An Adaptive Solution for Images Streaming in Vehicle Networks Using MQTT Protocol

Ming-Fong Tsai[1(✉)], Thanh-Nam Pham[2,3], Fu-Hsiang Ching[2], and Le-Hung Chen[4]

[1] Department of Electronic Engineering, National United University, Miaoli, Taiwan
mingfongtsai@gmail.com
[2] Department of Information Engineering and Computer Science, Feng Chia University, Taichung, Taiwan
[3] Department of Electronics and Communications Technology, Thai Nguyen University of Information and Communications Technology, Thái Nguyên, Vietnam
[4] Image Processing Section, Advanced Engineering Division, Hua-chuang Automobile Information Technical Center Company, Taichung, Taiwan

Abstract. In this study, we explored solutions to improve the quality of real-time image transmission in vehicle networks. We deployed multiple cameras in each vehicle to collect scene data on the road. Then, the collected data was transmitted to a streaming server through a gateway and using a 4G internet connection. We use the MQTT protocol to implement our system since this is a protocol designed specifically for Internet of Things technologies and has several advantages in terms of image streaming. In addition, in order to adapt to the change in bandwidth channel due to the movement of vehicles, we propose an algorithm to control the quality of image capture which is based on threshold levels. This algorithm is based on the current throughput of local network nodes, as compared with threshold values, to control the rate of sending data from each local node in subsequent transmissions. The results of simulation show that our proposed network significantly reduces both end-to-end delay and the delay in arrival of messages in the network when the number of nodes increases. The experimental results showed that the collected images are of high quality and allow accurate analysis of the surrounding environment of the moving vehicles.

Keywords: Internet of vehicles · MQTT protocol · 4G network
Image streaming

1 Introduction

In recent years, systems for driving assistance and the monitoring of vehicles have become increasingly prevalent, and now involve many types of applications, such as emergency vehicle notification systems, collision avoidance systems and car navigation systems. These systems are characterized by the use of Internet of Things (IoT) technology combined with wireless communication protocols. These protocols are generally

Y.-B. Lin et al. (Eds.): IoTaaS 2017, LNICST 246, pp. 263–275, 2018.
https://doi.org/10.1007/978-3-030-00410-1_31

mobile ad hoc networks (VANET) or Mesh networking for short-range communications, and WiMAX (IEEE 802.16) or Global System for Mobile Communications 3G for long-range communications. These systems collect data from sensors and cameras placed on vehicles or roadside units and combined with the use of a GPS device can support driving safety applications and other related systems. However, these forms of communication have several disadvantages such as large delays and high rates of data loss. Today, fourth-generation telecommunications technology (4G) has been implemented which has high data transmission rates. This has enabled the development of a variety of services built for on-road users, particularly real-time video streaming services. The 4G network covers Taiwan, and in this country we have developed and implemented an intelligent system for monitoring vehicles, using a network of cameras installed on the vehicles. The captured images are transmitted to a cloud streaming server using the Message Queuing Telemetry Transport (MQTT) protocol. Unlike other implemented systems such as [1, 3], our system uses a Raspberry Pi platform as local nodes mounted on each individual vehicle; each node is connected to a camera, collects data from the camera and directly transmits these data to the cloud server using the 4G network. MQTT [7] is a publish/subscribe messaging protocol for constrained Internet-of-Things devices and unstable networks such as VANETs, which have high latency, low-bandwidth and unreliable channels. Thus, the MQTT protocol is an ideal approach for machine-to-machine applications, such as those reported in [4–7]. However, in contrast to these studies, we have used the MQTT protocol in VANETs for real-time image streaming, with Raspberry Pi modules functioning as IoT nodes.

In this study, we organized each vehicle as a local IoT network. In particular, the cameras as the IoT nodes are located at different locations on the vehicle to collect data surrounding the vehicle. In each local IoT network, nodes send data to a streaming server via a gateway and use the MQTT protocol. We deployed many Raspberry Pi modules to monitor and control the data collection at IoT nodes. To adapt to the changes of available bandwidth due to the constantly changing channels and movement of vehicles, we have developed an algorithm that enables the system to automatically control the quality of the captured image in the cameras, based on the current throughput of capturing nodes. The total throughput of all capturing nodes on a vehicle is divided into threshold levels. When the total throughput is small, corresponding to a low threshold level, the quality of the captured images is reduced accordingly; when the total throughput is large, corresponding to a high threshold level, the quality of capturing images is increased to match. This enables the system to constantly adapt to changes in the channel.

