

## RESEARCH ARTICLE

# Fuzzy Congestion Control and Avoidance for CoAP in IoT Networks

THIEU NGA PHAM<sup>1</sup>, DANG HAI HOANG<sup>2</sup>, AND THI THUY DUONG LE<sup>1</sup> <sup>1</sup>Faculty of Information Technology, University of Civil Engineering, Hanoi 10000, Vietnam<sup>2</sup>Posts and Telecommunications Institute of Technology, Hanoi 10000, Vietnam

Corresponding author: Dang Hai Hoang (haihd@ptit.edu.vn)


**ABSTRACT** In Internet of Things (IoT) networks, congestion is growing with the increasing number of devices, and a large amount of collected data must be transferred. Congestion control is one of the most significant challenges for such networks. The Constrained Application Protocol (CoAP) has been adopted for the IoT to satisfy the demand for smart applications. However, CoAP uses a basic congestion control algorithm that operates only when congestion occurs. Thus, the basic CoAP and most similar loss-based congestion control schemes have remaining issues for burst data transfer in dynamic network environments. This paper proposes a novel rate-based congestion control scheme using fuzzy control for CoAP, called FuzzyCoAP. We use the round-trip time gradient and bottleneck bandwidth gradient as inputs for FuzzyCoAP to infer the degree of congestion. FuzzyCoAP uses this indicator to predict early congestion and adjusts the sending rate to avoid congestion. FuzzyCoAP uses the congestion degree to update the variable RTO for retransmissions. On the other hand, FuzzyCoAP dynamically checks for the available bandwidth to gain high performance for burst data transfer. Various simulation experiments have demonstrated the feasibility of the FuzzyCoAP in different traffic scenarios. We compared the proposed scheme with representative loss-based CoAP schemes, that is, the basic CoAP. The simulation results proved that FuzzyCoAP provides high performance in terms of delay, throughput, loss rate, and retransmissions compared with the basic CoAP.

**INDEX TERMS** Congestion control, rate control, fuzzy control, constrained application protocol, Internet of Things.

## I. INTRODUCTION

With the growth of Internet of Things (IoT) networks, various smart applications have been deployed to improve the quality of human life. Typical applications include event-monitoring networks in healthcare, agriculture, or environmental areas. An IoT network typically consists of a considerable number of IoT devices that collect data from the physical environment and send burst data to a server at a remote center.

The Internet Engineering Task Force (IETF) standardized the Constrained Application Protocol (CoAP) as a transport layer protocol for IoT networks [1]. CoAP is a lightweight protocol that works on top of an unreliable User Datagram Protocol (UDP) but provides a reliable connection-oriented data transfer mode such as the Transmission Control Protocol (TCP) [2], [3]. Similar to other transport protocols,

The associate editor coordinating the review of this manuscript and approving it for publication was Md. Arafatur Rahman .

CoAP uses congestion control (CC) to alleviate network congestion.

There has been debate regarding the deployment of TCP in IoT networks because TCP is the dominant transport layer for the Internet. TCP was designed with congestion control and reliable data delivery to efficiently transport bulk of data over the Internet. However, TCP faces significant challenges and has been neglected in IoT networks [4].

IoT networks have distinctive characteristics compared to traditional computer networks. IoT devices typically have limited resources and processing capabilities. IoT networks are often dynamic and have a high bit error rate. Many IoT applications rely not only on the occasional transmission of small data patterns but must also transfer a large amount of collected data to the server, resulting in enormous traffic on the network [5]. One example of such an application is the camera monitoring system [6]. To support the burst traffic of typical applications, the design of a suitable CC for CoAP

is undoubtedly necessary. However, this issue has not been resolved satisfactorily. The current standardized CoAP [1] uses basic congestion control that works only when congestion occurs. When a timeout occurs, CoAP assumes that the network is congested. In response, CoAP triggers the retransmission of the lost packets. This implies that the basic CoAP cannot detect early congestion. In addition, CoAP strictly limits the number of simultaneous outstanding interactions [1, p. 26]. Thus, the current CoAP does not support a burst traffic transfer. RFC 7252 [1] shows that further consideration of CoAP congestion control is expected in the future.

Congestion control (CC) is a broad research topic. This study focuses only on end-to-end CC solutions for the transport layer because CoAP operates on top of an unreliable transport protocol (i.e., UDP). Existing end-to-end CC schemes for transport protocols can be classified into *three* main groups: 1) loss-based CC; 2) delay-based CC; and 3) rate-based CC. In general, all CC schemes require feedback from the network or destination to the senders (in terms of timeout, packet loss, delay, or throughput) to control the load to the network (reduced window, load, or rate). Examples of existing CC schemes can be found in [7], [8].

Loss-based CC schemes use timeout or packet losses as an indication of network congestion to reduce the window. Typical examples are the schemes in [3], [9], and [10] for TCP, and [11], [12], [13], [14], [15], [16], [17] for CoAP. CoAP and most of its variants [11], [12], [13], [14], [15], [16], [17] use a loss-based CC approach based on a retransmission timeout (RTO) with a backoff strategy. It should be noted that CoAP uses a fixed window size called NSTART, whose default value is *one* [1]. In the case of packet loss, the CoAP sender attempts to retransmit the packet using a reduced retransmission rate (i.e., reduced load) in the backoff phase. Delay-based CC schemes utilize packet delay, round-trip time (RTT), or load to predict network congestion. The window or load is adjusted for each round-trip time. Examples are some schemes proposed for TCP in [18], [19], [20], [21], and [22]. Rate-based CC schemes leverage measurements of round-trip time (RTT), packet loss, or throughput to adjust the transmission rate, thus avoiding congestion. Examples include the TCP and TCP-friendliness schemes in [25], [26], and [27]. Recently, several rate-based schemes have been proposed for CoAP, such as those in [23], [24], [28], and [29].

Loss-based CC schemes are activated only when network congestion occurs. Their drawback is that they require an additional mechanism for early loss detection to avoid successive packet losses due to congestion. In the case of TCP schemes, Random Early Drop (RED) [10] is a good example of such a mechanism to support TCP. However, RED is not an end-to-end CC approach. Such schemes require support from the network routers. Delay-based and rate-based CC schemes are similar in that they both use RTT measures to predict congestion. However, they differ with respect to the control object. Delay-based CC schemes control the window

size, whereas rate-based CC schemes control the transmission rate. Delay-based CC schemes rely on window adjustment mechanisms (e.g., [18] and [42]). In contrast, rate-based CC schemes are based on rate adjustment (e.g., [23], [28], and [29]). Rate-based CC schemes are the most promising approaches because they operate in regions with low delays and high throughput. This point will be described in the next section. These schemes adjust the sending rate before congestion occurs. Thus, we can call these congestion avoidance schemes.

In this study, we focus on rate-based CC approaches. As the sending rate increased, the load on the network increased and the round-trip delay increased rapidly. Most rate-based CC schemes measure round-trip delays and use this information to set timeout timers and adjust the sending rate (or load). There are *three* crucial issues related to congestion control: 1) decisions on the increment/decrement of the sending rate (or load), 2) the number of changes, and 3) frequency of changes.

The decision to increase or decrease is clear for loss-based CC schemes, as they should decrease in case of a packet loss and increase otherwise. This decision is more complicated for rate-based CC schemes. It is difficult to determine the bounds of the RTT, load, and rate to make decisions because of the variance in these parameters, high burst traffic, and dynamic network conditions. In addition, such parameters are often fuzzy (i.e., not sharp, uncertain, vague, and ambiguous). A fuzzy control system (FCS) is an excellent choice for solving such complex problems. An FCS converts crisp inputs into fuzzy sets, and then produces crisp outputs based on fuzzy rules and an inference engine [30].

Fuzzy control has been widely applied to network congestion control in the past (e.g., [31], [32], [33], [34], [35], [36], [37], [38], [39], and [40]), either for TCP or other protocols such as routing. Such schemes use fuzzy control to infer the packet drop probability or congested routes based on measurements of buffer sizes in the network nodes. For CoAP, a fuzzy logic system was proposed in [41] for the backoff strategy and RTO adjustment. However, this system addresses only the issues of RTO estimation.

To the best of our knowledge, no study has applied fuzzy control to the problem of rate control to avoid congestion in CoAP. In this paper, we propose a fuzzy control system (FCS) for CoAP CC in IoT networks. Our scheme uses *two* crisp inputs: the *RT-gradient* and *bottleneck bandwidth gradient*, which are computed based on the ACK messages (acknowledgment) from the server. Based on these inputs, the FCS predicts the degree of network congestion and uses this indicator to control the sending rate and to avoid congestion. The proposed scheme can be implemented for burst data transfers in IoT networks.

The remainder of this paper is organized as follows. Section II presents the background and related work. In Section III, we propose an FCS for a CoAP. The proposed FuzzyCoAP is presented in Section IV. In Section V, we evaluate the performance of the proposed system and

discuss the related issues. Finally, Section VI concludes the paper.

## II. BACKGROUND AND RELATED WORK

### A. CONGESTION CONTROL AND AVOIDANCE

Network congestion is a state of degraded performance due to the saturation of network resources (link bandwidth, buffers, and processing capabilities). Adapted from [18] and [42], Figure 1 presents a formal relationship between the sending rate, RTT, throughput, and load. The network state was identified by three regions: app-limited, bandwidth-limited, and buffer-limited. The sending rate is the objective of any CC scheme because it must be increased or decreased according to the congestion state to adjust the load on the network. The knee is the point at which the sending rate saturates the bottleneck link. A connection (i.e., a path from a client to the server) has exactly one bottleneck, that is, the slowest link in the path. The bottleneck link is important because it determines the maximum sending rate, and it is where persistent queues form [42]. The load represents the inflight packets (packets sent but not yet acknowledged). A cliff is the point at which congestion occurs [18].

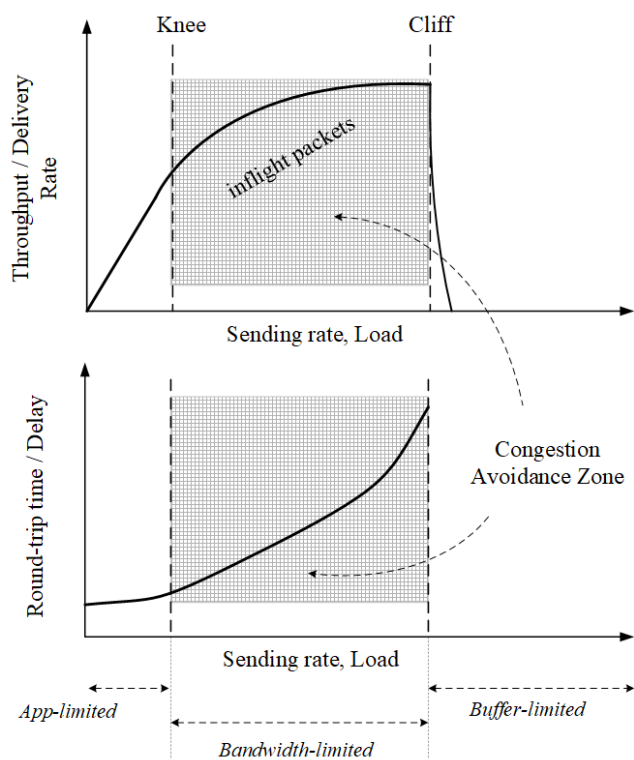


FIGURE 1. Overview of parameters for congestion control.

The app-limited region is characterized by low sending rates (e.g., there are insufficient packets to be sent). In this region, the throughput increases linearly with the load and the RTT increases weakly with the load. As the load reaches the knee point, the network state is in the second region (bandwidth-limited). In the second region, the throughput

increases slowly, whereas the RTT increases significantly because some queues build up at the bottleneck link. In the third region (buffer-limited), packets are dropped because of buffer overflows. The network was said to be congested. The RTT increased drastically, and the throughput decreased rapidly after the cliff point. Loss-based CC schemes operate at the right edge of the cliff point, whereas delay-based and rate-based CC schemes operate in a bandwidth-limited region. The knee is the theoretical optimal point, but we could not determine it in practice because of dynamic network conditions [42]. Therefore, our proposed scheme attempts to maintain the network state in the zone between the knee and cliff (shaded zone in the figure).

### B. COAP CONGESTION CONTROL

CoAP [1] is a lightweight web transfer protocol developed for constrained devices and networks. CoAP was designed to exchange messages over datagram-oriented transport (i.e., UDP) in the client/server model. CoAP defines *four* types of messages: confirmable (CON), non-confirmable (NON), acknowledgment (ACK), and reset (RST). CON messages must be acknowledged by the receiver (server). The CON and ACK messages ensured a reliable data transfer. If a CON message is not acknowledged within a fixed predefined retransmission timeout (RTO), the CoAP client initiates its retransmission. The initial RTO was fixed between *two* and *three* seconds as indicated in [1].

The basic congestion control of CoAP is loss-based. If a CoAP client detects packet losses, it is assumed that congestion has occurred. The CoAP reduces its retransmission rate using a binary exponential backoff (BEB) (i.e., the RTO is doubled for each retransmission). *Four* retransmissions are allowed. Subsequently, the transaction is considered failed. CoAP restricts the number of concurrent messages that can be sent without receiving an ACK (NSTART is *one* [1]).

CoCoA [11] is an enhancement to remedy the issues of the basic congestion control of CoAP. CoCoA uses *two* estimators to adjust the RTOs based on RTT measurements. In addition, CoCoA uses a variable backoff factor (VBF) instead of a fixed BEB to compute the RTOs. The RTOs were computed based on the formulas specified in RFC 6298 [43]. CoCoA+ [12], [13] is another enhancement that uses a more advanced CC. CoCoA+ introduces a smaller RTT variance factor to adjust the RTO values. Furthermore, CoCoA+ has a new aging policy for RTO calculation and resolves the problem of weak RTO estimation in CoCoA.

Most other proposed variants of CoAP have focused on the issue of RTO computation for CC in CoAP. The calculation of RTO is important because it determines the retransmission rate and efficiency of the protocol. Nevertheless, the RTO calculation depends on the estimation of variable RTTs. Many researchers have focused on the accuracy of RTO calculations using several enhanced algorithms for the measurement of RTT variances and adjustment of retransmission rates. The typical CoAP variants are as follows. The authors in [14] proposed a retransmission counter in the option field of a

CoAP header to update the RTO dynamically. Fast and slow RTO algorithms were proposed in [15]. Fast RTO was calculated for unambiguous RTT samples whereas slow RTO was performed for ambiguous RTT samples. The backoff logic follows *three* transitive states based on fast and slow RTO. The authors in [16] proposed a pCoCoA with a transmission count option to match the ACK message with the corresponding CON message, even for retransmission. pCoCoA introduces different methods to update the RTT and limits the minimum RTO to reduce spurious retransmissions. The AdCoCoA scheme [17] proposes a smooth estimation of the RTT variances to enhance the RTO computation algorithm in CoCoA+. CoCo-RED [44] proposed a combination of CC with revised Random Early Detection (RED) and a Fibonacci backoff algorithm. The importance of RTT variances has been recognized in most studies. Various methods for RTT variance estimation have been considered in recent works [15], [16], [17], [44], [46].

According to our survey, all the mentioned schemes are loss-based CC schemes because they focus only on RTO, retransmission, and backoff. The main differences between them are in the RTT estimation, RTO computation, and backoff policies. Their congestion control is activated only when packet loss has been detected, that is, when congestion occurs. These schemes attempt to mitigate congestion by reducing the retransmission rate (not the sending rate). These schemes do not allow for inflight packets, because they limit the number of outstanding packets. Therefore, these schemes are unsuitable for burst data transfer.

RTT-CoAP [24] is a delay-based CC scheme proposed for CoAP. This scheme monitors the growth of the RTT variances to infer the network state and determine the sending rate to prevent network congestion. RTT-CoAP defines *four* traffic regions: low congestion, normal operation, medium operation, and high variability. The sending rate was adjusted accordingly: fast/slow increased or fast/slow decreased depending on the network state in the regions. The scheme uses an additive increase/additive decrease (AIAD) policy. However, it is unclear how to select the factors for increments and decrements.

