A CONTROL MECHANISM FOR RELIABLE BURST DATA TRANSFER IN IoT NETWORKS

Le Thi Thuy Duong [*], Hoang Dang Hai[+], Pham Thieu Nga[*]
[*]Faculty of Information Technology, Hanoi University of Civil Engineering
[+]Posts and Telecommunications Institute of Technology

Abstract: IoT networks are widely used in various areas, such as industry, healthcare, agriculture, and the environment. Many applications require to transfer a large amount of data collected by IoT devices to a central server. Without an appropriate control mechanism, the network is prone to congestion. The Constrained Application Protocol (CoAP) has been proposed for data transmission in IoT networks. This paper analyzes the shortcomings of CoAP and indicates that CoAP does not have rate control and no support for burst data transfer. To extend CoAP, we develop an analytical model for reliable burst data transfer with CoAP using rate control. Based on this model, we propose a new rate control mechanism that allows reliable burst data transfer with high throughput, low delay, and improved congestion control of CoAP in IoT networks**.**

Keywords: Rate control, Burst data transfer, Congestion control, CoAP, IoT networks.

I. INTRODUCTION

The Internet of Things (IoT) networks are being developed and widely applied in many fields such as industry, agriculture, healthcare, transportation, and environment. Typical applications are, for instance, monitoring networks (in medical, security). In these applications, IoT devices collect data using various sensors and transfer blocks of data to a monitoring center. Such applications often require to transfer an enormous amount of collected data to a central server through the IoT network and Internet. The transmission of such data burst easily causes the risk of network congestion. The congestion control problem has been extensively studied in traditional computer networks, and became a new interested topic in IoT networks. This is because IoT networks have different characteristics compared to traditional networks. On the other hand, the design of lightweight transport protocols for IoT networks has reduced the required congestion control function. Lightweight protocols for IoT networks, such as the Constrained Application Protocol (CoAP), lack some features required for congestion control. This is why the issue of congestion control and avoidance is being a hot research topic in IoT networks.

An IoT network typically consists of multiple devices with attached sensors, which collect data from their surrounding environment. Furthermore, an IoT network includes several network devices (e.g., gateways, routers) for relaying data from an IoT network to a central processing system. IoT devices are typically small with constrained resources (limited memory and processing power). As a result, it is not possible to use the original Internet protocols for IoT devices. A study in [1] showed that TCP, that is the prominent transport protocol of the Internet, cannot be used for data transfer in IoT networks.

Recently, numerous lightweight protocols have been developed for IoT networks. The typical one is the Constrained Application Protocol (CoAP). CoAP has been standardized by the Internet Engineering Task Force (IETF) for IoT networks with RFC 7252 [2]. In essence, CoAP is a lightweight application layer protocol that runs on top of the UDP (User Datagram Protocol) layer. However, CoAP supports reliable connection-oriented data transport similar to TCP with acknowledgment (ACK) messages. Similar to TCP, CoAP has a simple congestion control mechanism that relies on timeout to retransmit packets whenever a packet loss occurs. Nevertheless, the design of CoAP reduces some congestion control facilities to keep the protocol lightweight. Numerous studies, such as [3][4], shown the limitations of CoAP in congestion control. The standard document RFC 7252 [2] indicated remaining issues of CoAP and outlined several future developments for CoAP.

In this paper, we examine two fundamental shortcomings of the CoAP: 1) CoAP does not support reliable burst data transfer; 2) CoAP does not control the sending rate with respect to congestion control. On this premise, this paper proposes a new rate-based congestion control mechanism for CoAP to support reliable burst data transfer in IoT networks. The proposed mechanism extends CoAP by adding a congestion detection and a rate adjustment mechanism to mitigate congestion. This mechanism improves the protocol performance in terms of delay and throughput. The key contributions of this paper are: 1) an analytical model for burst data transfer using CoAP, and 2) a new mechanism for CoAP rate control to avoid congestion. The remainder of this paper is organized as follows. Section II briefly presents the operation of CoAP, its shortcomings, and the related

Contact author: Hoang Dang Hai
Email: haihd@ptit.edu.vn
Manuscript received: 06/2022, revised: 08/2022, accepted: 08/2022

work. In Section III, we present our proposed control mechanism. In Section IV, we provide simulation results for the proposed control mechanism. Section V concludes the paper.

II. OVERVIEW OF COAP AND RELATED WORK

A. Overview of CoAP

As indicated in [2], CoAP has two operation modes: reliable and unreliable data transport. The reliable data transport mode performs similar to TCP. This means that CoAP uses acknowledgment (ACK) packets to confirm the transmission of confirmable (CON) packets. This paper focuses only on reliable data transport mode of CoAP. The term "packet" defines a message with a CoAP header and a payload. Figure 1 shows the operation of a CoAP in reliable transport mode. Let Ts denote the sending time of a CON packet, Ta be the receiving time of an ACK packet. The time difference Ta-Ts presents a round-trip time (RTT). RTT is the time interval between a transmitted CON packet and a received ACK from the receiver. RTO is the retransmission timeout, i.e., the time interval used to check for the receiving an ACK.