Since there are a high number of captured images from the cameras, the use of a traditional server by the system would require too much processing time, making it slow to respond in real time. However, the recent development of cloud computing technology allows the processing of large amounts of data and fast responses in real time. The proposed system uses a cloud streaming server to reduce response time and allow the expansion of the management system to larger numbers of vehicles. The remainder of this paper is organized as follows: Sect. 2 provides a description of the proposed system architecture and its components. Section 3 describes the analysis of the system and its performance parameters. Section 4 reports the implementation of the system and the results of simulation. The last section presents the conclusion

2 System Architecture

2.1 System Overview

Figure 1 describes the overview of the architecture of the proposed system. The system includes local MQTT client nodes installed in a vehicle, an MQTT broker (gateway) and a cloud server.

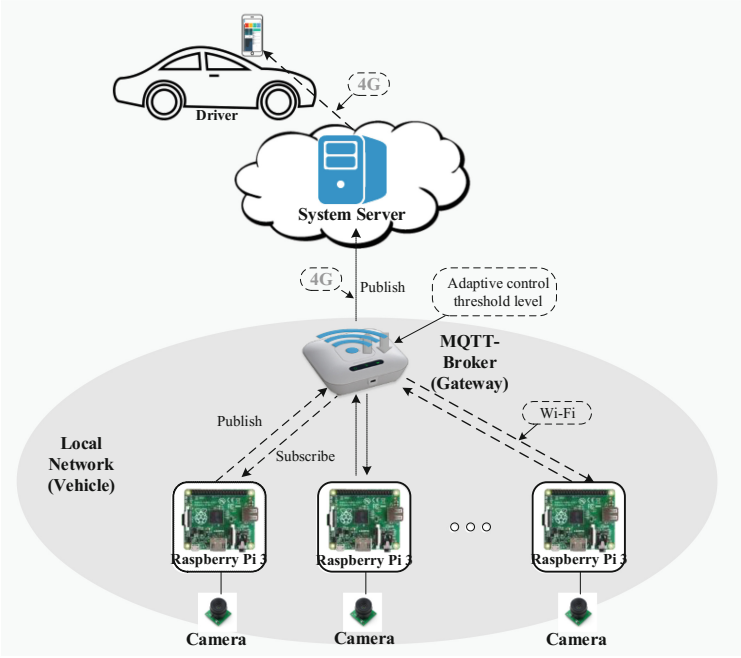

Fig. 1. Overview of the architecture of the proposed system.

In this system, each local node (MQTT Client) is connected to a camera. Cameras are mounted on the vehicle to capture images of the environment surrounding the vehicle. In our experiments, between one and six cameras were installed on a vehicle. The MQTT clients were connected to a MQTT broker through wi-fi connections. Each MQTT client registers a topic ID in the MQTT broker, and this registration process is referred to as subscribing. Each MQTT client sends data frequently to the subscribed topic in the MQTT broker. Our topic is formatted as vehicles/car-ip/node-ip, where each car has a unique identifier, and each MQTT client connected to a capturing camera also has a unique identifier. The MQTT broker acts as a gateway in this system. The MQTT broker receives real-time data from local nodes and forwards this to a cloud streaming server. When data is received, the cloud server checks its format; if correctly formatted, the server begins live-streaming data. When the user wants to view the real-time streaming data of the car, access to the streaming server follows the URI path of

resources, and users can access the system resources through a web browser running on their smartphones. The system requires user's smartphone to have a 4G connection. In addition to the transmission of image data, the node also sends information about its data speed to the gateway. The gateway is responsive to the adaptive adjustments to the quality of image capturing at all connecting nodes. After receiving information on the data speed of all local nodes, the gateway calculates the total current data speed over these nodes and compares it with pre-defined threshold levels to determine whether the image quality corresponding to that data rate is high; if not, it sends a 'publish' message to all subscribed topics in all local nodes to immediately adjust the quality of image capturing. In addition to sending image data, the gateway also sends GPS data (latitude, longitude, time) to the cloud server. This GPS data is then associated with the live images of specific locations along the route.