CoAP-R [28] is a rate-based CC scheme for CoAP. This scheme uses a tree-based routing deployment and supports burst traffic. This requires information on the node tree to discover bottleneck links. Congestion is detected using the present and past channel loads and the ratio between the current sending rate and allocated rate for a local node. This scheme uses a multiplicative increase/multiplicative decrease policy. However, this requires information on the buffer occupancy of the nodes. In addition, it is difficult to determine the allocation rates of nodes. BDP-CoAP [23] is another rate-based CC scheme for CoAP that leverages the BBR (bottleneck bandwidth and round-trip propagation time) protocol [42]. Based on bottleneck bandwidth estimation, BDP-CoAP determines the sending rate and keeps inflight packets bounded to the bandwidth-delay product. The pacing gain is either  $0.6$  or  $0.2$  when the retransmission frequency is

less or greater than  $20\%$ , respectively. CoAP-SC [45] can be called a rate-based scheme, because it regulates the sending rate based on flow control with error handling for streaming services. This requires adding sequence numbers to the option field headers of the packets. Based on the difference between the sending and receiving sequence numbers, the CoAP-SC can detect congestion. Recently, RCOAP [29] was proposed as a rate-based CC scheme for burst data transfer. It uses probe packets to infer the bottleneck bandwidth and regulates the rate according to an additive increase/multiplicative decrease policy. This allows for inflight packets and can distinguish congestion losses from wireless losses based on probe packets. A good survey for challenges and issues in the current CoAP can be found in [46].

As presented above, rate-based CC schemes offer several advantages: 1) they operate in a bandwidth-limited zone, 2) they can vary the transmission rate based on the network state and bottleneck bandwidth, and 3) they support inflight packets. Consequently, the proposed scheme used a rate-based approach.

### C. FUZZY CONTROL

Zadeh [47] introduced fuzzy theory in 1965. Since then, the fuzzy theory has been applied in many fields, with fuzzy control being an important application. Fuzzy control is a method based on fuzzy theory, which is used to control dynamic systems with ambiguous inputs. A fuzzy control system (FCS) has the advantage of controlling highly nonlinear, complex systems. Detailed information on the concept of FCS can be found in [47] and [50] for fuzzy sets, [48], [49] for fuzzy controllers, [51], [52] for membership grades and membership functions, and [30], [49], [52] for fuzzy relations, linguistic variables, fuzzy rules, fuzzification processes, fuzzy inference engines, and defuzzification processes.

Fuzzy control has been proven successful in congestion control, as shown in [31], [32], [33], [34], [35], [36], [37], [38], [39], [40], and [41]. In fact, it is difficult to develop a precise analytical model for congestion control owing its complexity. The network conditions are dynamic because of the burst nature of data transfer and the fluctuation of the parameters involved. In addition, the uncertainty in determining network parameters makes it challenging to obtain a realistic and cost-effective solution for congestion control. Thus, fuzzy control is a feasible approach to the congestion control problem.

In [31], the author demonstrated the difficulty of estimating the flow parameters in practice. A fuzzy controller was proposed to predict the time series parameters to adjust the flow rate. The authors in [32] proposed a Fuzzy-RED based on buffer length variations to predict congestion in TCP traffic [53]. The fuzzy CC scheme in [33] used buffer occupancy, traffic rate, and number of participating nodes to determine the congestion level in wireless sensor networks (WSN). The authors in [34] proposed combining RED with a fuzzy controller to adjust the sending rate for each node in a WSN using queue length differences and the variation of errors at

each node. An FCS was proposed in [35] using the number of contending nodes, buffer occupancy percentage of next-hop sensor nodes, and traffic load for video streams over a WSN. The authors in [36] proposed an FCS for routers using queue lengths and window size based on active queue management (AQM). Several studies have proposed the application of fuzzy control for route selection [37], [38], [39]. In [37], an FCS was proposed for route selection using an AQM in a WSN. This scheme isolates malfunctioning nodes to reduce the route congestion. In [38], a fuzzy priority-based RED mechanism was proposed to estimate the node congestion level based on the drop probability and find a less congested route. The fuzzy CC scheme in [39] uses node buffer occupancy, priority, and packet arrival rates to estimate congestion levels and regulate the sending rate. In [40], an FCS for rate adjustment was proposed, which leverages the queuing model in the node buffer to estimate congestion. The authors used buffer queue length and the ratio of RTT/RTO to estimate the network state and adjust the sending rate. In [41], an FCS was proposed for CoAP. However, this FCS is only for RTO estimation and backoff to reduce the number of retransmissions and not for sending rate control. Because this scheme is based on timeouts, it is a loss-based CC scheme.

In summary, the complexity and difficulty of congestion control motivated the use of fuzzy control as an appropriate solution to address the dynamic nature of network conditions and variations in the network parameters.

For CoAP, no research has addressed the application of fuzzy control for rate adjustment to avoid congestion. Therefore, this study proposes a rate-based congestion control and avoidance scheme for CoAP based on a fuzzy control system. In Section III, we present the design of our fuzzy control system that computes the degree of congestion to reflect the congestion state. This indicator is used to adjust the RTO and sending rate to avoid congestion, which is described in Section IV.

### III. PROPOSED FUZZY CONTROL SYSTEM FOR COAP

#### A. OVERVIEW

An FCS consists of *four* basic components: fuzzification, fuzzy rule base, fuzzy inference engine, and defuzzification [49], [52]. Fuzzification transforms crisp input values into a set of fuzzy values with membership functions for all linguistic variables and assigns grades of input membership. The fuzzy rule base presents a set of IF-THEN rules that define the input-output mapping based on membership functions. A fuzzy inference engine is used to infer the output based on the defined fuzzy rules and fuzzy implication operators. Defuzzification generates crisp outputs from the output fuzzy set provided by the fuzzy inference engine.

Depending on the number of inputs and outputs, an FCS system can be a single-input-single-output (SISO) system, a multiple-inputs-single-output (MISO) system, or a multiple-inputs-multiple-outputs (MIMO) system. Owing to the complexity of the congestion control, multiple inputs are required.

However, to reduce the complexity of CoAP CC, we use *two* inputs. The inputs of our FCS were the *RT-gradient* and *BG-gradient*. The *RT-gradient* indicates an increase in round-trip time. The *BG-gradient* is defined as the ratio of the delivery rate to the bottleneck bandwidth. To reflect the congestion level, *one* output, namely, the congestion degree (*C-degree*), is sufficient. Therefore, the proposed FCS is a MISO system.

The system model of the proposed FCS is shown in Fig. 2 with *four* components: *fuzzification*, *rule base*, *fuzzy inference engine*, and *defuzzification*.  $\mu(x)$ ,  $\mu(y)$ , and  $\mu(z)$  are the membership functions of inputs and outputs, respectively.  $Z_c$  is the crisp output (*C-degree*) of the FCS.

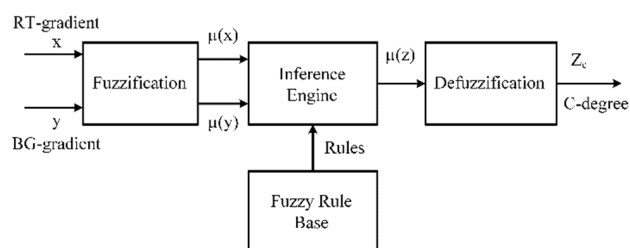


FIGURE 2. The system model of the proposed fuzzy control system.

The design of the inputs and outputs of the FCS is particularly important and is one of the main contributions of this study. In the following subsections, we present the inputs, outputs, and detailed components of our proposed FCS. Table 1 lists the notations used in this study.

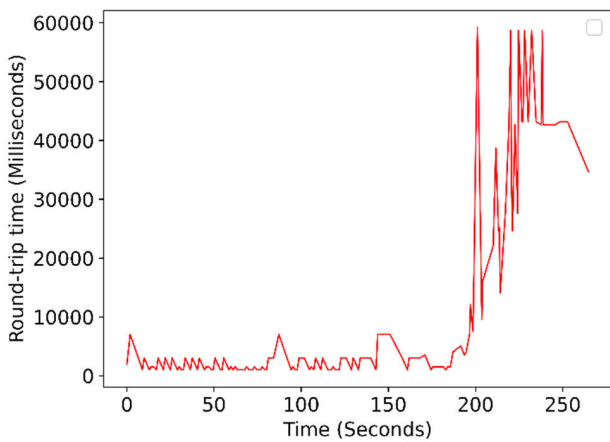
TABLE 1. Notations.

Notation	Meaning
$BG(k)$	Bottleneck bandwidth gradient at step $k$
$bW_{max}$	Maximum bottleneck bandwidth
$bW(k)$	Measured bottleneck bandwidth at step $k$
$dR(k)$	Instantaneous delivery rate at step $k$
$De(k-1, k)$	Amount of data delivered during $(k-1, k)$
$M$	Fuzzy subset $M$
$\mu(x), \mu(y), \mu(z)$	Membership function of input $x$ , input $y$ , output $z$
$nACK$	Number of received packets during startup
$RT(k)$	Round-trip time gradient at step $k$
$RTT_s(k)$	Estimated round-tip time at step $k$
$RTT_{min}(k)$	Minimum round-tip time at step $k$
$RTT_{max}(k)$	Maximum round-tip time at step $k$
$RTT_m(k)$	Measured round-tip time at step $k$
$RTO(k)$	Retransmission timeout value at step $k$
$S_{rate}(k)$	Sending rate at step $k$
$T(k-1, k)$	Sampling period between step $k-1$ and step $k$
$U$	Universe of discourse
$Z_c$ ( <i>C-Degree</i> )	Crisp output of the fuzzy control system

**B. RT-GRADIENT**

End-to-end CC schemes consider round-trip time (RTT) as a key parameter for CC and use it in diverse ways. Loss-based schemes use RTT for RTO estimation and RTO adjustment to limit the retransmission rate (e.g., [11], [12], [19], and [41]). Delay-based and rate-based schemes use RTT measurements to infer the possible congestion (e.g., [18], [23], [24], and [28]). It is well-known that RTT variances can be used as a good indication of the network state. Most solutions leverage TCP rules [19], [43] to estimate RTT samples using *three* variables: measured RTT, estimated RTT, and RTT variation. A smoothing factor was used to remove noise in the RTT measurements. The minimum of all measured RTTs (or all observed RTTs in a predefined time window) can be used to determine the network state [42], [23]. Recent schemes proposed for CoAP [23], [24] use the RTT variation variable to address the growth of RTT variance and define the regions of the network states. However, it has been recognized that the difference between RTT variations is meaningful [18]. Thus, the relative growth of RTT variations must be considered.

We conducted several simulation experiments using NS-3 [54] to explore changes in round-trip time (RTT) during congestion. In the experiments, *ten* CoAP clients sent back-to-back packets to a remote server to build a congestion situation. The common link bandwidth was 250 Kbps with a link delay of 50 ms. The simulation duration was 300 s. Fig. 3 shows the growth of the average RTT when congestion occurred.



**FIGURE 3.** Average RTT of CoAP in congestion.

As shown in Fig. 3, congestion started at 200 s with a significant growth in the RTT. Various RTT peaks were observed in the time interval between 90 and 200 s. The RTT peaks may indicate congestion threats. However, single values of the RTT or RTT variances are insufficient for predicting congestion.

In this study, we propose the *RT-gradient* to indicate the difference and relative growth of RTT variances. The *RT-gradient* is defined as the ratio between the relative and

absolute difference in the RTT variances, as follows:

$$RT(k) = \frac{RTT_s(k) - RTT_{min}(k)}{RTT_{max}(k) - RTT_{min}(k)} \quad (1)$$

$RTT_s(k)$ ,  $RTT_{max}(k)$ , and  $RTT_{min}(k)$  were determined as follows:

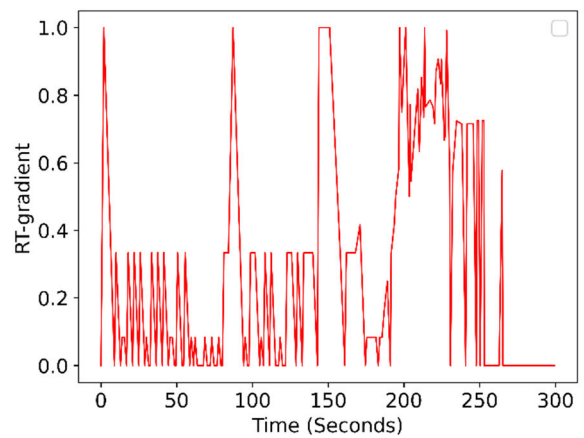
$$RTT_s(k) = (1 - \alpha) \times RTT_s(k - 1) + \alpha \times RTT_m(k) \quad (2)$$

$$RTT_{min}(k) = \min(RTT_{min}(k - 1), RTT_m(k)) \quad (3)$$

$$RTT_{max}(k) = \max(RTT_{max}(k - 1), RTT_m(k)) \quad (4)$$

where  $k$  and  $k-1$  are the step  $k$  and  $k-1$ , respectively, of the sampling periods in the discrete time interval. The sampling period  $T(k-1, k)$  is the interval between *two* successive ACKs.  $RTT_s(k)$  are the estimated RTT for each time interval ( $k-1, k$ ).  $RTT_m(k)$  is the current measured RTT at step  $k$ .  $\alpha$  is a parameter, and we choose  $\alpha = 0.25$  for a smooth estimation of RTT according to [12], [43].  $RTT_{min}(k)$  and  $RTT_{max}(k)$  are the minimum and maximum RTT, respectively, in each interval ( $k-1, k$ ).  $RT(k)$  is the *RT-gradient* at step  $k$ . Based on Eq. (1), the value of  $RT(k)$  is in the range of  $(0, 1)$ .  $RT(k)$  is *zero* if  $RTT_s(k) = RTT_{min}(k)$ , and  $RT(k)$  is *one* if  $RTT_s(k) = RTT_{max}(k)$ .

Fig. 4 shows the evolution of the *RT-gradient* according to the variation of RTT, as shown in Fig. 3. The *RT-gradient* was varied between *zero* and *one*. *RT-gradient* values close to *one* indicate the likelihood of congestion. Therefore, the *RT-gradient* was a good input for our fuzzy control system.



**FIGURE 4.** Example of the RT-gradient.

**C. BOTTLENECK BANDWIDTH GRADIENT**

The fuzzy control system is more efficient when a second input is used. Intuitively, the traffic load can be selected. However, it is difficult to determine the maximum load as the threshold for congestion detection. Another possibility is the use of a bandwidth-delay product (*BDP*). The BDP of a connection is defined as the product of its round-trip delay and bottleneck bandwidth [42]. However, BDP is often small in IoT environments. In addition, BDP is directly derived

from RTT. Thus, BDP is not a good choice because both the FCS inputs have the same effect.

Therefore, we propose using *bottleneck bandwidth* for the second input of our FCS. Specifically, we propose a *bottleneck bandwidth gradient (BG-gradient)* instead of a single bottleneck bandwidth value. As indicated in [42], bottleneck bandwidth is one of the most important parameters for congestion control. The bottleneck bandwidth is defined as the slowest link (or bottleneck) in the end-to-end path of a connection. In practice, the bottleneck bandwidth continuously changes depending on the variable load and the number of active connections that share a common network path. According to [42], if the network path is a physical pipe, the maximum bottleneck bandwidth is its diameter. To avoid congestion, the transmission rate must be less than or equal to the maximum bottleneck bandwidth. From this evidence, we can deduce that the data delivery rate must be less than or equal to the maximum bottleneck bandwidth.

Let  $bW_{max}$  denote the maximum bottleneck bandwidth and  $dR(k)$  be the instantaneous delivery rate in the discrete-time interval  $T(k-1, k)$ . The *bottleneck bandwidth gradient (BG-gradient)* is defined as follows:

$$BG(k) = \frac{\min(bW_{max}, dR(k))}{bW_{max}} \quad (5)$$

The instantaneous delivery rate  $dR(k)$  can be computed as follows:

$$dR(k) = \frac{De(k-1, k)}{T(k-1, k)} \quad (6)$$

where  $De(k-1, k)$  is the amount of data delivered during time interval  $T(k-1, k)$ . The sampling interval  $T(k-1, k)$  for  $dR(k)$  can be selected as the time interval between *two* successive ACKs.

