuming less BW and with short latency. Though HTTP is most used and adapted for different applications, its large overhead and BW consumption make it inappropriate as an IoVT edge-node protocol in a constrained environment. We can propose its appropriate use for communication between a resourceful fog and an unconstrained cloud in an IoVT video surveillance case. AMQP can also efficiently handle large visual data. But the observations proved that if the node number and data size increases, it yields higher latency. CoAP seems to consume much lower BW and costs less overhead per packet, but the significant delay in transmitting large visual data makes it inappropriate for an IoVT edge-node. Due to the complex architecture, DDS already lacks sufficient deployments. The experiments also confirm that this protocol possesses high overhead, high energy consumption, and large latency which discourage its usage in a constrained IoVT node. XMPP is also not suitable for deployment in IoVT-edge nodes because of its high overhead and latency in data transmission. We need to analyze other features (e.g., security) in the future as well to check the change of performance of the ALPs. The work done in this paper will create exciting opportunities in new IoVT-based VSS architectures that will need to combine IoVT edge, fog, and cloud computing systems based on the appropriate selection of underlying ALPs.

ACKNOWLEDGMENT

The authors thank Rashedul Hasan, Mohammad Wajih Alam, and Md. Hanif Ali Shohag for their suggestions in this work.

REFERENCES

[1] S. W. Yang, O. Tickoo, and Y. K. Chen, "A framework for visual fog computing," in *2017 IEEE Int. Symposium on Circuits and Systems (ISCAS),* Baltimore, MD, 2017 pp. 1-4.

[2] V. Gouaillier and A. Fleurant, "Intelligent video surveillance: Promises and challenges," *Technological and Commercial Intelligence Report, CRIM and Technpole Defence and Security,* vol. 456, pp. 468, 2009.

[3] "Top video surveillance trends for 2018," [Online]. Available: https://ihsmarkit.com/Info/1217/top-video-surveillance-trends-2018.html. Accessed on: 22-Oct-2018.

[4] J. Dizdarević, F. Carpio, A. Jukan, and X. Masip-Bruin, "A Survey of Communication Protocols for Internet of Things and Related Challenges of Fog and Cloud Computing Integration," *ACM Comput. Surv.*, vol. 51, no. 6, pp. 1–29, Jan. 2019.

[5] A. Botta, W. de Donato, V. Persico, and A. Pescapé, "On the Integration of Cloud Computing and Internet of Things," in *2014 Int. Conf. on Future Internet of Things and Cloud*, Barcelona, 2014, pp. 23-30.

[6] Chengjia Huo, Ting-Chou Chien, and P. H. Chou, "Middleware for IoT-Cloud Integration Across Application Domains," *IEEE Des. Test*, vol. 31, no. 3, pp. 21–31, Jun. 2014.

[7] G. Fortino, A. Guerrieri, W. Russo, and C. Savaglio, "Integration of agent-based and Cloud Computing for the smart objects-oriented IoT," *Proc. of the 2014 IEEE 18th Int. Conf. on Computer Supported Cooperative Work in Design (CSCWD)*, Hsinchu, 2014, pp. 493-498.

[8] D. Elliott, "Intelligent video solution: A definition," *Security,* vol. 47, no. 6, pp. 46–48, 2010.

[9] Honghai Liu, Shengyong Chen, and N. Kubota, "Intelligent Video Systems and Analytics: A Survey," *IEEE Trans. Ind. Informatics*, vol. 9, no. 3, pp. 1222–1233, Aug. 2013.

[10] A. Mohan *et al*, "Internet of video things in 2030: A world with many cameras," in *2017 IEEE Int. Symposium of Circuits and Systems (ISCAS),* Baltimore, MD, 2017, pp. 1-4.

[11] A. Sammoud, A. Kumar, M. Bayoumi, and T. Elarabi, "Real-time streaming challenges in Internet of Video Things (IoVT)," in *2017 IEEE Int. Symposium on Circuits and Systems (ISCAS)*, 2017, pp. 1-4.

[12] Y. Mori, X. T. Nguyen, and C. K. Pham, "Reliable and energy-efficient transmission on the Internet-of-Video-Things," in *2017 17th Int. Symposium on Communications and Information Technologies (ISCIT)*, Cairns, QLD, 2017, pp. 1-4.