2.2 System Components

This section gives details of the components in the proposed system and their related processes.

Figure 2 shows the details of the components and modules in the system. We can divide the system into two major components: the IoT network placed on the vehicle and the cloud server which stores the data of the system as a whole. The sub-components are described below.

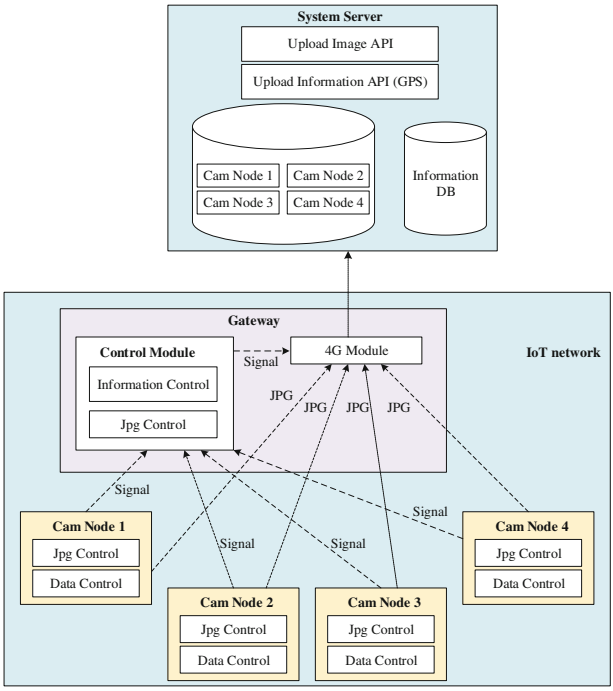

Fig. 2. Components/modules of the proposed system.

Cam Nodes. In the proposed system, the hardware platforms used to implement these cam nodes are the Raspberry Pi 3 and the Pi camera. First, a cam node connects to the gateway and initializes the environment variables. When ready, the system begins a routine periodic calculation, and the result returned is the threshold level of the subscribed channel. A cam node has two primary activities: listening and publishing. Initially, the value of threshold level is set to Level 4, which corresponds to the value of Level 95 in the JPEG image compression standard. In view of the characteristics of 4G channels, five threshold levels are used here to divide the quality of the input images into Levels 1 to 5, corresponding respectively to Levels 1, 25, 50, 95 and 100 of the JPEG standard. A greater value of the threshold level means a higher quality of image capture. When a cam node is informed of which threshold level to use, it acquires images with a size based on this threshold level. Finally, the local node begins publishing images to the subscribed topic ID in the MQTT broker. The number of images captured is also published to the MQTT broker, with an average of 8–10 images per second.

The gateway collects information about the speed of data streaming from these cam nodes and calculates the total amount of their data throughput from which to perform adaptive control. The control information will be sent back from the gateway to these nodes, and the cameras will proceed to adjust the quality of the collected image based on the control data.

Gateway. The gateway acts as a local wireless access point for the MQTT clients to connect to it via a wi-fi connection, and then connects to the cloud server via the 4G network. Thus, MQTT clients can send data to the online server. In addition, the gateway also acts as a controller which can make decisions on changing the rate of data transfer at each cam node. The hardware platform used in this system is a Raspberry Pi 3 combined with a 4G module.

The initialization processes of the gateway and the local nodes are very similar. Firstly, several environment variables are initialized and the data rate of each local node (node-rate) is subscribed. This subscription allows the gateway to monitor the current throughput of each node, indicated by the parameter node-rate. Following this, the gateway evaluates the information from all existing nodes, and determines whether the total throughput is higher or lower than the current threshold value. Following this, the new threshold level is decided based on this comparison. The gateway then publishes the new threshold level to all the connecting nodes. On receiving the new threshold level, local nodes adjust the image capturing based on this level. In addition to sending data to the server, the gateway also includes GPS data (latitude, longitude) to the MQTT server to provide information about the current location of the vehicle on the road.