The value of  $bW_{max}$  can be determined at the startup stage of a connection as follows. The CoAP client continuously sends back-to-back packets to the server during the startup stage. The server recursively computes the minimal delivery time ( $T_{min}$ ) between two successive received packets and sends this value back to the client using piggyback ACK responses. The client computes  $bW_{max}$  as follows:

$$bW_{max} = \max(bW(k), \frac{s}{T_{min}}) \quad (7)$$

where  $s$  is the ratio between the probe packet size and data packet size.  $bW(k)$  is the estimated bottleneck bandwidth in step  $k$ . Our scheme uses probe packets to estimate the bottleneck bandwidth. In our implementation, the size of a probe packet was 49 bytes, and the size of each data packet was 106 bytes. At the end of the startup stage, the client obtains the final values for  $bW_{max}$ .

We used the same simulation experiments as those shown in Subsection III.B to explore the evolution of the *BG-gradient* in congestion. Using the experiments, we recognized that the maximum  $bW_{max}$  of a connection can be accurately estimated within an interval of *six* to *ten* RTTs under the fixed network conditions (fixed route, unchanged

link bandwidth, and number of nodes on the path). The instantaneous  $bW(k)$  fluctuates depending on the number and load of connections sharing the network path. The maximum  $bW_{max}$  is computed at the end of the startup stage and remains unchanged. An analogy was given in [42], where the network path was a physical pipe and  $bW_{max}$  was its diameter. The instantaneous  $bW(k)$  of a connection is part of the maximum  $bW_{max}$ . The duration of the startup stage can be between *six* and *ten* RTT cycles, as indicated in the next section. Based on Eq. (5), the value of  $BG(k)$  is in the range (0, 1).  $BG(k)$  is *zero* if  $dR(k) = 0$ , and  $BG(k)$  is *one* if  $dR(k) = bW_{max}$ .

Fig. 5 shows the evolution of the *BG-gradient* according to the variation in the data delivery rate. As shown in Fig. 5, the *BG-gradient* varies between *zero* and *one*. *BG-gradient* values close to *one* indicate the possibility of congestion. If the network is less congested, then the *BG-gradient* values are close to *zero*. Thus, the *BG-gradient* was selected as the second input for our fuzzy control system.

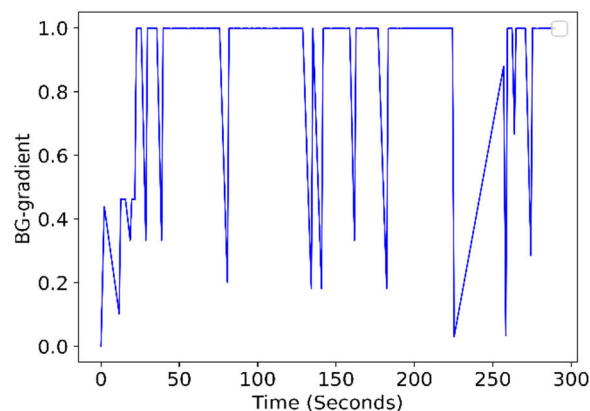


FIGURE 5. Example of the BG-gradient.

#### D. CONGESTION DEGREE

We define the congestion degree (*C-degree*) as the output of the fuzzy control system, which ranges from *minus one* to *plus one* ( $-1$  to  $1$ ). The *C-degree* can be used to interpret the network state as follows:

If the *C-degree* value is greater than *zero*, we assume that the network is in a non-congested state. The closer the *C-degree* is to  $1$ , the less congested is the network. If  $C-degree = 1$ , it can be assumed that there is no congestion in the network. The operating point corresponds to the left knee point, as shown in Fig. 1 (app-limited region). The sending rate can be aggressively increased.

By contrast, if the *C-degree* value is negative, the network is assumed to be congested. The sending rate must be reduced to avoid congestion. The decrease in the sending rate corresponds to a negative value of the *C-degree*. The closer the *C-degree* is to  $-1$ , the more congested is the network. The point with a  $C-degree = -1$  is corresponding to the cliff point (see Fig. 1). If the *C-degree* value is *zero*, then the sending rate must remain unchanged.

That is, the *C-degree* provides the direction and magnitude of the sending rate adjustment. If the load is low, the congestion possibility is low and the *C-degree* is positive. If the load is high, the congestion possibility is high, and the *C-degree* is negative. Therefore, by computing the *C-degree*, we can determine the direction and the amount of increases/decreases in the sending rate for a connection.

Until now, we have already answered the issues raised in Section I, that is, the decision on increase/decrease depends on the sign of the computed congestion degree (*C-degree*). This implies that the decision corresponds to the *RT-gradient* and *BG-gradient*, which represent the current network state. The amount of increase or decrease depended on the value of the *C-degree*. One round-trip time was selected as the sampling period for the *RT-gradient* and *BG-gradient*. The sending rate is adjusted for each RTT interval to avoid congestion.

**E. DESIGN OF THE FUZZY CONTROL SYSTEM**

As mentioned in the previous subsection, the fuzzy control system comprises *four* key modules: fuzzification, fuzzy rule base, inference engine, and defuzzification. The design of the FCS followed the standard steps indicated in [49], [52], and [30].

**1) FUZZIFICATION**

The function of this component is to represent the input and output data with linguistic variables and to derive their membership functions. This component converts crisp input data into fuzzy data. For each input, the number of linguistic variables should be odd, from *three* to *seven*, as recommended in [52]. Intuitively, the number of linguistic variables depends on the scale of each input and output variable. However, there is often no concrete scale for input. Furthermore, the complexity of the system may increase with the number of linguistic variables. In the case of CoAP, we consider *two* inputs and *three* linguistic variables for each input in this study.

The first input was the *RT-gradient*, which was normalized to the range [0, 1]. The second input was the *BG-gradient*, which was normalized to the range [0, 1]. We define the following linguistic variables for the inputs:

- *RT-gradient*: *small, medium, large*
- *BG-gradient*: *small, medium, large*

Linguistic variables are ‘variables whose values are not numbers but words or sentences in a natural or artificial language’ [52]. According to the definition of linguistic variables in [52, p.142], we can demonstrate the linguistic variables of *RT-gradient* input as follows. Let *X* denote a linguistic variable with a label called “*RT-gradient*”. The term set *T* for this input is as follows:

$$T(RT - gradient) = \{small, medium, large\} \tag{8}$$

Each term in this set is a fuzzy set. For our FCS, the universe of discourse *U* is in the interval [0,1], and a base

variable *u* ranges over *U*. For instance, let  $\mu_x(u)$  denotes the membership function of a linguistic variable *X* and let  $\tilde{M}(medium)$  denote a fuzzy subset of *U* that assigns a meaning to the term *medium*. We have:

$$\tilde{M}(medium) = \{(u, \mu_{medium}(u)|u \in [0, 1])\} \tag{9}$$

Similarly, we define the term set for the second input as follows:

$$T(BG - gradient) = \{small, medium, large\} \tag{10}$$

Several types of shapes can be deployed for membership functions including triangular, trapezoidal, and Gaussian. However, triangular and trapezoidal shapes are often used to represent the membership functions of inputs and outputs. In this study, we applied triangular and trapezoidal shapes to the membership functions as follows.

Let *a, b, c,* and *d* represent the *x* coordinates of the membership function  $\mu_M(x)$  in  $M = (a, b, c, d)$ , where *M* is a fuzzy set (*a* is the lower boundary, *d* is the upper boundary where the membership grade is *zero*, and *b* and *c* are in the middle, where membership grade is *one*). The trapezoidal shape membership function is defined as follows:

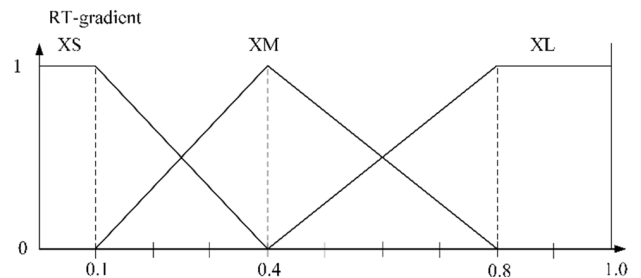
$$\mu_M(x) = \max \left\{ 0; \min \left\{ 1; \frac{x-a}{b-a}; \frac{d-x}{d-c} \right\} \right\} \tag{11}$$

If *b* coincides with *c*, let *m* represent the coincident coordinates of *b* and *c*. Then, we define a triangular shape membership function as follows:

$$\mu_M(x) = \max \left\{ 0; \min \left\{ 1; \frac{x-a}{m-a}; \frac{d-x}{d-m} \right\} \right\} \tag{12}$$

where  $M = (a, m, d)$  is a fuzzy set (*a* is the lower boundary, *d* is the upper boundary where the membership grade is *zero*, and *m* is the center where the membership grade is *one*).

Fig. 6 and Fig. 7 show the membership functions of the linguistic variables for the inputs to the FCS.



**FIGURE 6. Membership functions of RT-gradient.**

The *three* sets of linguistic variables for the input *RT-gradient* are: *small (XS), medium (XM), and large (XL)*. Similarly, there are *three* sets of linguistic variables for the input *BG-gradient*: *small (YS), medium (YM), and large (YL)*. Fuzzy sets may overlap each other in some parts. The size of the overlapping parts depends on the characteristics of the input characteristics.



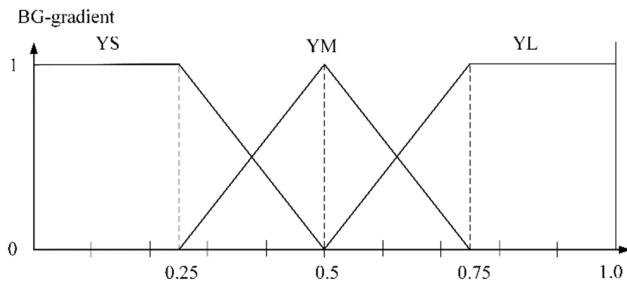


FIGURE 7. Membership functions of BG-gradient.

We propose membership functions for the output, that is, the *C-degree*, of our FCS, as shown in Figure 8. The output *C-degree* ranges from  $-1$  to  $1$ . These membership functions represent the arguments indicated in subsection III.D. Five linguistic variables were defined for the output as follows:

- *C-degree*: very low, low, medium, high, very high

The term set  $T$  for this output will be the following:

$$T(C - degree) = \{verylow, low, medium, high, veryhigh\}$$

Accordingly, linguistic variables for the output were divided into five sets: very low (ZVL), low (ZL), medium (ZM), high (ZH), very high (ZVH).

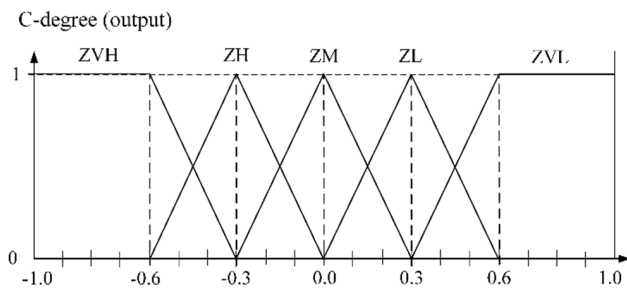


FIGURE 8. Membership functions of FCS output (C-degree).

## 2) FUZZY RULE BASE

This component defines the rules for determining the output linguistic variables based on the input linguistic variables. We propose the set of rules in Table 2.

A rule base is a set of IF-THEN rules, each of which is described by a fuzzy set, and fuzzy implication operators such as AND and OR. In this study, the FCS had two inputs, each of which had three linguistic variables. Our FCS uses the AND operators. Thus, we obtained  $3 \times 3 = 9$  fuzzy rules.

For instance, a rule can be described as follows:

**Rule 3:** IF the RT-gradient is “Small” AND the BG-gradient is “Large”, THEN the congestion degree is “Low”.

## 3) FUZZY INFERENCE ENGINE

The fuzzy inference engine determines the fuzzy output set. This process consists of two steps:

TABLE 2. Fuzzy Rules.

Rules	RT-gradient	BG-gradient	Congestion Degree
1	Small (XS)	Small (YS)	Very Low (ZVL)
2	Small (XS)	Medium (YM)	Very Low (ZVL)
3	Small (XS)	Large (YL)	Low (ZL)
4	Medium (XM)	Small (YS)	Low (ZL)
5	Medium (XM)	Medium (YM)	Medium (ZM)
6	Medium (XM)	Large (YL)	Medium (ZM)
7	Large (XL)	Small (YS)	High (ZH)
8	Large (XL)	Medium (YM)	High (ZH)
9	Large (XL)	Large (YL)	Very High (ZVH)

- 1) The membership grade of the result set was calculated. This involves the use of a rule base (Table 2).
- 2) Using inference methods for generating output fuzzy sets. There are several inference methods such as max-min, max-product, and singletons. Our FCS uses the max-min method for inference because of its fast operation.

## 4) DEFUZZIFICATION

This component is used to convert fuzzy output data into crisp output data.

For defuzzification, this study used the well-known center of gravity (CoG) approximation method [52]. The crisp values at the output can be approximated using the mean of the centers as follows:

$$C\_degree = Z_c = \frac{\sum_{i=1}^N Z_i \times \mu(Z_i)}{\sum_{i=1}^N \mu(Z_i)} \quad (13)$$

where  $Z_c$  is the crisp output (mean of center of gravity among  $N$  fuzzy sets),  $z_i$  is the center of gravity of each output set, and  $\mu(z_i)$  is the membership grade. The resulting  $Z_c$  is the *C-degree*, ranging between  $-1$  and  $1$ , which interprets the network congestion state, as indicated in Subsection III.D.

## IV. PROPOSED FuzzyCoAP

In this section, we present the specific mechanisms of our proposed protocol, FuzzyCoAP, using FCS designed in the previous section. FuzzyCoAP is an extended version of the basic CoAP over the UDP but with several modifications. FuzzyCoAP replaces the loss-based CC algorithm of basic CoAP with a rate-based CC algorithm that uses the designed FCS.

Fig. 9 provides an overview of the proposed FuzzyCoAP. The shaded blocks represent the three core states in the proposed scheme. The other blocks represent the core functions of the FuzzyCoAP.

The novelty of FuzzyCoAP is the design of an FCS for the CoAP CC that detects congestion and controls the sending rate to avoid congestion based on the measurements of

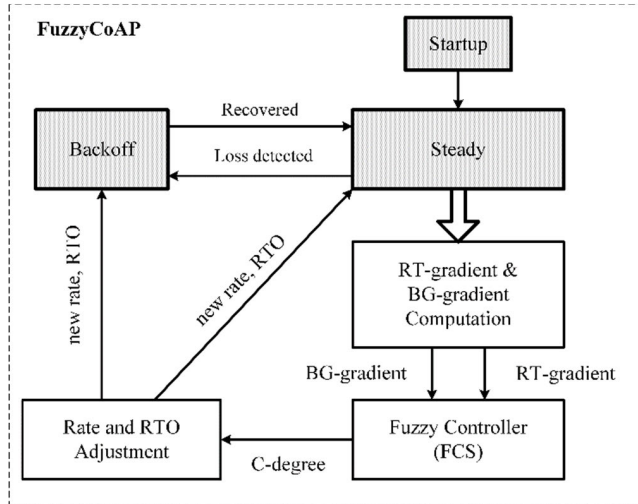


FIGURE 9. Overview of the FuzzyCoAP scheme.

RTTs and bottleneck bandwidth, namely, the *RT-gradient* and *BG-gradient*. Using dynamic rate control, FuzzyCoAP enables inflight packets and is efficient for burst data transfers. FuzzyCoAP controls the sending rate bounded by the maximal bottleneck bandwidth of the connection, and keeps the inflight packets limited to the bandwidth-delay product.