[13] S. T. Lin, Y. H. Liao, Y. Tsao, and S. Y. Chien, "Object-based on-line video summarization for internet of video things," *IEEE Int. Symposium on Circuits and Systems (ISCAS)*, Baltimore,2017, pp.1-4

[14] N. Naik, "Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP," in *2017 IEEE Int. Systems Engineering Symposium (ISSE)*, Vienna, 2017, pp. 1-7.

[15] J. Zhou *et al.*, "CloudThings: A common architecture for integrating the Internet of Things with Cloud Computing," *in Proc. of the 2013 IEEE 17th Int. Conf. on Computer Supported Cooperative Work in Design (CSCWD),* Whistler, BC, 2013, pp. 651–657.

[16] C. Stergiou, K. E. Psannis, B.-G. Kim, and B. Gupta, "Secure integration of IoT and Cloud Computing," *Futur. Gener. Comput. Syst.*, vol. 78, pp. 964–975, Jan. 2018.

[17] C. Stergiou and K. E. Psannis, "Recent advances delivered by Mobile Cloud Computing and Internet of Things for Big Data applications: a survey," *Int. J. Netw. Manag.*, vol. 27, no. 3, p. e1930, May 2017.

[18] A. P. Plageras *et al*., "Efficient Large-scale Medical Data (eHealth Big Data) Analytics in Internet of Things." in *2017 IEEE 19th Conf. on Business Informatics (CBI),* Thessaloniki, 2017, pp. 21-27.

[19] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proc. of the First Edition of the MCC Workshop on Mobile Cloud Computing,* 2012, pp. 13-16.

[20] I. Stojmenovic and S. Wen, "The Fog computing paradigm: Scenarios and security issues," in *2014 Federated Conf. on Computer Science and Information Systems,* Warsaw, 2014, pp. 1-8.

[21] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, "Fog Computing: A Platform for Internet of Things and Analytics," Springer, Cham, 2014, pp. 169–186.

[22] S. Yi, C. Li, and Q. Li, "A survey of fog computing: Concepts, applications and issues," in *Proc. of the 2015 Workshop on Mobile Big Data - Mobidata'15,* Hangzhou, 2015, pp. 37-42.

[23] I. Stojmenovic, S. Wen, X. Huang, and H. Luan, "An overview of Fog computing and its security issues," *Concurr. Comput. Pract. Exp.*, vol. 28, no. 10, pp. 2991–3005, Jul. 2016.

[24] G. Kokkonis, K. E. Psannis, M. Roumeliotis, and D. Schonfeld, "Real-time wireless multisensory smart surveillance with 3D-HEVC streams for internet-of-things (IoT)," *J. Supercomput.*, vol. 73, no. 3, pp. 1044–1062, Mar. 2017.

[25] G. Kokkonis, K. E. Psannis, M. Roumeliotis, Y. Ishibashi, B.-G. Kim, and A. G. Constantinides, "Transferring Wireless High Update Rate Supermedia Streams Over IoT," Springer, Cham, 2018, pp. 93–103.

[26] A. P. Plageras, K. E. Psannis, Y. Ishibashi, and B. Kim, "IoT-based surveillance system for ubiquitous healthcare," in *42nd Annual Conf. of the IEEE Industrial Electronics Society*, 2016, pp. 6226-6230.

[27] C. Stergiou, K. E. Psannis, A. P. Plageras, G. Kokkonis, and Y. Ishibashi, "Architecture for security monitoring in IoT environments," in *2017 IEEE 26th Int. Symposium on Industrial Electronics (ISIE)*, Edinburgh, 2017, pp. 1382-1385.

[28] V. A. Memos, K. E. Psannis, Y. Ishibashi, B.-G. Kim, and B. B. Gupta, "An Efficient Algorithm for Media-based Surveillance System (EAMSuS) in IoT Smart City Framework," *Futur. Gener. Comput. Syst.*, vol. 83, pp. 619–628, Jun. 2018.

[29] H. S. Parekh, D. G. Thakore, and U. K. Jaliya, "A survey on object detection and tracking methods," *IJIRCCE,* vol. 2, *(2),* pp. 2970-2978, 2014.