System Server. The system server is a streaming server. Streaming server will receive data from the MQTT clients and display in real time to users through a web interface. The streaming server is composed of three functional components: the *blackhole* function, the *streaming* function and the *GPS* function. The *blackhole* function checks the format of the uploaded images. If the format is correct, these images will be saved in the storages. The *streaming* function is responsible for displaying real-time image data to the user with very small latency. The *GPS* function helps to store geographic

information about the current location of the driver and of the network nodes to display combined with the image data.

2.3 MQTT Message Processes

The MQTT protocol primarily operates based on the exchange of data in the form of messages. It uses publish and subscribe methods to exchange these messages, which are similar to the response and request methods of the HTTP protocol. In this section, a detailed description is presented of the process of exchanging messages, as shown in Fig. 3. The message flows are divided into three links: the first is between cam node and gateway; the second is between gateway and server; and the third is between server and user.

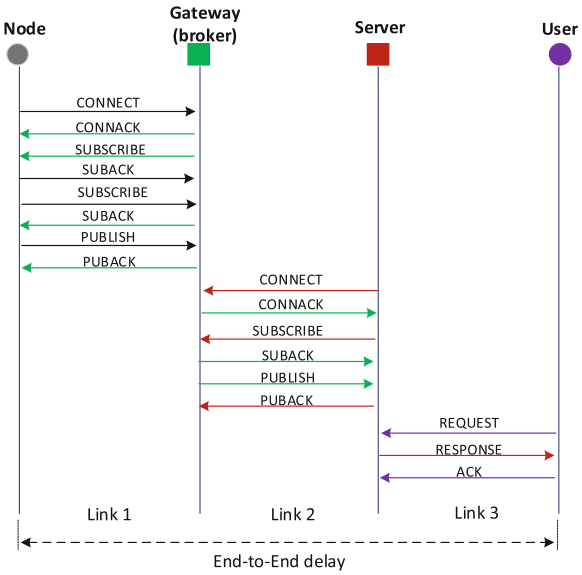

Fig. 3. Processes of message exchange in the proposed system.

When the cam node sends data to a subscribed channel, it can select one of three levels for quality of service (QoS) transmission: these levels are QoS0, QoS1 and QoS2. QoS0 means that the broker/client will deliver the message once, with no confirmation; QoS1 means that the broker/client will deliver the message at least once, with confirmation required; and QoS2 means the broker/client will deliver the message only once, using a four-step handshake [4]. The choice of QoS level will affect the performance of the system.

3 System Analysis

The performance of the system is analyzed based on metrics such as TCP end-to-end throughput, TCP end-to-end delay, arrival delay of messages (jitter), packet delivery ratio and packet drop rate.

3.1 Arrival Delay of Messages (Jiiter)

This section, we consider the arrival delay between messages generated by the cam nodes. Consider the case where the cam nodes, the gateway and the server are synchronized in time. The messages are generated using a generator with a constant rate and published to the MQTT broker (gateway). These messages arrive at the server with a varying delay depending on the network conditions. For two consecutive messages received by server, the arrival time of message n is denoted by t_n. The inter-message production period is denoted by T. With reference to [5], the inter-arrival jitter time J_n between message n and message $n - 1$ is given by:

$$J_n = t_n - t_{n-1} - T \tag{1}$$

Using the values of the inter-arrival timestamps in each message, the jitter value of two consecutive messages can be calculated. The proposed system is implemented in the 4G network, with a channel bandwidth of between 20 Mbps and 100 Mbps in the case of high mobility. The experimental results show that this jitter value is on the order of a few milliseconds and is therefore very close to zero, thus enabling high quality image streaming.