FuzzyCoAP primarily consists of algorithms at the sender to achieve effective congestion control. In FuzzyCoAP, each sender estimates  $bW_{max}$  and  $dR(k)$  during the startup stage of the connection. The startup stage is typically in the range of six to ten RTTs (we use ten RTTs in our implementation). The estimated  $dR(k)$  is used to determine the sending rate. In steady state, the sender continuously estimates RTT,  $bW(k)$ , and  $dR(k)$  whenever an ACK is received. The estimated RTT,  $bW(k)$ , and  $dR(k)$  values are used by the FCS to compute the *C-degree*, which is used to adjust the sending rate. In addition, FuzzyCoAP uses the *C-degree* to adjust the RTO values as follows:

$$RTT_s(k) = 0.75 \times RTT_s(k-1) + 0.25 \times RTT_m(k) \quad (14)$$

$$D\_RTT_s(k) = RTT_s(k) - RTT_s(k-1) \quad (15)$$

$$RTO(k) = RTT_s(k) + C\_degree(k) \times D\_RTT_s(k) \quad (16)$$

where  $D\_RTT_s(k)$  is the RTT variance computed at step  $k$ ,  $C\_degree(k)$  is the computed crisp output of the FCS at step  $k$ , and  $RTO(k)$  is the retransmission timeout value at step  $k$ . The estimation of  $RTT_s(k)$  follows Eq. (2) using  $\alpha = 0.25$  [43].

If the sender does not receive an ACK within the computed RTO, the sender attempts to retransmit the corresponding packet and checks for the possible transmission of other packets. Four retransmissions are allowed for each retransmitted packet. The RTO value is updated for each retransmission. FuzzyCoAP does not estimate the bottleneck bandwidth from retransmitted packets.

After four unsuccessful retransmissions, the timeout timer function marks the corresponding lost packet for loss

detection and returns it. During steady state, if packet loss is detected, the sender enters the backoff state. At this moment, several packets may be in flight, including recently transmitted and retransmitted packets. Packet loss indicates that congestion has occurred. To avoid further congestion, the sender should not send further packets. In the backoff state, the sender checks the ACK during each RTT interval. If an ACK is received, then the sender updates the *C-degree*. Subsequently, the sender returns to a steady state. Otherwise, the transaction is considered to have failed after the maximum transaction time, as defined in [1]. The proposed FuzzyCoAP is presented in detail in the following subsections.

### A. STARTUP

Algorithm 1 presents the pseudocode for the startup stage of FuzzyCoAP.

#### Algorithm 1 Startup

```

1: function Startup
2:   mState = Startup
3:   if (Cycle > maxCycle) then
4:     EndStartup()
5:   else
6:     packet ← SendNextPacket() //probe packets
7:     if (ACK == true) then
8:       nACK = nACK + 1
9:       Update1(RT(k), BG(k), dR(k))
10:      IPG = 1/dR(k)
11:    else
12:      if ((nACK == 0) && (Cycle = maxCycle)) then
13:        Restart()
14:      endif
15:    endif
16:    Cycle = Cycle + 1
17:    Schedule(IPG, Startup)
18:  endif
19: end function
20: function EndStartup()
21:   Srate = nACK / (now - t)
22:   (IPG = 1/Srate
23:   Update2(bWmax)
24:   mState = Steady
25:   Steady()
26: end function

```

At startup, a FuzzyCoAP client performs its probe phase to determine  $bW_{max}$  and the start sending rate. To this end, the client sends back-to-back packets to probe bottleneck bandwidth. The startup phase does not use the timeout timer for the probe packets, that is, the lost probe packets are not retransmitted. In each cycle, the client sends the next probe packet (line 6). Upon receiving a packet, the server immediately sends an ACK back to the client. When an ACK is received (line 7), the client updates the necessary parameters (line 9), as indicated in Eq. (1)–(4) and (5)–(7), using the function  $Update1(RT(k), BG(k), dR(k))$ , and increments the

number of received ACKs by *one* (line 8):

$$nACK = nACK + 1 \quad (17)$$

The pacing time *IPG* was updated accordingly (line 10). If no ACK is received, the client checks for connection timeout (line 12). If  $nACK \neq 0$  or  $Cycle < maxCycle$ , the client invokes the next startup cycle after *IPG seconds* using the schedule function (lines 16-17). In the last loop (i.e.,  $Cycle = maxCycle$ ), the variable *nACK* can be less than or equal to *maxCycle*. If  $nACK \neq 0$  (line 12), the client considers that at least *one* ACK has been received. The client then continues to schedule the next startup (lines 16-17). At the beginning of the next loop, the startup phase finishes (lines 4 and 20) because  $Cycle > maxCycle$ . If  $nACK = 0$  and  $Cycle = maxCycle$  (line 12), the client does not receive an ACK after all cycles. In this case, the client must restart the connection from the beginning (line 13). The initial *IPG* is unimportant because it changes when an ACK is received (line 10). In our implementation, we initialize the *IPG* by *one millisecond* for back-to-back packet transmission. The variable *t* is the start time of the startup state.

At the end of the startup stage (line 20), the client computes the sending rate  $S_{rate}$  (line 21) and *IPG* (line 22) and updates the final bottleneck bandwidth  $bW_{max}$  (line 23) using (7). The FuzzyCoAP client then enters a steady state (line 25).

The startup stage typically lasts for *six to ten* RTT cycles. This is because the connection and network require time to be stable. Through experiments, we found that an interval of *six* RTT cycles may be sufficient for the startup phase. Note that BBR [42] requires *eight* RTT cycles and BDP-CoAP [23] requires *ten* RTT cycles for the initial stage.

## B. STEADY

The pseudocode for the steady state of the FuzzyCoAP is presented in algorithm 2.

At the beginning of the steady stage, the FuzzyCoAP client computes the bandwidth-delay product *BDP* (line 4) based on the bottleneck bandwidth  $bW_{max}$  computed at the startup. The number of inflight packets will be checked. If this number is less than or equal to the *BDP* (line 5), the client sends the next data packet (line 6). The client sets the initial RTO for each new packet. If the timer expires, the timeout timer function (not presented in the algorithm) checks for a retransmission. If the client has not received an ACK from the server for the packet and the number of retransmission attempts is less than *four*, the timeout timer function retransmits the corresponding packet and sets an updated RTO. If *four* retransmissions occur unsuccessfully, the timeout function marks a flag *PacketLoss* for the lost packet.

After each estimated RTT (line 7), the client updates the *RT\_gradient* and *BG\_gradient* (line 8). It then calls the FCS function to compute  $C_{degree}$  (line 9). The client adjusts the sending rate (line 10) and the new inter-packet interval *IPG* (line 11). The  $RTT_s$  are smooth estimated RTT computed based on Eq. (14) for each RTT cycle. A smooth RTT estimation helps to avoid the impact of RTT fluctuations. The client

### Algorithm 2 Steady

---

```

1: function Steady
2:   mState = Steady
3:   t = now
4:    $BDP = bW_{max} \times RTT_{min}$ 
5:   if ( $inflight \leq BDP$ ) then
6:     packet  $\leftarrow$  SendNextPacket()
7:     if ( $t \geq last\_adjust + RTT_s$ ) then
8:       Update(RT(k), BG(k))
9:        $C_{degree} = FCS(RT(k), BG(k))$ 
10:       $S_{rate} = S_{rate} + C_{degree} \times (1/RTT_s)$ 
11:       $IPG = 1/S_{rate}$ 
12:      RTO_Update( $C_{degree}$ , RTO)
13:       $last\_adjust = t$ 
14:     endif
15:   else
16:     if ( $ACK == true$ ) then
17:       packet  $\leftarrow$  SendNextPacket()
18:     endif
19:   endif
20:   if (PacketLoss) then
21:     mState = Backoff()
22:     Backoff()
23:   else
24:     Schedule (IPG, Steady)
25:   endif
26: end function

```

---

updates the new RTO (Line 12) and the next adjustment time (line 13).

If the number of inflight packets exceeds *BDP* (line 15), the client must slow the transmission. The reason is that there are a lot of inflight packets that can cause congestion. To this end, the client sends only a new packet (line 17) when an ACK is received (line 16).

During the steady state, the client checks for packet loss (line 20). Flag *PacketLoss* causes a gap in the receiving ACKs for the transmitted packets. Packet loss indicates that congestion has occurred. Successive packet losses occurred in the case of heavy congestion. If packet loss is detected, the client exits the steady state and enters the backoff state (lines 21-22). Otherwise, the client schedules the next loop of the steady state after *IPG* seconds (line 24) by using the function *schedule (IPG, steady)*.

The FuzzyCoAP client adjusts the sending rate  $S_{rate}(k)$  based on  $C_{degree}(k)$  as follows (line 10):

- If  $1 \geq C_{degree}(k) > 0$ , the sending rate will be increased. The amount of the increase is the absolute value of the congestion degree  $|C_{degree}(k)|$ , which determines a fast or slow increase.

$$S_{rate}(k) = S_{rate}(k - 1) + |C_{degree}(k)|x1/RTT_s(k)$$

- If  $C_{degree}(k) = 0$ , the sending rate will be unchanged.

$$S_{rate}(k) = S_{rate}(k - 1)$$

- If  $-1 \leq C\_degree(k) < 0$ , the sending rate will be decreased. The amount of the decrease is the absolute value of the congestion degree  $|C\_degree(k)|$ , which determines a fast or slow decrease.

$$S_{rate}(k) = S_{rate}(k-1) - |C\_degree(k)| \times 1/RTT_s(k)$$

FuzzyCoAP differs from other CoAP schemes in that it allows for the sending of inflight packets. This means that FuzzyCoAP attempts to send the next packets while the retransmitted packets are in flight. FuzzyCoAP does not use *BW* estimates from the retransmissions. This is because parameters such as RTT and delay of retransmitted packets can lead to incorrect estimation of the bottleneck bandwidth, as indicated in [23], [42], and [43].

Packet duplication may occur because the retransmitted packet is sent to the receiver during flight. In addition, the received packets may be disordered at the receiver owing to the retransmission. These issues can be solved at the higher protocol layer, where the application can rearrange the sequences of the received packets and discard the duplicated packets. However, these issues are beyond the scope of this study.

### C. BACKOFF

Algorithm 3 shows the pseudocode for the backoff state. FuzzyCoAP enters this state in the case of congestion, i.e., if packet losses are detected.

---

#### Algorithm 3 Backoff

---

```

1: function Backoff
2:   mState = Backoff
3:   t = now
4:   if (ACK == true) then
5:     packet ← SendNextPacket()
6:     Update(RT(k), BG(k))
7:     C_degree = FCS(RT(k), BG(k))
8:     S_rate = S_rate + C_degree × (1/RTTs)
9:     IPG = 1/S_rate
10:    RTO_Update(C_degree, RTO)
11:    mState = Steady
12:    Steady()
13:    return
14:  else
15:    if (t > maxTransactionTime) then
16:      ConnectionFailed()
17:    else
18:      next_pacing = t + IPG
19:      Schedule (next_pacing, Backoff)
20:    endif
21:  endif
22: end function

```

---

The client must slow transmission to avoid further congestion. In this state, the client continues to check the acknowledgment from the server after a pacing time without sending

further packets. Packets that were already sent in the previous stage (including the retransmitted packets) may be in flight. If an ACK is received (line 4), the client assumes that congestion has been resolved. The client then sends a new packet (line 5), updates the parameters (line 6-10), and returns to steady state (line 11-13).

If no ACK is received (line 14), the client checks for maximum transmission time (defined in [1]). If no ACK is received after this value (line 15), the transaction is considered to have failed (line 16). The connection stopped. Otherwise, the client schedules the next backoff period after the next pacing time (line 18-19). The client may switch between steady state and backoff in the case of several packet losses due to heavy congestion. This change depends on the occasional ACK received for the transmitted or retransmitted packets.

### V. PERFORMANCE EVALUATION

In this section, we first present the simulation setup used for the performance evaluation of our FuzzyCoAP. Then, we present results for a variety of different simulation scenarios. We use the network simulator NS-3 [54] as the simulation platform to implement the basic CoAP and the FuzzyCoAP. NS-3 is a discrete event network simulator that supports fast and efficient prototyping.

The core implemented functions of the basic CoAP are startup, send, receive, retransmit, and timeout-timer. For FuzzyCoAP, we implemented the core functions at the client, including startup, steady, backoff, fuzzy control, send, receive, retransmit, and timeout-timer. On the server, both FuzzyCoAP and basic CoAP perform the main functions for receiving packets and sending ACKs. The method for creating CoAP headers was adapted from [55], [56]. In [55], the authors presented a partial implementation of the basic CoAP for multicast topology and DNS experiments in NS-3. This implementation was based on the code sketch for CoAP in the Spark Core system provided in [56]. Both authors in [55] and [56] used the same method to create basic CoAP header using standard parameters indicated in RFC 7252 [1]. The same standard parameters were used for the CoAP header.

Network simulator NS-3 provides numerous facilities to create various traffic scenarios similar to real networks. In our simulation, all clients were implemented at the wireless nodes in the Wi-Fi network. The clients are connected to the server through a Wi-Fi access point (AP node) and a bottleneck link (see Fig. 10). The clients and access point (AP) are implemented using the standard IEEE 802.11, standard internet stack, standard mobility model, and standard interface models provided by NS-3 [54].

In our simulation, the size of the CON packet was 106 bytes, including a fixed header of 4 bytes and payload of 102 bytes. The size of an ACK message was 49 bytes, including 4 bytes added in the option field. This field is used by the server to piggyback the delivery time to clients. The clients use delivery time to accurately measure the delay and bottleneck bandwidth. To investigate different congestion states and dynamic bottleneck bandwidth, we used various

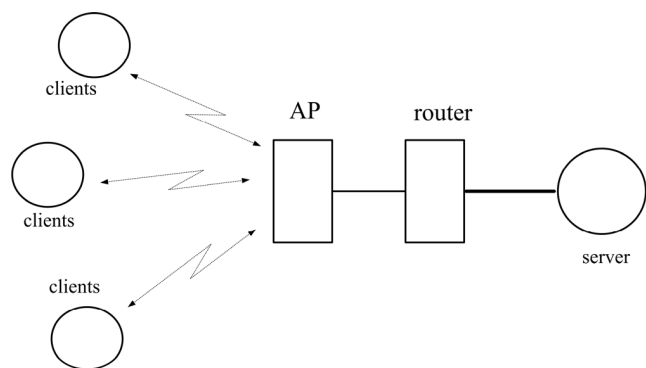


FIGURE 10. Simulation model for the homogenous traffic scenarios.

configuration parameters for the link bandwidth and delay between the access point and the server. The details are provided for specific traffic scenarios. Moreover, we assume that each client has sufficient collected data to demonstrate burst traffic transfer. All clients can generate *two* traffic types: continuous traffic and burst traffic. The simulation time was 300 s for all experiments.

Although many CoAP variants have been proposed, the basic CoAP has been standardized by the IETF [1], [57]. In Section II, we explain how the basic CoAP and most of its variants use loss-based approaches. In this paper, it will be meaningful to compare two types of CC schemes: loss-based and rate-based schemes. Therefore, we focused only on the performance comparison between our rate-based FuzzyCoAP and basic CoAP, because basic CoAP is representative of loss-based CoAP variants.

We evaluated FuzzyCoAP and the basic CoAP using the following indicative metrics:

- End-to-end delay: The delay between the sending time at the client and the receiving time at the server.
- Throughput: the delivery rate computed at the server for each time interval of *one* RTT.
- Retransmission: the number of retransmissions and the percentage of retransmissions.
- Packet duplication: the number of duplicated packets.
- Packet loss: the number and percentage of packet losses.

### A. SIMULATION SETUP

Two network topologies were used for simulation: star and dumbbell topologies. In the star topology, clients send data to a common server via the access point and router. The link between the router and the server was the bottleneck link for the experiments. In dumbbell topology, clients send data to different servers via the access point, network router, and edge router that connects to the remote servers. The link between the network router and edge router was considered the bottleneck link for the experiments.

We used *three* types of traffic scenarios for experiments: 1) homogenous traffic, 2) heterogenous traffic, and 3) dynamic traffic scenarios. Homogenous traffic scenarios are used for clients using the same CC scheme, either

FuzzyCoAP or basic CoAP. Heterogeneous traffic scenarios are used for a mixture of FuzzyCoAP, basic CoAP, and unresponsive clients. The unresponsive client used in the experiment was an unconfirmable (NON) CoAP client that sent messages without the need for acknowledgment. Dynamic traffic scenarios use TCP flow and unresponsive flow (UDP) as background traffic. Unresponsive flow dynamically changes the transmission rate. The purpose of these experiments was to demonstrate the behavior of the schemes under dynamic network conditions.