[30] K. S. Huang and M. M. Trivedi, "Video arrays for real-time tracking of person, head, and face in an intelligent room," *Mach. Vis. Appl.*, vol. 14, no. 2, pp. 103–111, Jun. 2003.

[31] H. Liu, S. Chen and N. Kubota, "Intelligent Video Systems and Analytics: A Survey," *IEEE Trans. Ind. Informatics*, vol. 9, no. 3, pp. 1222–1233, Aug. 2013.

[32] E. Fellores, "Chilean salmon farm enhances security with intelligent video edge," Security, p. 22, 2008.

[33] R. Chang, L. Guan, and J. A. Burne, "An automated form of video image analysis applied to classification of movement disorders," *Disabil. Rehabil.,* vol. 22, *(1-2),* pp. 97-108, Jan. 2000.

[34] G. Diamantopoulos and M. Spann, "Event detection for intelligent car park video surveillance," *Real Time Imaging,* vol. 11, *(3),* pp. 233-243, Jun. 2005.

[35] T. Wang, Z. Zhu, and E. Blasch, "Bio-inspired adaptive hyperspectral imaging for real-time target tracking," *IEEE Sens. J.,* vol. 10, *(3),* pp. 647-654, Mar. 2010.

[36] C. Quek, W. Irawan, and E. Y. K. Ng, "A novel brain-inspired neural cognitive approach to SARS thermal image analysis," *Expert Syst. Appl.*, vol. 37, no. 4, pp. 3040–3054, Apr. 2010.

[37] Z. B. Babovic, J. Protic, and V. Milutinovic, "Web Performance Evaluation for Internet of Things Applications," *IEEE Access*, vol. 4, pp. 6974-6992, 2016.

[38] H. Sardeshmukh and D. Ambawade, "Internet of Things: Existing protocols and technological challenges in security," in *2017 Int. Conf. on Intelligent Computing and Control (I2C2)*, 2017, pp. 1-7.

[39] M. H. Amaran, N. A. M. Noh, M. S. Rohmad, and H. Hashim, "A Comparison of Lightweight Communication Protocols in Robotic Applications," *Procedia Comput. Sci.*, vol.76, pp. 400–405, Jan. 2015.

[40] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," *IEEE Commun. Surv. Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[41] R. Vezzani, R. Cucchiara, "Video Surveillance Online Repository (ViSOR): an integrated framework", *Multimedia Tools and Applications*, vol. 50, n. 2, Kluwer Academic Press, pp. 359-380, 2010

[42] P. Bellavista, A. Corradi, L. Foschini and A. Pernafini, "Data Distribution Service (DDS): A performance comparison of OpenSplice and RTI implementations," in *2013 IEEE Symposium on Computers and Communications (ISCC)*, pp. 377-383.

[43] T. Yokotani and Y. Sasaki, "Transfer protocols of tiny data blocks in IoT and their performance evaluation," in *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, 2016, pp. 54-57.

[44] G, Christian. "ONVIF security recommendations," *ONVIF white paper*, pp. 1-12, 2010

[45] T. Pflanzner and A. Kertesz, "A survey of IoT cloud providers," *2016 39th Int. Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO),* 2016, pp. 730-735.

[46] S. M. Babu, A. J. Lakshmi, and B. T. Rao, "A study on cloud based Internet of Things: CloudIoT," in *2015 Global Conf. on Communication Technologies (GCCT)*, Thuckalay, 2015, pp. 60-65.

[47] J. Granjal, E. Monteiro, and J. Sa Silva, "Security for the Internet of Things: A Survey of Existing Protocols and Open Research Issues," *IEEE Commun. Surv. Tutorials*, vol. 17, no. 3, pp. 1294–1312, 2015.

[48] O. Hahm, E. Baccelli, H. Petersen, and N. Tsiftes, "Operating Systems for Low-End Devices in the Internet of Things: A Survey," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 720–734, Oct. 2016.

[49] M. Marjani *et al.*, "Big IoT Data Analytics: Architecture, Opportunities, and Open Research Challenges," in *IEEE Access*, vol. 5, pp. 5247-5261, 2017.

[50] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, and S. Clarke, "Middleware for Internet of Things: A Survey," in *IEEE Internet of Things J.*, vol. 3, no. 1, pp. 70-95, Feb. 2016.