3.2 TCP End-to-End Delay

We define end-to-end delay as transmission delay denoting the time needed to transmit data from the cam node to the user. As shown in Fig. 3, assuming that data have been published on a given topic in the gateway:

$$t_{arr(i)\_topicID} < t_{req(i)\_topicID} < t_{arr(i+1)\_topicID} \tag{2}$$

where $t_{arr(i)\_topicID}$ is the time at which the i^{th} data is published to this topic ID and $t_{req(i)\_topicID}$ is the time of the request for the i^{th} data. This request should be made after the i^{th} data is present; if the request is made before the data is available, the system waits for a short period, and this increases the end-to-end delay of the system. This delay arises from several factors such as packets being dropped in the queue or network congestion.

The end-to-end delay from the cam node to the user can be calculated as:

$$\Theta_{end\text{-}end} = \sum_{i=1}^{N} \left(D_{trans(i)} + D_{prop(i)} + D_{proc(i)} + D_{queue(i)} + D_{TA(i)} \right) \tag{3}$$

where N is the number of links. There are three links in our system. D_{trans} is the transmission delay, and is proportional to the length of the message. The transmission delay is considered here in the cam node, the gateway and the server. D_{prop} is the propagation delay, and depends on the physical length of the link. D_{proc} is the processing delay, which is the time required for processing the message header. We assume that D_{proc} is small compared with other network delays and can therefore be neglected. D_{queue} is the queuing delay, which is the time a message waits in a queue before being executed. In this system, the message waits in both the queue of the gateway and the queue of the server. D_{TA} is the turn-around time delay, which denotes the delay in adapting to a request for data from a specific topic ID. The queuing delay and the turn-around time delay depend on the size of the queue length.

The equation for computing the end-to-end delay can be rewritten to follow the flow of the message as:

$$\begin{aligned}\Theta_{end\text{-}end} =& D^{N\text{-}G}_{CONNECT\text{-}TCP} + 2 \times D^{N\text{-}G}_{SUB\text{-}TCP} + D^{N\text{-}G}_{PUB\text{-}TCP} + D^{G\text{-}S}_{CONNECT\text{-}TCP} \\ &+ D^{G\text{-}S}_{SUB\text{-}TCP} + D^{G\text{-}S}_{PUB\text{-}TCP} + D^{S\text{-}U}_{REQ\text{-}RES} + D_{queue} + 2 \times D_{TA}\end{aligned} \tag{4}$$

where $D_{CONNECT\text{-}TCP}$, $D_{SUB\text{-}TCP}$ and $D_{PUB\text{-}TCP}$ form the round-trip TCP delay when the MQTT protocol is used. *N-G*, *G-S* and *S-U* are the connections between the cam node and gateway, gateway and server, and server and user respectively. *N-G* is the Wi-Fi connection, and *G-S* and *S-U* are 4G connections.$D^{S\text{-}U}_{REQ\text{-}RES}$ is the round-trip TCP delay when the HTTP protocol is used to transmit data to the user from the server.

Considering the round-trip time delay (RTT) of TCP, with re-transmission if packet loss takes place, we can obtain:

$$\Theta_{end\text{-}end} = 4 \times RTT^{NG}_{TCP} + 3 \times RTT^{GS}_{TCP} + 2 \times RTT^{SU}_{TCP} + D_{queue} + 2 \times D_{TA} \tag{5}$$

3.3 TCP Throughput, Packet Delivery, Packet Dropped Rate

The TCP throughput represents the total amount of transmitted data per second from sources to the server. The TCP throughput is normally calculated in bit/sec, and depends on various parameters such as the error rate of the channel, the bandwidth of the channel, packet size, queue size and the number of cam nodes. The evaluation of TCP throughput is given in Sect. 4. Theoretically, the TCP throughput is calculated as follows:

$$TCPthroughput = \frac{Total\ number\ of\ received\ bits(N)}{period\ of\ time\ (\tau)} \tag{6}$$

where N is all the bits received at the destination node over a period of time τ. In this system, the value of throughput is based on adjusting the quality of image capturing in the source node, as described above.