We used random start times for the flows in all experiments. The random seed was set between 0 ms and 200 ms to generate the start time for each flow. *Thirty* random test runs were conducted for each experiment. We used real-time tracing data for delay, retransmission, packet duplication, and throughput to produce the graphic data. The instantaneous throughput was calculated using the received packet and the time difference between sending and receiving packets at the server. The average values of the delay and throughput were calculated using confidence intervals with a normal distribution.

Although various link bandwidths and link delays can be selected for the simulation experiments, we used specific link bandwidths and link delays to demonstrate the bottleneck bandwidth for cases with no congestion, single congestion, and heavy congestion. The aim of these experiments was to compare the schemes under the same network conditions.

### B. HOMOGENOUS TRAFFIC SCENARIOS

#### 1) CHECKING THE FUZZY CONTROLLER

The network topology for homogenous traffic scenarios is illustrated in Fig. 10. To verify the fuzzy controller, *three* FuzzyCoAP flows were used. In these experiments, the link bandwidth between the AP and router was 250 Kbps with a link delay of 50 ms. The link between the router and server had a bandwidth of 45 Kbps and a link delay of 300 ms. The *three* FuzzyCoAP flows shared a common link bandwidth. The measured bottleneck bandwidth was approximately 2.6 Kbps, which was much smaller than the link bandwidth in these experiments owing to the competence of the flows.

Fig. 11 presents the behavior of the RT-gradient, BG-gradient, and computed congestion degree (C-degree) over time for FuzzyCoAP flows.

As shown in the figure, the clients started shortly after the initial stage and sent burst data. The load increased quickly (*BG-gradient* line). RTT increased accordingly (*RT-gradient* line). A short positive *C-degree* peak at 10 s indicates that the client tried to test the available bandwidth. Subsequently, the *C-degree* became negative. This implies that the client must send packets at a lower rate. In the interval between 10 s and 70 s, the client retained the sending rate to maintain its performance. The rate was updated at each RTT interval. The instantaneous packet delay was large, because several packets were on flight to the server. At 75 s, the *C-degree*

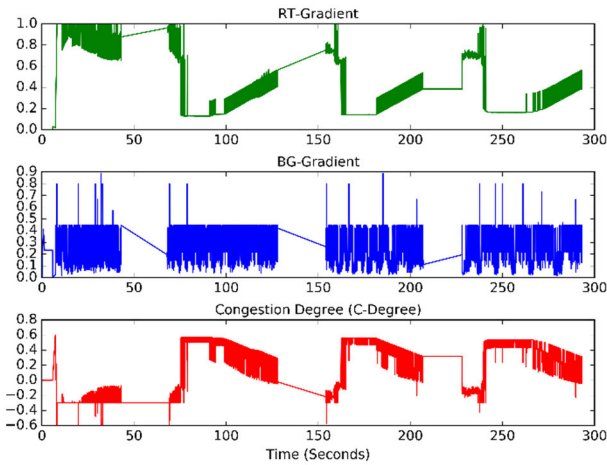


FIGURE 11. Behavior of RT-gradient, BG-gradient, and C-Degree.

became positive, indicating a low congestion threat. The client can then increase the sending rate. That is why the load (*BG-gradient*) was increased. In the time interval from 90 s to 160 s, the *C-degree* decreased owing to the increase in the *RT-gradient*. The client must reduce the sending rate to avoid congestion. The same control process was repeated for the subsequent intervals.

The variation in the *C-degree* depends on the available bandwidth of the bottleneck link. As presented, FuzzyCoAP can predict early congestion based on the computed *C-degree* and adjusts the sending rate accordingly.

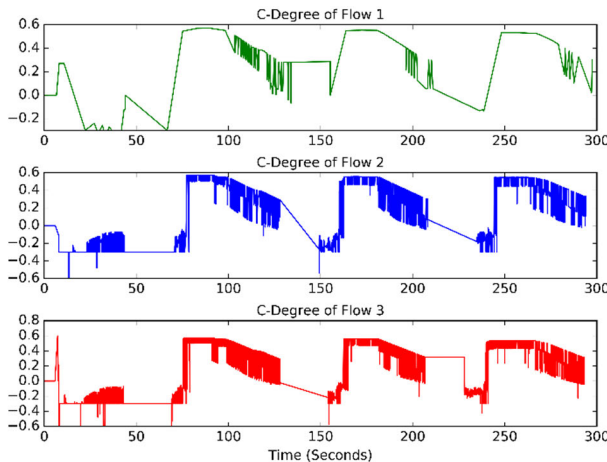


FIGURE 12. Change of C-Degrees of FuzzyCoAP flows.

In Fig. 12, we compare the *C-degrees* of the *three* FuzzyCoAP flows sharing a common bottleneck link. As shown in the figure, the *C-degrees* are increased or decreased according to the *RT-gradient* and *BG-gradient* inputs, as shown in Fig. 11. The changes in the *C-degrees* are different because the flows compete for the available bandwidth of the bottleneck link. However, the difference was small, indicating fairness between the flows.

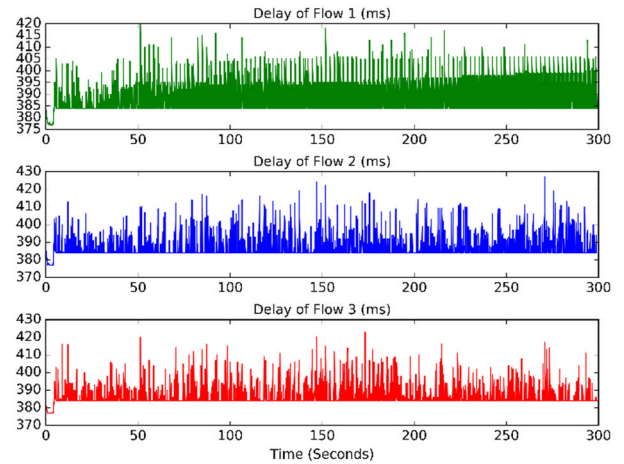


FIGURE 13. End-to-end delay of FuzzyCoAP flows.

## 2) PERFORMANCE OF FuzzyCoAP FLOWS

We used the same network topology as that shown in Fig. 10 for the performance evaluation of the *three* FuzzyCoAP flows. Fig.13 presents the end-to-end delay of the FuzzyCoAP flow. After the initial few seconds, the client attempted to measure bottleneck bandwidth. The measured value was 2.6 Kbps for each flow. Subsequently, clients started to send burst data. At the start of the connection, most of the packets exhibit a small delay. Packet delay increases with an increasing number of inflight packets. If the load is high, more packets are delayed. However, we can see that the average packet delay was less than 410 ms for all flows. This is because FuzzyCoAP limits the sending rate below the bottleneck bandwidth. According to the current measured available bandwidth, FuzzyCoAP flows attempt to maintain the sending rate smaller than the bottleneck bandwidth to avoid congestion. Thus, FuzzyCoAP can limit the maximum delay in flow.

There were some spikes in the delay. This is because of the competing packets of flows. As the number of inflight packets increases, more spikes are observed. However, the values of the spikes were less than 420 ms for all flows. This implies that the instantaneous sending rate may temporarily be larger than the bottleneck bandwidth without causing congestion, as explained in Section III.

Fig. 14 shows the throughput of the *three* FuzzyCoAP flows in the homogeneous experimental set.

Although the bottleneck bandwidth of the shared link was 2.6 Kbps, the throughput of each flow reached an average of 8 Kbps. This means that FuzzyCoAP can achieve a high performance. The throughput may vary at the beginning of the flow, but it becomes more stable in the long term.

In Table 3, we present the comparative performance of the *three* FuzzyCoAP flows using various metrics. In this set of experiments, no packet loss was observed. There was only one retransmission for each flow owing to timeout. However, this retransmitted packet is duplicated. This is because

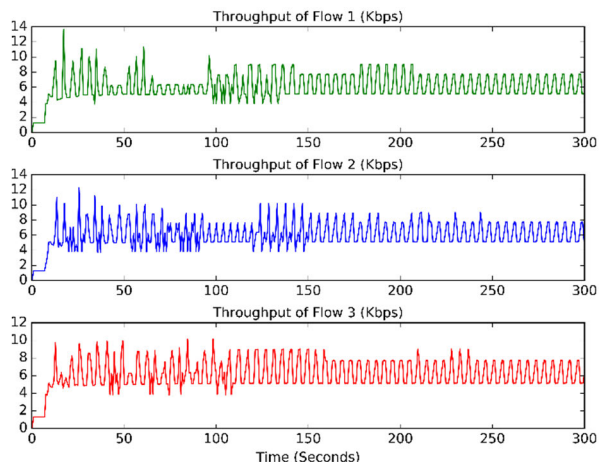


FIGURE 14. Throughput of FuzzyCoAP flows.

FuzzyCoAP uses variable RTOs. In the case of a small RTO, a packet may be received immediately after its timeout, resulting in the retransmission of its duplication. However, no congestion loss occurred during this experiment. Note that the delay was not computed for retransmitted duplications. The results indicate that FuzzyCoAP flows can compete for high throughput without congestion loss. Congestion situations will be studied in the subsequent experiments.

TABLE 3. Performance comparison: FuzzyCoAP flows.

	Flow 1	Flow 2	Flow 3
Sent packets	2601	2641	2618
Received packets (%)	2601 (100%)	2641 (100%)	2618 (100%)
Lost packets (%)	0 (0%)	0 (0%)	0 (0%)
Retransmissions (%)	1 (0.04%)	1 (0.04%)	1 (0.04%)
Duplicated packets (%)	1 (0.04%)	1 (0.04%)	1 (0.04%)
Average Delay	387.79 ms	386.41 ms	385.92 ms
Average Throughput	8.49 Kbps	8.62 Kbps	8.55 Kbps

C. HETEROGENOUS TRAFFIC SCENARIOS

1) COMPARATIVE PERFORMANCE FOR THE FuzzyCoAP AND BASIC CoAP FLOWS

We used a dumbbell network topology, as shown in Fig. 15, for the heterogeneous traffic scenarios (i.e., a mixture of basic CoAP and FuzzyCoAP flows).

For the experiments, the following parameters were used:

- The link bandwidth between the AP and router was 250 Kbps with a link delay of 50 ms.
- The link bandwidth between the router and the edge router was 60 Kbps with a link delay of 300 ms.
- The link bandwidth between the edge router and servers was 1 Mbps with a link delay of 20 ms.

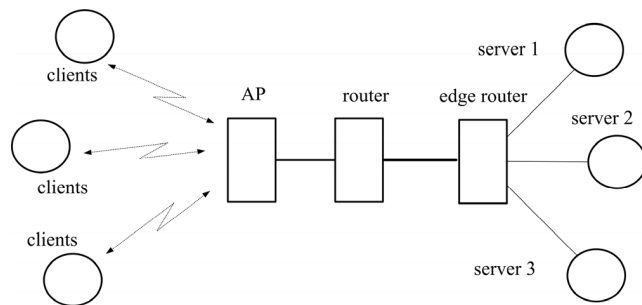


FIGURE 15. Simulation model for the heterogenous traffic scenarios.

The link between the router and the edge router is a bottleneck. For the link between the edge router and servers, we used a higher link bandwidth to quickly solve the inflight packets. This configuration ensures that only one bottleneck link exists in the end-to-end connection path.

Fig. 16 shows a comparison of the packet delays in FuzzyCoAP and basic CoAP flows. As indicated in the figure, the packet delay fluctuated in both schemes owing to competition between the flows. If the load increases, more packets will be delayed, and the delay may increase. However, the maximum delay was less than 430 ms for all the flows. The density in the plot indicates the number of packets sent by each flow, with a corresponding delay.

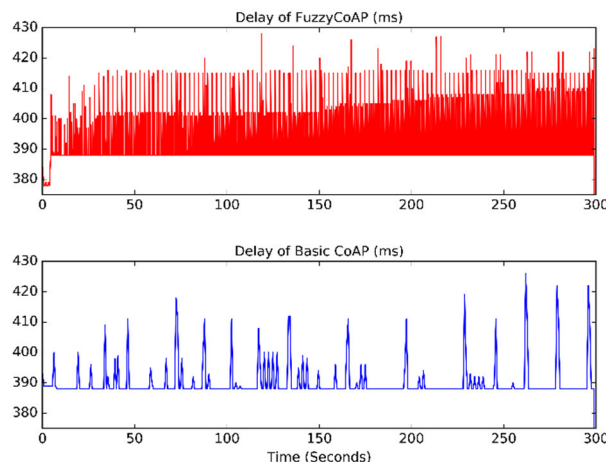


FIGURE 16. End-to-end delay of FuzzyCoAP and basic CoAP.

For the basic CoAP, the average delay was 390.31 ms for 389 packets sent within 300 s of the simulation. The confidence interval was computed for 389 samples using a confidence level of 99%, giving confidence intervals of (389.46, 391.17). The sending rate does not change. The delay variation was only due to the delayed ACKs and competition of the flows.

In the case of FuzzyCoAP, the average delay was 392.43 ms, but for 2558 packets. The confidence interval was computed for 2558 samples using a confidence level of 99%, giving confidence intervals of (392.02, 392.85). That is, the FuzzyCoAP successfully delivered more packets than the

basic CoAP. This is because FuzzyCoAP sent higher sending rates using FCS. It had more packets in flight and, in turn, more delay variation owing to the CC control. This is why FuzzyCoAP has more delayed packets than the basic CoAP. However, the sending rate of FuzzyCoAP was bounded by the bottleneck bandwidth (60 Kbps in these experiments). Thus, FuzzyCoAP limits the maximum delay in flows and avoids congestion.

Fig. 17 shows the throughput evaluation for the FuzzyCoAP and basic CoAP flows. Owing to the bottleneck bandwidth and competition of flows, the basic CoAP flow cannot send more packets than the allocated available bandwidth. By contrast, FuzzyCoAP can send more packets. At startup, the FuzzyCoAP flow attempted to measure the available bandwidth. This is why there was some spike in the figure. Subsequently, the FuzzyCoAP flow attempted to increase throughput while avoiding congestion. As explained in Section III, the instantaneous sending rate may be temporarily larger than the bottleneck bandwidth, without causing congestion. The results provide evidence that FuzzyCoAP operates in a congestion avoidance zone with a high throughput and low delay, as shown in Figure 1.

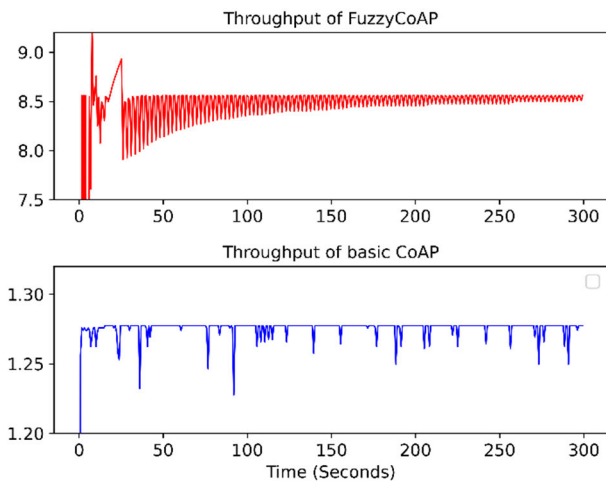


FIGURE 17. Throughput comparison for FuzzyCoAP and basic CoAP.

TABLE 4. Performance comparison: FuzzyCoAP and basic CoAP.

	FuzzyCoAP	Basic CoAP
Sent packets	2558	389
Received packets (%)	2558 (100%)	389 (100%)
Lost packets (%)	0 (0%)	0 (0%)
Retransmissions (%)	1 (0.04 %)	0 (0.0 %)
Duplicated packets (%)	1 (0.04 %)	0 (0.0 %)
Average Delay	392.43 ms	390.31 ms
Average Throughput	8.35 Kbps	1.27 Kbps

The average throughput of the basic CoAP flows was 1.27 Kbps with confidence intervals of (1.26, 1.28) using a

confidence level of 99%. In contrast, the average throughput was approximately 8.35 Kbps in FuzzyCoAP with confidence intervals of (8.24, 8.50) using a confidence level of 99%. This means that FuzzyCoAP achieved high performance without congestion. Table 4 presents a performance comparison of FuzzyCoAP and CoAP.