[51] S. S. Solapure and H. Kenchannavar, "Internet of Things: A survey related to various recent architectures and platforms available," in *2016 Int. Conf. on Advances in Computing, Communications and Informatics (ICACCI)*, Jaipur, 2016, pp. 2296-2301.

[52] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks," RFC 4944 (Proposed Standard), Internet Engineering Task Force, Sep. 2007, updated by RFC 6282. [Online]. Available: http://www.ietf.org/rfc/rfc4944.

[53] L. De Nardis and M. Di Benedetto, "Overview of the IEEE 802.15.4/4a standards for low data rate Wireless Personal Data Networks," in *2007 4th Workshop on Positioning, Navigation and Communication*, Hannover, 2007, pp. 285-289.

[54] M. Chiang and T. Zhang, "Fog and IoT: An Overview of Research Opportunities," in *IEEE Internet of Things J.*, vol. 3, no. 6, pp. 854-864, Dec. 2016.

[55] M. Chernyshev, Z. Baig, O. Bello, and S. Zeadally, "Internet of Things (IoT): Research, Simulators, and Testbeds," in *IEEE Internet of Things J.*, vol. 5, no. 3, pp. 1637-1647, Jun. 2018.

[56] J. Ni, K. Zhang, X. Lin, and X. S. Shen, "Securing Fog Computing for Internet of Things Applications: Challenges and Solutions," *IEEE Commun. Surv. Tutorials*, vol. 20, no. 1, pp. 601–628, 2018.

[57] C. Long, Y. Cao, T. Jiang, and Q. Zhang, "Edge Computing Framework for Cooperative Video Processing in Multimedia IoT Systems," *IEEE Trans. Multimed.*, vol. 20, no. 5, pp. 1126–1139, May 2018.

[58] S. S. N. Perala, I. Galanis, and I. Anagnostopoulos, "Fog Computing and Efficient Resource Management in the era of Internet-of-Video Things (IoVT)," in *2018 IEEE Int. Symposium on Circuits and Systems (ISCAS)*, Italy, 2018, pp. 1-5.

[59] M. Frustaci, P. Pace, G. Aloi, and G. Fortino, "Evaluating Critical Security Issues of the IoT World: Present and Future Challenges," *IEEE Internet of Things J.*, vol. 5, no. 4, pp. 2483-2495, Aug. 2018.

[60] M. Endler, A. Silva, and R. A. M. S. Cruz, "An approach for secure edge computing in the Internet of Things," in *2017 1st Cyber Security in Networking Conf. (CSNet)*, Rio de Janeiro, 2017, pp. 1-8.

[61] B. Skrbic, D. Radovanovic, S. Tomovic, L. Lazovic, Z. Zecevic, and I. Radusinovic, "A decentralized platform for heterogeneous IoT networks management," *2018 23rd Int. Scientific-Professional Conf. on Information Technology (IT)*, Zabljak, 2018, pp. 1-4.

[62] S. Mijovic, E. Shehu, and C. Buratti, "Comparing application layer protocols for the Internet of Things via experimentation," in *IEEE 2nd Int. Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI)*, Bologna, 2016, pp. 1-5.

[63] J. Yan and B. Plattner, "A Simple Solution to Find the Distribution of TCP Window Sizes," *IEEE Commun. Lett.*, vol. 17, no. 2, pp. 417–419, Feb. 2013.

[64] T. Nogiwa and K. Hirata, "Identification of TCP congestion control algorithms with convolution neural networks," in *2018 Int. Conf. on Information Networking (ICOIN)*, Chiang Mai, 2018, pp. 663-665.

[65] B. Cheng, S. Zhao, J. Qian, Z. Zhai, and J. Chen, "Lightweight Service Mashup Middleware With REST Style Architecture for IoT Applications," *IEEE Trans. Netw. Serv. Manag.*, vol. 15, no. 3, pp. 1063–1075, Sep. 2018.

[66] B. Cheng, M. Wang, S. Zhao, Z. Zhai, D. Zhu, and J. Chen, "Situation-Aware Dynamic Service Coordination in an IoT Environment," *IEEE/ACM Trans. Netw.*, vol. 25, no. 4, pp. 2082–2095, Aug. 2017.