The packet delivery ratio (PDR) denotes the ratio of total packets received by the destinations to those generated by the sources. It can be expressed by the following formula:

$$PDR = \frac{Total\ number\ of\ received\ packets\ (N_r)}{Total\ number\ of\ generated\ packets\ (N_g)} \tag{7}$$

Its value describes the state of the channel in terms of the error rate, network congestion and queuing overflow. The analysis of packet delivery ratio is also carried out in the following section, which describes the simulation results. Packet drop rate is a parameter that denotes the number of packets lost during end-to-end transmission. An M/M/1 queue, the probability of packet drop, is calculated by:

$$P = (1 - \rho)\rho^k \tag{8}$$

where $\rho = \frac{\mu_c}{\mu_s}$ is the system factor.

4 Implementation and Performance Evaluation

4.1 Experimental Results

In this implementation of image streaming using the MQTT protocol, MQTT Version 3.1.1 [8] was used. MQTT is an open-source message broker service that uses the MQTT protocol to send and receive messages from MQTT clients.

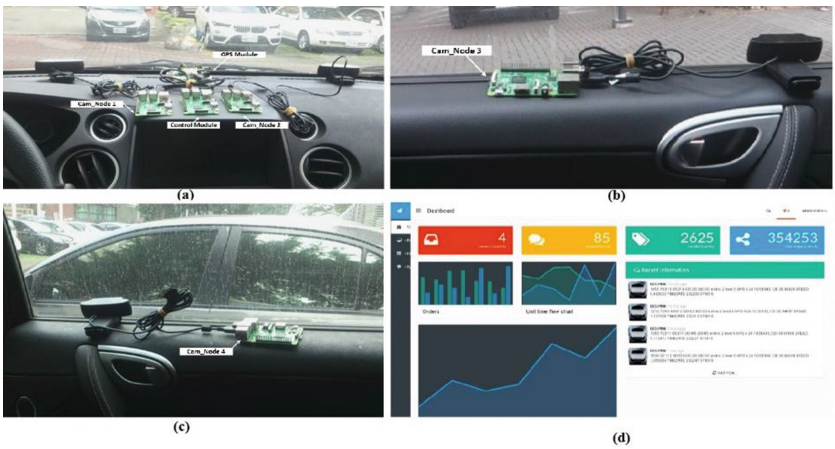

Fig. 4. Implementation of the proposed system: (a) cam nodes (Raspberry Pi 3) mounted at the front of the car; (b) a cam node mounted on the right side of the car; (c) cam node mounted on the left side of the car; (d) website interface.

Figure 4 illustrates the implementation of local nodes. Four Raspberry Pi 3 machines were mounted as cam nodes on a car. Each Raspberry Pi is connected to a camera to process the images captured by this camera. MQTT Mosquitto [8] was installed on each Raspberry Pi for image-streaming application. Figure 4(a) shows cam nodes 1 and 2, the GPS module and control module placed at the front of the car. Figure 4(b) shows cam node 3, placed at the right of the car; Fig. 4(c) shows cam node 4, placed at the left of the car; and Fig. 4(d) shows the website interface. The top-right corner of the web interface shows the storage for all the locations of and images from the vehicle; the middle of the web interface shows statistics on driving history; and the right-hand interface is the recent timeline history. Figure 5(a) shows the GPS route that the user has taken on Google maps. Figure 5(b) describes the normal view of the user web interface for two cam nodes placed at the front and back of the vehicle.