The FuzzyCoAP flow successfully sent 2558 packets, whereas the basic CoAP flow sent only 389 packets on average. The average throughput of FuzzyCoAP was much higher than that of the basic CoAP flows. Both schemes exhibited no packet losses. No congestion loss was observed in either scheme. The average delay was 392.43 ms for 2558 packets in FuzzyCoAP. The basic CoAP flows had an average delay of 390.31 ms for 389 packets. This means that FuzzyCoAP is more efficient than the basic CoAP because it can maintain a high throughput with a low packet delay in the competition.

A single retransmission was observed in FuzzyCoAP owing to timeout. This retransmitted packet was duplicated. This is because of the variable RTOs of FuzzyCoAP, as explained previously. In addition, neither scheme computes the delay for the retransmitted duplications. Duplicated packets were discarded and did not affect the average delay of the flows. A retransmission percentage of 0.04% is acceptable for a high number of packets sent by FuzzyCoAP.

The results indicated that FuzzyCoAP flows can send more packets with congestion avoidance based on the FCS. FuzzyCoAP is more efficient than basic CoAP in terms of high throughput, high received rate, and acceptable delay under high-load conditions.

## 2) SINGLE CONGESTION SCENARIOS

In this experimental set, we investigate the case of a single congestion situation by adding an unresponsive active flow. This active flow sent several packets at a high sending rate ranging from 20 to 90 Kbps to cause a single congestion during the time interval from 50 to 120 s for the basic CoAP flow and FuzzyCoAP flow. Fig 18 shows the delay comparison between the FuzzyCoAP and basic CoAP flows. The delay values were large between 50 s and 130 s because of several retransmission timeouts.

Again, there were more delayed packets in FuzzyCoAP than in basic CoAP because FuzzyCoAP sent more packets than basic CoAP. However, the average delay in FuzzyCoAP was less than that in the basic CoAP, owing to the variable RTOs. The RTO was doubled for each retransmission in both the schemes. Thus, the delay was longer than 25 s for some retransmitted packets.

Fig. 19, Fig. 20, and Fig. 21 present the retransmissions, duplicated retransmissions, and throughput of FuzzyCoAP and basic CoAP, respectively, under the same single congestion. There were *nine* retransmissions in in FuzzyCoAP and *seven* in the basic CoAP, as indicated in Fig. 19. The vertical axis represents the number of retransmission attempts for each packet. In this figure, *four* packets are successfully received using *four* retransmission attempts for both the schemes. In Fig. 20, the number of duplicated packets is



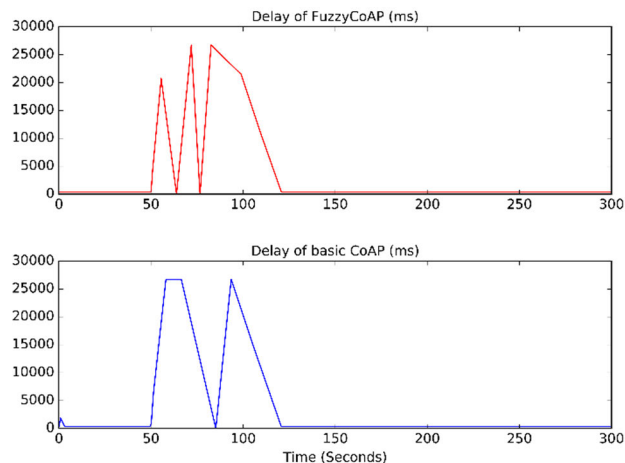


FIGURE 18. Delay of schemes in single congestion situation.

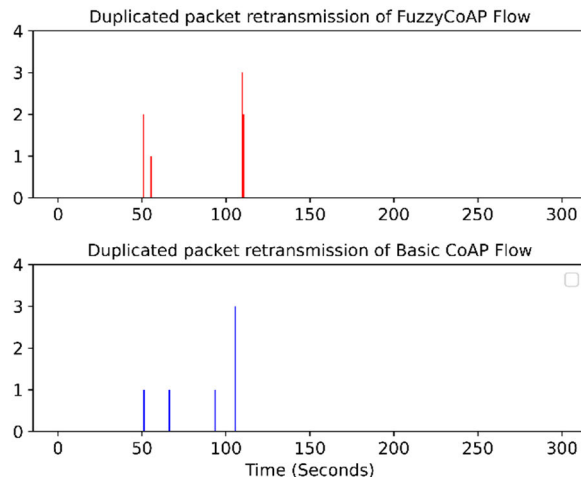


FIGURE 20. Duplicated packet retractions in single congestion.

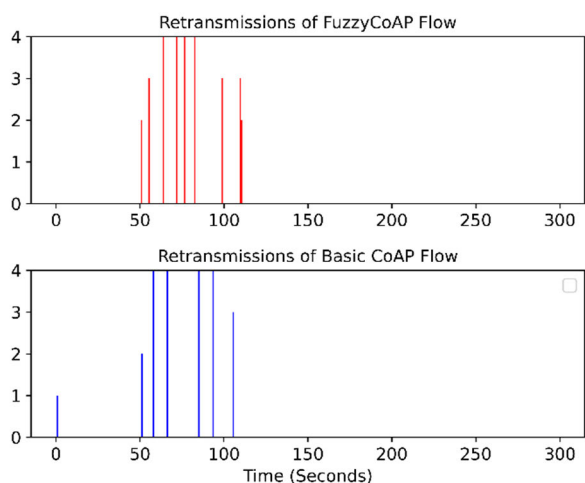


FIGURE 19. Retransmissions of schemes in single congestion.

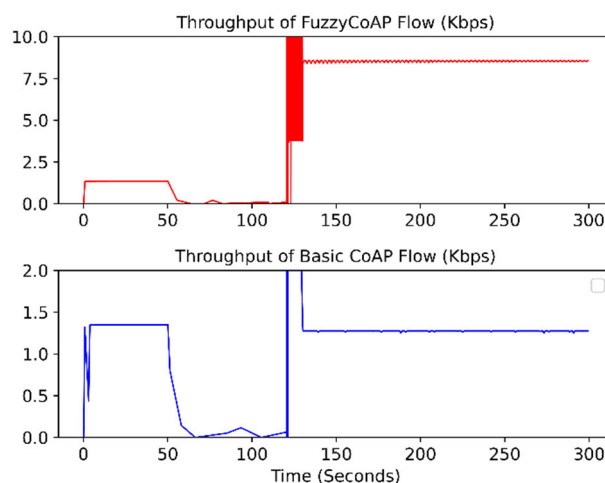


FIGURE 21. Throughput of schemes in single congestion.

one, two, or three if the same packet is duplicated one, twice, or thrice, respectively. The results indicated that FuzzyCoAP and basic CoAP have the same number of duplicated packets (i.e., the same four duplicated packet retransmissions) under these experimental conditions.

Fig. 21 shows the average throughput of both schemes. At startup, the throughput increased with the load. Owing to the competition with the unresponsive flow, the throughput of FuzzyCoAP was approximately 1.8 Kbps and remained unchanged from 0 s to 50 s. The throughput of the basic CoAP was unchanged at approximately 1.4 Kbps during the same time interval. When congestion occurred from 50 s to 130 s, both schemes were in the backoff phase with low throughput.

After the congestion had been resolved at 130 s, both schemes attempted to improve performance. The Fuzzy-CoAP flow checked for available bandwidth and achieved a throughput of 8.50 Kbps. In contrast, the throughput of the basic CoAP flow was only approximately 1.4 Kbps. The unresponsive flow terminated and released its occupied bandwidth

to other flows. There was only one FuzzyCoAP flow and one basic CoAP flow in the time interval from 130 to 300 s. The FuzzyCoAP flow gained the bandwidth released by the unresponsive flow. This is why its throughput was higher after 130 s compared to the previous time interval. The results indicate that FuzzyCoAP can achieve a higher performance than the basic CoAP when the network operation becomes normal without congestion.

Table 5 summarizes the results of the experimental set. Within a simulation time of 300 s, the FuzzyCoAP flow sent 1660 packets. Among them, 1658 packets were successfully received, and two packets were lost because of congestion. By contrast, the basic CoAP flow sent only 372 packets, with 370 successful received and only two lost packets. This is because FuzzyCoAP can adapt the sending rate based on the FCS according to the congestion state, whereas basic CoAP cannot. The number of retransmissions was nine in FuzzyCoAP and seven in the basic CoAP. Both schemes had four packet duplications. However, the

percentages were much lower for FuzzyCoAP owing to the high number of delivered packets. The average delay in FuzzyCoAP (458.64 ms) was lower than that of the basic CoAP (650.23 ms). Consequently, the average throughput of the FuzzyCoAP (5.40 Kbps) was much higher than that of the basic CoAP (1.21 Kbps). The mean values were computed using 1658 samples in FuzzyCoAP and 370 samples in basic CoAP.

The results indicated that in the case of single congestion scenarios, FuzzyCoAP is more efficient than basic CoAP in terms of average throughput, average delay, packet loss, retransmissions, and duplication percentages.

### 3) HEAVY CONGESTION SCENARIOS

In this set of experiments, we investigated the case of heavy congestion by adding an unconfirmable CoAP flow. The common link bandwidth was changed to 75 Kbps with a link delay of 300 ms. The unresponsive flow started sending packets at 2 Kbps during the time interval from 0 to 160 s. At 160 s, the rate was increased in a stepwise manner by 1 Kbps during the time interval from 160 to 200 s. This action aimed to create severe congestion at the end of the simulation period. After 200 s, the unresponsive flow decreased its rate in a stepwise manner by 2 Kbps to quickly resolve the inflight packets. Heavy congestion can be expected in the time interval between 180 s and 280 s. Using this experimental set, we show the behavior of FuzzyCoAP and basic CoAP under a heavy congestion situation.

Fig 22 shows the delay for both schemes. During the interval from 0 to 160 s, both the FuzzyCoAP and basic CoAP schemes operated without congestion. As the load increased with unresponsive flow, the delay in both schemes increased. Several retransmissions occurred from 155 to 180 s. At 180 s, packet losses were observed in both schemes (the delay was not computed for lost packets). The delay values were large between 220 s and 280 s because of the retransmission timeout of the packets. The delay plot of FuzzyCoAP is sparser than that of the basic CoAP. This means that FuzzyCoAP had fewer delayed packets than the basic CoAP. The figure indicates that the average delay in FuzzyCoAP was less than that in the basic CoAP. This is because FuzzyCoAP uses variable RTOs provided by the FCS. Thus, the number of retransmissions in FuzzyCoAP was less than that in the basic CoAP (see the density of the plot).

Fig. 23 presents the retransmissions for both schemes. As indicated, there were several retransmissions in both schemes during the congestion interval. Most packets required a maximum number of *four* retransmissions. This is why the average delay was high for both schemes. In the case of heavy congestion, FuzzyCoAP attempted to retransmit lost packets similar to basic CoAP. However, the retransmissions of FuzzyCoAP were less than those of the basic CoAP. The number of retransmissions was 669 for FuzzyCoAP and 2954 for the basic CoAP. This is because the variable RTOs in FuzzyCoAP help reduce the number of unnecessary retransmissions.

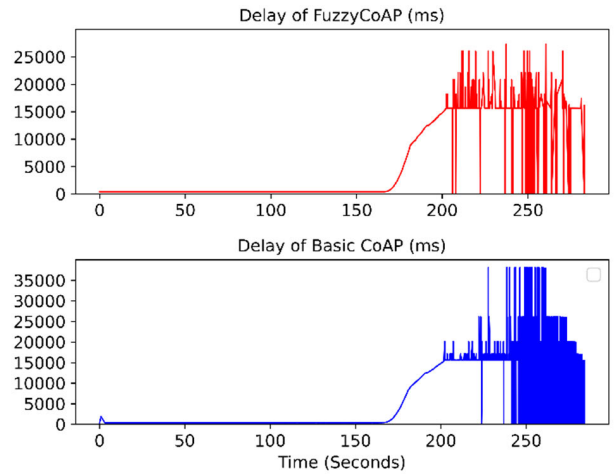


FIGURE 22. Delay of schemes in heavy congestion.

TABLE 5. Performance comparison under single congestion.

	FuzzyCoAP	Basic CoAP
Sent packets	1660	372
Received packets (%)	1658 (99.88 %)	371 (99.73 %)
Lost packets (%)	2 (0.12 %)	1 (0.27 %)
Retransmissions (%)	9 (0.54 %)	7 (1.88 %)
Duplicated packets (%)	4 (0.24 %)	4 (1.08 %)
Average Delay	458.64 ms	650.23 ms
Average Throughput	5.40 Kbps	1.21 Kbps

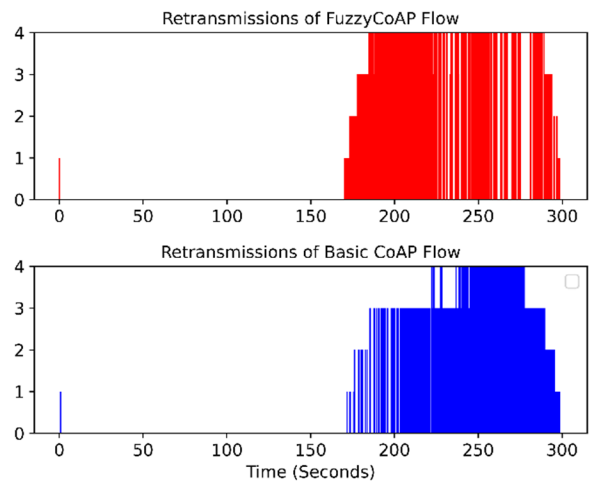


FIGURE 23. Retransmissions of schemes in heavy congestion.

Fig. 24 shows the duplicated retransmissions in the FuzzyCoAP and the basic CoAP. Several packets were duplicated during retransmission in both schemes. The number of duplications was 571 for FuzzyCoAP and 922 for basic CoAP. As shown in the figure, several packets were duplicated *four* times. The number of duplications used *one* time, *two* times,

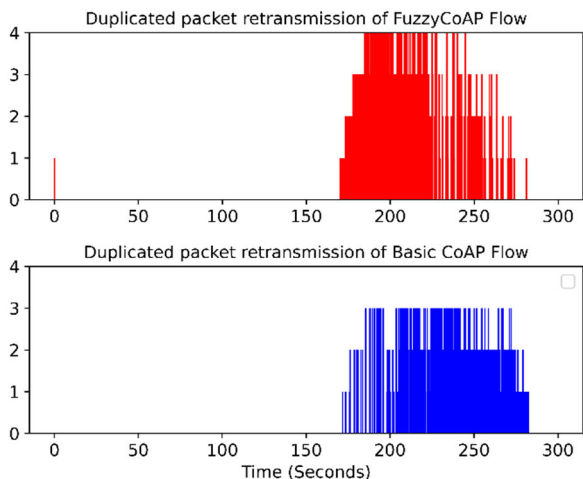


FIGURE 24. Duplicated packet retrasmissons in heavy congestion.

and *three* times was smaller. The duplicated packets were discarded at the receiver and neglected in the delay computation.

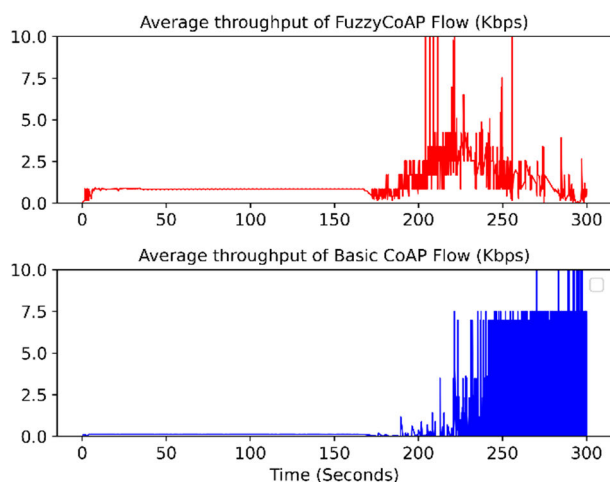


FIGURE 25. Throughput of schemes in heavy congestion.