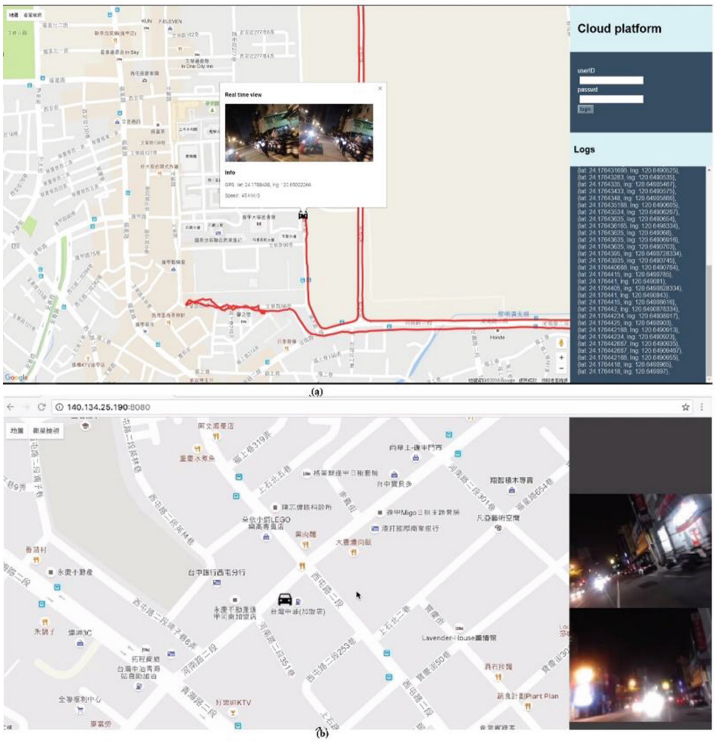

Fig. 5. (a) Tracking the location of the user on the road; (b) image streaming in the user's view.

4.2 Simulation Scenarios

To simulate the proposed architecture, we used the Network Simulation 2 (NS2.35) tool, and its configuration is shown in Table 1. The simulation scenario is described as follows: A local IoT network of cam nodes is connected to a 4G network for data

transmission. Cam nodes in the local network will send data to the gateway through a Wi-Fi connection. The connection from the gateway to the server is a 4G connection. The gateway will connect to a base station called the eNB node, and this eNB node will be connected to the server via a wired connection with a delay of 2 ms. The simulation of our 4G network is based on the study in [9].

Table 1. Simulation settings.

Parameter	Value
Network area	1500 × 1500 m
Simulation time	100 s
Error model	Uniform random distribution
Number of nodes	{1, 2, 3, 4, 5, 6}
Channel	802.11
Bandwidth	11 Mb
Data rate	11 Mbps
Traffic type	Constant bit rate (CBR)
Packet size	1500 bytes
Routing protocol	AODV
Transport protocol	TCP Reno
Bandwidth AP – eNB	20 Mbps
Bandwidth eNB – server	100 Mbps
Propagation delay eNB – server	2 ms

To analyze the performance of the network, trace files were generated after each simulation scenario in the NS2 simulator. A traffic generator was used with constant bit rate (CBR) and was attached to each TCP flow with sending rate corresponding to the experimental values as described in Sect. 4.1. The results of the simulation show that the obtained TCP throughput of our network is very close to the experimental results. Figure 6(a) shows the total throughput of the network as the number of nodes is increased.

In Fig. 6(b), (c), we change the length of the queue and change the number of nodes to evaluate the performance of the system. Figure 6(b) describes a comparison of the end-to-end delay of the normal network and the proposed network when the number of cam nodes is increased. The TCP end-to-end delay is calculated as discussed in Sect. 3.2. From the simulation results, we can see that the end-to-end delay values of the proposed network and the normal networks are approximately the same when the number of nodes is small; when the number of nodes is increased, the proposed network has a significantly reduced end-to-end delay than the normal network. Because the normal network will have a high packet drop rate when the channel conditions change, thus leading to more retransmissions and increasing the latency. Figure 6(c) describes the analysis of the packet drop rate. It can be seen that the proposed network keeps a smaller packet drop rate than the normal network. That is because with the proposed network, we can adjust the arrival rate, thus reducing the congestion at the queues and leading to lower packet drop rate.