In Fig. 25, we present the throughput comparison of the schemes. It should be noted that the average throughput was computed for acknowledged packets, including duplicated packets. Keeping this in mind, the plot does not reflect the real throughput of the flows (i.e., only acknowledged packets). The condensed plot of the basic CoAP was due to many retransmissions and duplications from 230 s to 300 s. The real average throughput of the flows was smaller in these heavy congestion scenarios. Although the plot of FuzzyCoAP was sparser, FuzzyCoAP provided a real average throughput that was much higher than that of the basic CoAP.

Table 6 presents a performance comparison of the schemes in heavy congestion scenarios.

As shown in the table, the number of sent packets was high in both schemes owing to the high bandwidth and number of retransmissions. However, the number of successful

TABLE 6. Performance comparison under heavy congestion.

	FuzzyCoAP	Basic CoAP
Sent packets	2106	3172
Received packets (%)	2020 (95.92 %)	1840 (58.01 %)
Lost packets (%)	86 (4.10 %)	1332 (41.99 %)
Retransmissions (%)	669 (31.77 %)	2954 (93.13 %)
Duplicated packets (%)	571 (27.11 %)	922 (29.07 %)
Average Delay	4304.14 ms	8639.60 ms
Average Throughput	6.97 Kbps	6.32 Kbps

received packets in FuzzyCoAP was significantly higher than that in the basic CoAP. This number was 2020 (95.92%) in FuzzyCoAP and 1840 (58.01%) in the basic CoAP. More lost packets were in basic CoAP than in FuzzyCoAP. The number of lost packets was 86 (4.10%) for FuzzyCoAP and 1332 (41.99%) for basic CoAP. The number of packet retransmissions was 669 (31.77%) in FuzzyCoAP, which was much smaller compared to 2954 (93.13%) in basic CoAP. The number of duplicated packets was 571 (27.11%) for FuzzyCoAP, and 922 (29.07%) for basic CoAP. All percentage values were computed with respect to the number of packets sent. Due to the higher number of successful received packets (duplicated packets were excluded), the average delay was 4304.14 ms in FuzzyCoAP, which was much smaller than 8639.60 ms in basic CoAP. The average throughput was 6.97 Kbps in FuzzyCoAP and was 6.32 Kbps in basic CoAP.

The simulation results indicated that FuzzyCoAP provided better performance than the basic CoAP in the case of heavy congestion.

D. DYNAMIC TRAFFIC SCENARIOS

1) DYNAMIC SCENARIOS USING FuzzyCoAP, BASIC CoAP, AND UNRESPONSIVE FLOWS

Using this experimental set, we studied the behavior of FuzzyCoAP and the basic CoAP in dynamic traffic scenarios. We used the same network topology with *ten* flows, as shown in Fig. 15. *One* FuzzyCoAP flow competes for the available bandwidth with *eight* basic CoAP flows and *one* non-confirmable flow (a basic UNCON CoAP flow, called BUNCON flow). The BUNCON flow started sending packets at a rate of 0.5 Kbps. The rate was increased in a step-wise manner to 25 Kbps. Subsequently, the rate dropped to 0.5 Kbps. This procedure was repeated *three* times. Finally, the BUNCON flow decreased its rate and stopped at 280 s. The aim of this action was to create a dynamic network condition for all the flows.

Fig. 26 shows the delay in the FuzzyCoAP, basic CoAP, and BUNCON flow. As indicated, the delay of all flows increased during the short congestion situation at 55, 110, and 150 s. The high load was caused by the increased rate of BUNCON flow during these time intervals. The measured delay was less than 800 ms for FuzzyCoAP, whereas it was

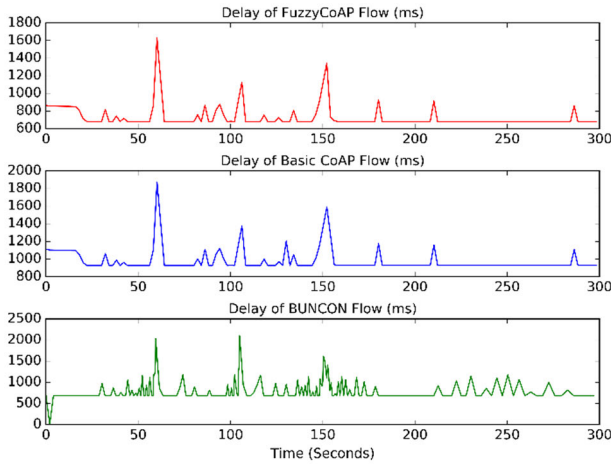


FIGURE 26. Delay performance in dynamic traffic scenarios.

approximately 950 ms on average for all basic CoAP flows. The average delay was 725.91 ms for FuzzyCoAP with confidence intervals of (700.40, 751.43) and was 973.95 for basic CoAP with confidence intervals of (948.03, 999.86).

Fig. 27 shows a throughput comparison of the schemes. In the case of congestion (from 40 s to 70 s, from 80 s to 120 s, and from 140 s to 180 s), the average throughput was 0.49 kbps with a confidence interval of (0.479, 0.501) for FuzzyCoAP and was 0.48 Kbps with a confidence interval of (0.477, 0.495) for basic CoAP, respectively. The FuzzyCoAP flow achieved a throughput comparable to that of the basic CoAP flows. This is because the BUNCON flow used almost all available bandwidth. Thus, the FuzzyCoAP flow cannot gain more available bandwidth to increase the sending rate, as shown in previous experimental set. However, the average throughput of the FuzzyCoAP flow was higher than that of the basic CoAP flows during the congestion situation. The results indicate that FuzzyCoAP reacts better than the basic CoAP in dynamic traffic scenarios.

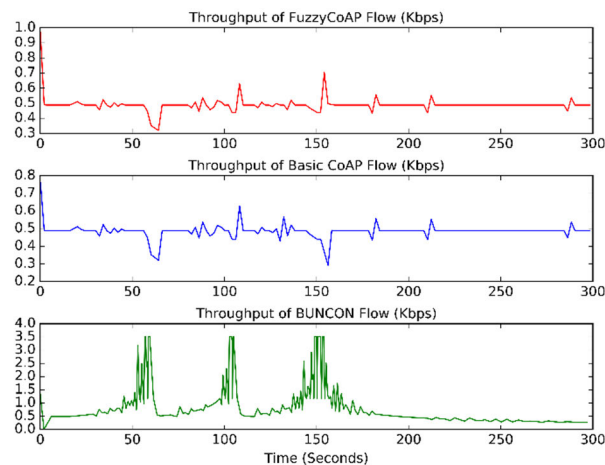


FIGURE 27. Throughput performance in dynamic traffic scenarios.

## 2) DYNAMIC SCENARIOS USING FuzzyCoAP, BASIC CoAP, UDP, AND TCP FLOWS

In this experimental set, we compared the performance of FuzzyCoAP and basic CoAP in the case of background traffic using UDP and TCP flows. The same network topology was used, as shown in Fig. 15. However, the bottleneck link bandwidth was set to 500 Kbps with a link delay of 500 ms. The TCP flow was established using the standard TCP socket of the network simulator NS-3 with a data rate of 10 Mbps and a data segment size of 1040 bits. The UDP flow was created using the standard UDP socket of NS-3. The UDP flow started with a data rate of 20 Kbps. The rate was then changed to 50 Kbps at 80 s, 150 Kbps at 150 s, and 5 Kbps at 200 s. The purpose of this experiment was to create mixed dynamic background traffic scenarios for simulation using the FuzzyCoAP and basic CoAP flows. All flows started at random times between 0 and 200 ms. We executed 30 random test runs for this experiment.

Fig. 28 shows the comparable delay performance of FuzzyCoAP and the basic CoAP in the case of TCP/UDP background traffic.

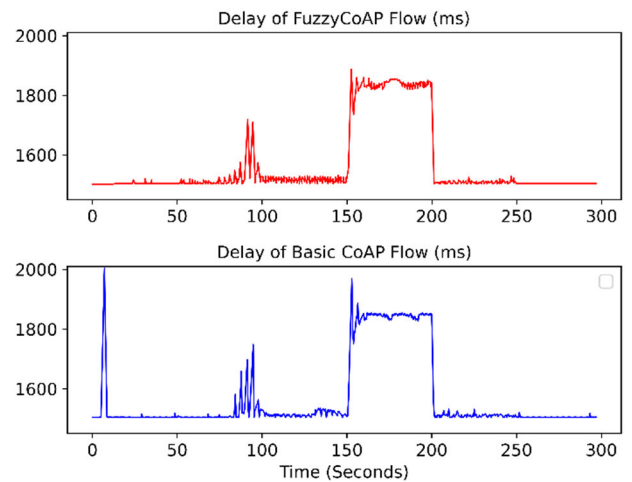


FIGURE 28. Delay performance of FuzzyCoAP and basic CoAP using TCP/UDP background traffic.

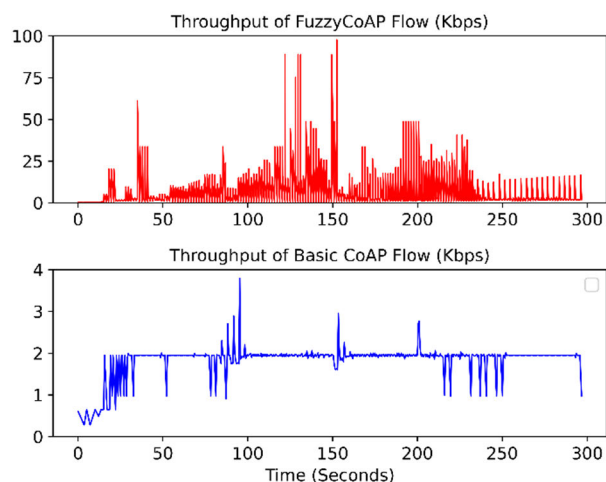
As indicated, the delay in both schemes fluctuated depending on the rate of change in the UDP flow. A large delay was observed during the time interval from 80 s to 120 ms and from 150 s to 200 s because the UDP flow sent packets at a high speed during that time interval. The average delay in the basic CoAP was approximately 1563.25 ms with confidence intervals of (1550.77, 1575.74). The large delay was 2004 ms at 7 s and 1970 ms at 153 s. By contrast, the average delay was approximately 1561.26 ms with a confidence interval of (1554.82, 1579.70) for FuzzyCoAP. The maximum delay in the FuzzyCoAP flow was 1888 ms at 153 s, which was smaller than that of the basic CoAP (2004 ms). This is because of the variable RTOs in the FuzzyCoAP flow.

Fig. 29 shows the throughput of the Fuzz CoAP and basic CoAP. In the time interval from 0 to 50 s, all flows compete

for the available bandwidth of the bottleneck link. Fuzzy-CoAP flow gained a higher bandwidth than the basic CoAP during this time interval.

At 80 s, the UDP flow was changed from 20 to 50 Kbps. At this moment, the FuzzyCoAP flow attempted to leverage the recently released bandwidth to increase its sending rate. Owing to the recently updated bottleneck bandwidth, the FuzzyCoAP flow could increase throughput during 80-90 s based on its FCS. By contrast, the basic CoAP flow cannot maintain its throughput.

During the moment of the rate change of the UDP flow at 150 s, both FuzzyCoAP flow and the basic CoAP flow attempted to gain the bandwidth released by the UDP flow. However, the basic CoAP flow could not, whereas the FuzzyCoAP could obtain some more bandwidth to increase its throughput based on its updated bottleneck bandwidth. However, the throughput of FuzzyCoAP fluctuated owing to competition between the flows. Several packets were lost for both schemes from 150 s to 200 s because of the high rate of UDP flow. Thus, the throughput of both schemes degraded accordingly.



**FIGURE 29.** Throughput performance of FuzzyCoAP and basic CoAP using TCP/UDP background traffic.

At 200 s, the UDP flow rate was decreased to 5 Kbps. FuzzyCoAP can gain more available bandwidth to increase its throughput, whereas the basic CoAP flow cannot. However, the throughput of FuzzyCoAP fluctuated because many packets remained in flight. The average throughput was approximately 1.895 Kbps with confidence intervals of (1.863, 1.927) in the basic CoAP. This is because the basic CoAP had more retransmitted and duplicated packets in the congested interval. In contrast, the average throughput was approximately 9.228 Kbps with confidence intervals of (7.907, 10.549) in FuzzyCoAP.

We can conclude that FuzzyCoAP can leverage the available bandwidth of the bottleneck shared by flows to increase its performance. By contrast, the basic CoAP could not. This is because FuzzyCoAP uses the FCS to control

the sending rate according to the dynamic network state. Thus, FuzzyCoAP can be used to predict early congestion. FuzzyCoAP can limit the maximum delay while maintaining a high throughput performance.

All experiment results demonstrate that the rate-based FuzzyCoAP behaves more efficiently than the loss-based basic CoAP in various traffic scenarios.

## E. DISCUSSION

This paper proposes a new rate-based CC scheme for CoAP. We focused on experiments to compare the performance of the rate-based CC and the existing loss-based CC scheme, that is, the basic CoAP. As discussed in Section II, most of the existing CoAP variants follow a loss-based CC approach. We chose the basic CoAP for comparison because it is representative of the loss-based CC schemes that have been standardized. Moreover, the aim of this study is to demonstrate the feasibility and efficiency of a fuzzy rate-based CC scheme for CoAP compared to the loss-based CC scheme for CoAP. Experiments with other related schemes will be a topic for further studies.

A key issue in the design of our FCS is the computation of the bottleneck bandwidth to determine the *BG-gradient*. As presented in Section III, we used a simple method for sending back-to-back packets at the startup stage. The initial state of FuzzyCoAP requires *six* to *ten* RTT cycles to estimate the bottleneck compared with the *ten* cycles indicated in [23] and [42]. This is because the clients and network require some time to be stable. This overhead is negligible compared to the total time of burst data transfer. In addition, another method can be deployed for bottleneck bandwidth estimation, such as the packet pair method indicated in [58]. However, this issue will be a topic for further study.

In the proposed FuzzyCoAP, we use the *C-degree* to adjust the sending rate of the clients, as well as the RTO values for retransmissions. Accurate computation of variable RTOs can reduce the number of unnecessary retransmissions and packet duplications. Our experiment results indicate that the *C-degree* can be used to adjust the RTOs, which are used to adjust the retransmission speed. The issue of RTO adjustment is a topic for further research.

## VI. CONCLUSION

Constrained Application Protocol (CoAP) has been adopted for IoT networks to satisfy the growing demand for numerous smart applications. Congestion control (CC) is one of the most challenges in such networks owing to the requirements for reliable burst data transfer and the dynamic network conditions. CoAP uses a basic CC algorithm that only regulates the retransmission rate, that is, in the case of timeout or packet loss. This typical loss-based CC algorithm operates only when a network congestion occurs. Enhancement of CoAP CC is undoubtedly necessary, particularly for burst data transfers.

In this study, we proposed a novel rate-based CC scheme called FuzzyCoAP based on fuzzy control system (FCS).

An FCS can be an excellent choice because parameters such as RTT, delivery rate, and traffic load are not sharp, unclear, approximate, or ambiguous. Thus, an FCS can help in making appropriate decisions for congestion control. We chose the *RT-gradient* and *BG-gradient* as inputs to the FCS. The crisp output of the FCS, that is, the *C-degree*, reflects the congestion state. Using the FCS, the proposed FuzzyCoAP can detect early congestion and adjust the sending rate to avoid congestion threats. To the best of our knowledge, this study is the first to address the application of FCS for CoAP CC using a rate-based approach.

Numerous experiments and traffic scenarios have been conducted to demonstrate the feasibility of the proposed FuzzyCoAP. We compared the performance of FuzzyCoAP with that of the basic CoAP because it is representative of loss-based schemes. The results demonstrate that the proposed FuzzyCoAP is feasible. FuzzyCoAP frequently checks for network congestion and reduces the likelihood of congestion by decreasing the transmission rate. Furthermore, FuzzyCoAP enables high throughput by dynamically checking the available bandwidth to increase the sending rate. FuzzyCoAP is more efficient than basic CoAP in terms of delay, throughput, retransmission, packet duplication, and packet loss ratio in different traffic scenarios. Moreover, FuzzyCoAP is efficient for burst data transfer.

FuzzyCoAP uses a variable RTO estimation based on the computed congestion degree provided by the FCS. Therefore, FuzzyCoAP reacts better to dynamic network states than basic CoAP in both single and heavy congestion situations. Using variable RTOs, FuzzyCoAP can reduce the number of retransmissions, packet duplications, and loss rate, while maintaining a reasonable delay and high throughput. The use of the computed congestion degree for the RTO adjustment can be studied in future research.