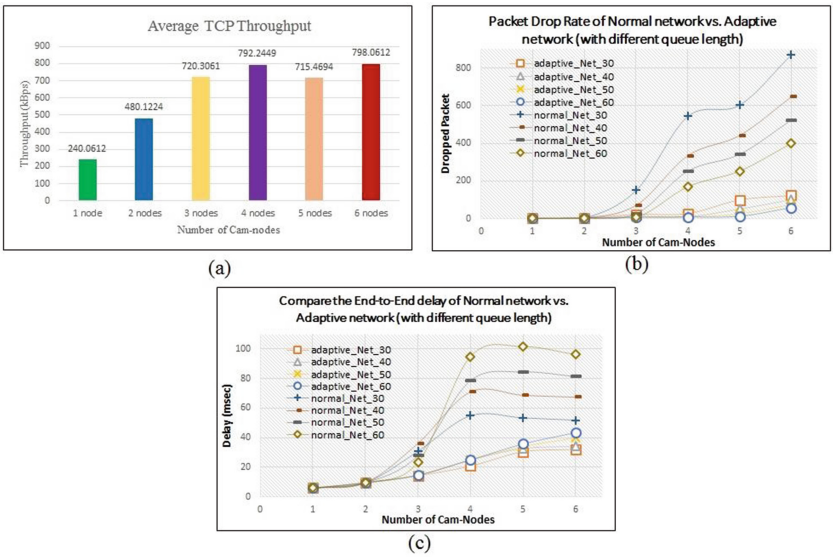

Fig. 6. (a) Network throughput for various numbers of cam nodes; comparison between the adaptive network and normal network for changes in queue length and number of cam nodes: (b) end-to-end delay; (c) packet drop rate.

5 Conclusion

The objective of this study is to introduce a system of smart monitoring cameras. Our system can automatically adjust the quality of streaming image capture to adapt to changes in the data rate of the channel due to the movement of vehicles. We propose dividing the uploading throughput into levels by which the quality of image capture can be controlled. An IoT network was set up in which each normal node was a Raspberry Pi, and these nodes were connected to a gateway to transfer data to a cloud streaming server using the MQTT protocol. Our implementation shows that this system consistently gives accurate information about the route through the captured images. This information can help enable driving safety applications such as risk warning and collision avoidance. The simulation results show that our adaptive control algorithm significantly reduced end-to-end delay by up to 65% when the number of nodes was increased and is thus very suitable for data streaming applications in intelligent transport systems. In the near future, we will focus on adding multi-hop functionality to the network architectures when considering the interaction between vehicles, improving the quality of images collected based on determining the optimal values of thresholds, and also extending the system with a greater number of nodes.

References

1. Wark, T., Corke, P., Karlsson, J., Sikka, P., Valencia, P.: Real-time image streaming over a low-bandwidth wireless camera network. In: 3rd International Conference on Intelligent Sensors, Sensor Networks and Information, pp. 113–118 (2007)
2. Bhatt, A., Patoliya, J.: Cost effective digitization of home appliances for home automation with low-power WiFi devices. In: International Conference on Advances in Electrical, Electronics, Information, Communication and Bio-Informatics, pp. 643–648 (2016)
3. Yaqub, M.A., Ahmed, S.H., Bouk, S.H., Kim, D.: FBR: fleet based video retrieval in 3G and 4G enabled vehicular ad hoc networks. In: IEEE ICC 2016 – Communication QoS, Reliability and Modeling Symposium (2016)
4. Tekin, Y., Sahingoz, O.K.: A publish/subscribe messaging system for wireless sensor networks. In: Sixth International Conference on Digital Information and Communication Technology and its Applications (DICTAP) (2016)
5. Luzuriaga, J.E., Cano, J.C., Calafate, C., Manzoni, P.: Handling mobility in IoT applications using the MQTT protocol. In: Internet of Things Technologies and Applications (2015)
6. Luzuriaga, J.E., Perez, M., Boronat, P., Cano, J.C., Calafate, C., Manzoni, P.: A comparative evolution of AMQP and MQTT protocols over unstable and mobile networks. In: 12th Annual IEEE Conference on Consumer Communications and Networking Conference (CCNC) (2015)
7. Govinda, K., Azad, A.P.: End-to-end service assurance in IoT MQTT-SN. In: 12th Annual IEEE Consumer Communications and Networking Conference (CCNC) (2015)
8. MQTT Mosquitto Version 3.1.1 Homepage. http://mosquitto.org/. Accessed 22 Dec 2016
9. Abed, G.A., Ismail, M., Jumari, K.: Traffic modeling of LTE mobile broadband network based on NS-2 simulator. In: 3rd International Conference on Computational Intelligence, Communication Systems and Networks (2011)