## ACKNOWLEDGMENT

The authors would like to thank the Hanoi University of Civil Engineering (HUCE) and the Posts and Telecommunications Institute of Technology (PTIT) for the facilities, and also would like to thank the anonymous reviewers for their valuable comments.

## REFERENCES

- [1] *The Constrained Application Protocol (CoAP)*, document RFC 7252, Jun. 2014. [Online]. Available: <https://rfc-editor.org/info/rfc7252>
- [2] *CUBIC for Fast Long-Distance Networks*, document RFC 8312, [Online]. Available: <https://rfc-editor.org/info/rfc8312>
- [3] T. Henderson, S. Floyd, A. Gurtov, and Y. Nishida, *The NewReno Modification to TCP's Fast Recovery Algorithm*, document RFC 6582, Apr. 2022, [Online]. Available: <https://rfc-editor.org/info/rfc6582>
- [4] C. Gomez, A. Arcia-Moret, and J. Crowcroft, "TCP in the Internet of Things: From ostracism to prominence," *IEEE Internet Comput.*, vol. 22, no. 1, pp. 29–41, Jan./Feb. 2018.
- [5] C. Bormann and Z. Shelby, *Block-Wise Transfers in the Constrained Application Protocol (CoAP)*. Accessed: Jun. 24, 2021. [Online]. Available: <https://rfc-editor.org/info/rfc7959>
- [6] W. U. Rahman, Y.-S. Choi, and K. Chung, "Performance evaluation of video streaming application over CoAP in IoT," *IEEE Access*, vol. 7, pp. 39852–39861, 2019.
- [7] H. Haile, K.-J. Grinnemo, S. Ferlin, P. Hurtig, and A. Brunstrom, "End-to-end congestion control approaches for high throughput and low delay in 4G/5G cellular networks," *Comput. Netw.*, vol. 186, Feb. 2021, Art. no. 107692.
- [8] H. Jiang, Q. Li, Y. Jiang, G. Shen, R. Sinnott, C. Tian, and M. Xu, "When machine learning meets congestion control: A survey and comparison," *Comput. Netw.*, vol. 192, Jun. 2021, Art. no. 108033.
- [9] V. Jacobson, "Congestion avoidance and control," in *Proc. Symp. Proc. Commun. Archit. Protocols SIGCOMM*, New York, NY, USA, 1988, pp. 314–329.
- [10] S. Floyd, R. Gummadi, and S. Shenker, "Adaptive RED: An algorithms for increasing the robustness of RED's active queue management," *Int. Comput. Sci. Inst. (ICSI)*, Berkeley, CA, USA, Tech. Rep. 301, Aug. 2001. [Online]. Available: <https://www.icsi.berkeley.edu/icsi/node/2032>
- [11] A. Betzler, C. Gomez, I. Demirkol, and J. Paradells, "CoAP congestion control for the Internet of Things," *IEEE Commun. Mag.*, vol. 54, no. 7, pp. 154–160, Jul. 2016.
- [12] A. Betzler, C. Gomez, I. Demirkol, and J. Paradells, "CoCoA+: An advanced congestion control mechanism for CoAP," *Ad Hoc Netw.*, vol. 33, pp. 126–139, Oct. 2016.
- [13] C. Bormann, A. Betzler, C. Gomez, and I. Demirkol. (Feb. 2018). *CoAP Simple Congestion Control/Advanced*. Internet-Draft. Accessed: Jul. 24, 2021. [Online]. Available: <https://tools.ietf.org/id/draft-bormann-core-cocoa-03.txt>
- [14] J. J. Lee, K. T. Kim, and H. Y. Youn, "Enhancement of congestion control of constrained application protocol/congestion control/advanced for Internet of Things environment," *Int. J. Distrib. Sensor Netw.*, vol. 12, no. 11, pp. 1–13, Nov. 2016.
- [15] I. Jarvinen, M. Kojo, I. Raitahila, and Z. Cao, *Fast-Slow Retransmission Timeout and Congestion Control Algorithm for CoAP*, document draft-ietf-core-fasor-01, IETF CoRE Working Group, 2020. [Online]. Available: <https://core-wg.github.io/fasor/draft-ietf-core-fasor.html>
- [16] S. Bolettieri, G. Tanganelli, C. Vallati, and E. Mingozzi, "PCoCoA: A precise congestion control algorithm for CoAP," *Ad Hoc Netw.*, vol. 80, pp. 116–129, Nov. 2018.
- [17] S. Deshmukh and V. T. Raisinghani, "AdCoCoA-adaptive congestion control algorithm for CoAP," in *Proc. 11th Int. Conf. Comput., Commun. Netw. Technol. (ICCCNT)*, Kharagpur, India, Jul. 2020, pp. 1–7.
- [18] R. Jain, "A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks," *CM SIGCOMM Comput. Commun. Rev.*, vol. 19, no. 5, Oct. 1989, pp. 56–71, doi: [10.1145/74681.74686](https://doi.org/10.1145/74681.74686).
- [19] A. Gurtov, T. Henderson, and S. Floyd, *The NewReno Modification to TCP's Fast Recovery Algorithm*, document RFC 3782, 2004.
- [20] S. Ha, I. Rhee, and L. Xu, "CUBIC: A new TCP-friendly High-speed TCP variant," in *Proc. SIGOPS Oper. Syst. Rev.*, vol. 42, no. 5, 2008, pp. 64–74.
- [21] R. Mittal, V. T. Lam, N. Dukkupati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats, "TIMELY: RTT-based congestion control for the datacenter," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 537–550, Oct. 2015.
- [22] S. Liu, T. Başar, and R. Srikant, "TCP-Illinois: A loss- and delay-based congestion control algorithm for high-speed networks," *Perform. Eval.*, vol. 65, nos. 6–7, pp. 417–440, Jun. 2008.
- [23] E. Ancillotti and R. Bruno, "BDP-CoAP: Leveraging bandwidth-delay product for congestion control in CoAP," in *Proc. IEEE 5th World Forum Internet Things (WF-IoT)*, Limerick, Ireland, Apr. 2019, pp. 656–661.
- [24] E. Ancillotti, E. Bolettieri, and R. Bruno, "RTT-based congestion control for the Internet of Things," in *Proc. 19th IEEE Int. Symp. Wired/Wireless Internet Commun. (WWIC)*, vol. 10866, Jun. 2018, pp. 3–15.
- [25] R. Rejaie, M. Handley, and D. Estrin, "RAP: An end-to-end rate-based congestion control mechanism for realtime streams in the internet," in *Proc. IEEE INFOCOM Comput. Commun. Societies*, vol. 3, Mar. 1999, pp. 1337–1345.
- [26] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-based congestion control for unicast applications," in *Proc. ACM SIGCOMM Comput. Commun. Rev.*, Oct. 2000, vol. 30, no. 4, pp. 43–56.
- [27] D. H. Hoang, *Quality of Service Control in the Mobile Wireless Environment*. Bern, Switzerland: PeterLang Publisher, 2002.
- [28] E. Ancillotti, R. Bruno, C. Vallati, and E. Mingozzi, "Design and evaluation of a rate-based congestion control mechanism in CoAP for IoT applications," in *Proc. IEEE 19th Int. Symp. World Wireless, Mobile Multimedia Networks (WoWMoM)*, Chania, Greece, Jun. 2018, pp. 14–15.
- [29] D. H. Hoang and T. T. D. Le, "RCOAP: A rate control scheme for reliable bursty data transfer in IoT networks," *IEEE Access*, vol. 9, pp. 169281–169298, 2021, doi: [10.1109/ACCESS.2021.3135435](https://doi.org/10.1109/ACCESS.2021.3135435).

- [30] T. J. Ross, *Fuzzy Logic with Engineering Applications*, 3rd Ed. Hoboken, NJ, USA: Wiley, 2010.
- [31] S. Keshav, "A control-theoretic approach to flow control," in *Proc. SIGCOMM Comput. Commun. Rev.*, Sep. 1991, vol. 21, no. 4, pp. 3–15, doi: [10.1145/115994.115995](https://doi.org/10.1145/115994.115995).
- [32] C. Chrysostomou, A. Pitsillides, L. Rossides, M. Polycarpou, and A. Sekercioglu, "Congestion control in differentiated services networks using Fuzzy-RED," *IFAC Control Eng. Pract.*, vol. 11, no. 19, pp. 1153–1170, 2003.
- [33] S. Jaiswal and A. Yadav, "Fuzzy based adaptive congestion control in wireless sensor networks," in *Proc. 6th Int. Conf. Contemp. Comput. (IC3)*, Aug. 2013, pp. 433–438.
- [34] A. A. Rezaee and F. Pasandideh, "A fuzzy congestion control protocol based on active queue management in wireless sensor networks with medical applications," *Wireless Pers. Commun.*, vol. 98, no. 1, pp. 815–842, Jan. 2018, doi: [10.1007/s11277-017-4896-6](https://doi.org/10.1007/s11277-017-4896-6).
- [35] C. Sonmez, O. D. Incel, S. Isik, M. Y. Donmez, and C. Ersoy, "Fuzzy-based congestion control for wireless multimedia sensor networks," *EURASIP J. Wireless Commun. Netw.*, vol. 2014, no. 1, p. 63, Dec. 2014.
- [36] W. Chang, P. Chen, and C. Yang, "Robust fuzzy congestion control of TCP/AQM router via perturbed Takagi–Sugeno fuzzy models," *Intl. J. Fuzzy Syst.*, vol. 15, no. 2, pp. 203–213, Jun. 2013.
- [37] M. Zarei, A. M. Rahmani, and R. Farazkish, "CCTF: Congestion control protocol based on trustworthiness of nodes in wireless sensor networks using fuzzy logic," *Int. J. Ad Hoc Ubiquitous Comput.*, vol. 8, Jan. 2011, pp. 54–63.
- [38] P. Aimtongkham, T. G. Nguyen, and C. So-In, "Congestion control and prediction schemes using fuzzy logic system with adaptive membership function in wireless sensor networks," *Wireless Commun. Mobile Comput.*, vol. 2018, pp. 1–19, Aug. 2018.
- [39] M. H. Homaei, F. Soleimani, S. Shamshirband, A. Mosavi, N. Nabipour, and A. R. Varkonyi-Koczy, "An enhanced distributed congestion control method for classical 6LoWPAN protocols using fuzzy decision system," *IEEE Access*, vol. 8, pp. 20628–20645, 2020, doi: [10.1109/ACCESS.2020.2968524](https://doi.org/10.1109/ACCESS.2020.2968524).
- [40] G. Cui, H. Wang, Z. Fan, and P. Li, "NDN congestion control based on fuzzy comprehensive evaluation algorithm," *J. Phys., Conf.*, vol. 1883, no. 1, Apr. 2021, Art. no. 012008.
- [41] P. Aimtongkham, P. Horkaew, and C. So-In, "An enhanced CoAP scheme using fuzzy logic with adaptive timeout for IoT congestion control," *IEEE Access*, vol. 9, pp. 58967–58981, 2021.
- [42] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-based congestion control," *Queue*, vol. 14, no. 5, p. 50, Oct. 2016.
- [43] *Computing TCP's Retransmission Timer*, document RFC 6298, [Online]. Available: <https://rfc-editor.org/info/rfc6298>
- [44] C. Suwannapong and C. Khunboa, "Congestion control in CoAP observe group communication," *Sensors*, vol. 19, no. 15, p. 3433, Aug. 2019, doi: [10.3390/s19153433](https://doi.org/10.3390/s19153433).
- [45] J. H. Jung, M. Gohar, and S. J. Koh, "CoAP-based streaming control for IoT applications," *Electronics*, vol. 9, no. 8, p. 1320, 2020, doi: [10.3390/electronics9081320](https://doi.org/10.3390/electronics9081320).
- [46] M. A. Tariq, M. Khan, M. T. R. Khan, and D. Kim, "Enhancements and challenges in CoAP—A survey," *Sensors*, vol. 20, no. 21, p. 6391, 2020, doi: [10.3390/s20216391](https://doi.org/10.3390/s20216391).
- [47] L. A. Zadeh, "Toward a theory of fuzzy systems," NASA, Washington, DC, USA, Tech. Rep. CR-1432, 1969, pp. 1–35.
- [48] L. A. Zadeh, "Outline of a new approach to the analysis of complex systems and decision processes," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-3, no. 1, pp. 28–44, Jan. 1973.
- [49] E. H. Mamdani, "Applications of fuzzy algorithms for simple dynamic plant," *Proc. IEEE* vol. 121, no. 12, pp. 1585–1588, Dec. 1974.
- [50] R. E. Bellman and L. A. Zadeh, "Decision-making in a fuzzy environment," *Manage. Sci.*, vol. 17, no. 4, pp. 141–164, Dec. 1970.
- [51] S.-H. Chen, "Ranking fuzzy numbers with maximizing set and minimizing set," *Fuzzy Sets Syst.*, vol. 17, no. 2, pp. 113–129, Nov. 1985.
- [52] H. J. Zimmerman, *Fuzzy Set Theory—And Its Applications*, vol. 17, 4th ed. Norwell, MA, USA: Kluwer Academic, 2001.
- [53] D. H. Esawi, G. Attiya, and G. Allam, "Fuzzy controller based TCP-Vegas enhancement for congestion control," *Menoufia J. Electron. Eng. Res.*, vol. 30, no. 2, pp. 39–44, Jul. 2021.
- [54] *NS-3 Network Simulator, NS3.36*. Accessed: Aug. 16, 2022. [Online]. Available: <https://www.nsnam.org/>
- [55] S. Maesoser. *A Partial CoAP Implementation With mDNS Support, Multicast*. Accessed: Aug. 16, 2022. [Online]. Available: <https://github.com/maesoser/ns3-coap/>
- [56] H. Hirotakaster. *CoAP Client, Server Library for Spark Photon, Spark Core*. Accessed: Aug. 16, 2022. [Online]. Available: <https://github.com/hirotakaster/CoAP>
- [57] M. Boucadair and J. Shallow. (May 2021). *Constrained Application Protocol (CoAP) Block-Wise Transfer Options Supporting Robust Transmission*. Internet-Draft. Accessed: Jun. 20, 2021. [Online]. Available: <https://tools.ietf.org/id/draft-ietf-core-new-block-14>
- [58] S. Keshav, "The packet pair flow control protocol," Int. Comput. Sci. Inst. (ICSI), Berkeley, CA, USA, Tech. Rep. TR-91-028, 1991. [Online]. Available: <http://www.icsi.berkeley.edu/pubs/techreports/tr-91-028.pdf>



**THIEU NGA PHAM** received the Diploma-Ing. (M.Eng.) degree in technical cybernetics and automation and the Dr.-Ing. degree in informatics and automation from the Technical University of Ilmenau, Germany, in 1987 and 2000, respectively.

Since 2006, she has been a Senior Lecturer at the Faculty of Information Technology, University of Civil Engineering, Hanoi, Vietnam. Her current research interests include fuzzy control, fuzzy optimization, fuzzy decision, expert systems, wireless sensor networks, the IoT networks, system techniques, and control systems.



**DANG HAI HOANG** received the Diploma-Ing. (M.Eng.) degree in technical cybernetics and automation and the Dr.-Ing. and Dr.-Ing. (Habilitation) degrees in telematics and communication systems from the Technical University of Ilmenau, Germany, in 1984, 1999, and 2002, respectively.

Since 2009, he has been an Associate Professor at the Posts and Telecommunications Institute of Technology, Hanoi, Vietnam. His current research interests include information security, communication protocols, communication systems, QoS mechanisms, and control systems.



**THI THUY DUONG LE** received the B.S. degree in telecommunications and electronic engineering and the M.S. degree from the Technical University of Hanoi, Vietnam, in 2002 and 2008, respectively. She is currently pursuing the Ph.D. degree with the Posts and Telecommunications Institute of Technology, Hanoi, Vietnam.

Since 2005, she has been a Lecturer with the Faculty of Information Technology, University of Civil Engineering, Hanoi. She is currently a Senior Lecturer. Her current research interests include computer and communication systems, wireless sensor networks, QoS mechanisms, and network performance.

• • •