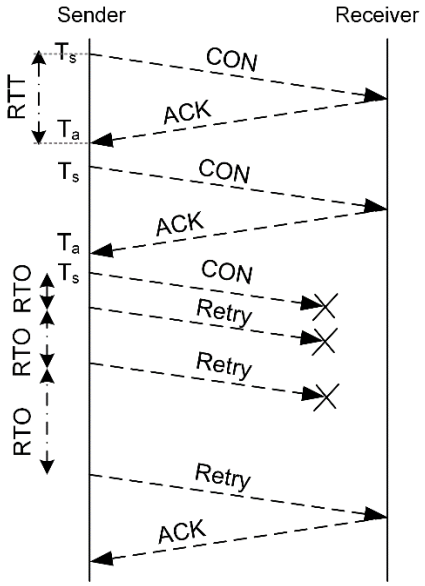

Figure 1. Operation of CoAP in reliable transport mode

As shown in the figure, CoAP uses confirmable packets (CON) and acknowledgment packets (ACK) for the reliable mode. A CoAP sender sends only the next CON packet after receiving an ACK. In this way, CoAP performs a simple stop-and-wait mechanism for reliable communication. There are idle intervals between the time of receiving an ACK and sending a next CON packet (the time difference of T_a and T_s).

The CoAP sender sets an initial RTO for each CON packet. According to [2], the initial RTO is a fixed value, which is selected between 2s and 3s. The CoAP sender may not receive an ACK for the transmitted CON packet. The reason may be: 1) the CON packet cannot arrive at the destination (as illustrated in the figure) because of channel errors, link errors, or congestion; or 2) the ACK packet sent from the receiver cannot arrive at the sender owing to the errors of the backward link and processing of the receiver. In this paper, we focus only on the loss of ACK packets owing to congestion, as indicated in [2].

When an ACK is not received within the initial RTO, the CoAP sender assumes a loss of CON packet and attempts to retransmit the lost packet (Retry in the figure). After each retransmission, the RTO value is doubled. *Four* retransmissions are allowed for each retransmitted packet. After *four* unsuccessful retries, the transmission is considered to have failed. The strategy for RTO doubling is called binary exponential backoff (BEB). Figure 1 demonstrates three lost CON packets. The last packet was successfully retransmitted after *three* attempts.

As presented, the retransmission mechanism for lost packets in CoAP is similar to TCP because CoAP assumes packet loss as an indicator of congestion. A difference is that TCP considers a triple of ACK losses. The main function of reliable transport protocols such as TCP and CoAP is the retransmission of lost packets. However, retransmitted packets may be duplicated or disordered at the receiver. In principle, the higher layer is responsible to process such problem, not TCP or CoAP. For instance, the higher application layer can discard duplicated packets and rearrange the order of received packets. Another problem may arise in case of temporal losses. In this case, packets may have lost after *four* unsuccessful retransmissions. The sender can discard the lost packets and continues to send further packets, or the connection will be interrupted. In a such situation, the higher application layer must restart the connection and retransmit the block of lost packets. Nevertheless, this issue and the functionality of higher application layers are out of scope of this paper.

B. Shortcomings of CoAP

Referring to the operation of CoAP presented in the previous section, we indicate two fundamental shortcomings of CoAP as follows: 1) the lack of support for reliable burst data transfer; 2) the lack of rate adjustment for congestion control. We analyze these shortcomings in this subsection.

First, CoAP does not support reliable burst data transfer like TCP. The CoAP can only send another packet when it receives an ACK for the previous packet. Assume that a packet *A* is sent at T_1, and this packet arrives at the receiver at T_2. The receiver gets the packet and sends an ACK back to the sender at T_3. At T_4, the sender receives this ACK. In this stop-and-go mechanism, the sender is idle for all the time interval from T_1 to T_4 except waiting for the ACK. The round-trip time RTT is equal to $T_4 - T_1$. For a long distance (e.g., the receiver is far from the sender), RTT becomes large. The allocated bandwidth for the connection would be wasted owing to long idle intervals. The default leisure time was 5 seconds for CoAP [2]. This means that CoAP may wait for the acknowledgments at a fixed rate. Thus, CoAP is inefficient and has undesirable poor performance in this case.

According to [2], the CoAP restricts the number of concurrent packets that can be sent without receiving ACK. Concurrent packets are defined as *inflight* packets (i.e., packets in transit in the network not yet acknowledged). As shown in [2], the CoAP limits the maximum number of outstanding interactions by a fixed value named NSTART (default is *one*). The stop-and-go mechanism of CoAP is not suitable for burst traffic. This means that CoAP does not support burst data transfer.

The second deficiency of CoAP is the lack of rate control in congestion. The sending rate depends on the arrival time of ACK packets. This means that the sending rate would be constant in the network condition without congestion. The CoAP sender adjusts only the retransmission speed based on RTO backoff in case of congestion, i.e., when congestion has occurred. The retransmission speed is halved each time because the RTO is doubled for each retransmission attempt. If the RTO value is short compared to the propagation delay between the sender and receiver, the sender may early trigger the retransmission leading to spurious retransmission and undesirable additional load for the network. Large RTO values can lead to long idle delays that cause inefficiency and poor performance of the protocol. Furthermore, fixed RTO values do not reflect the dynamic nature of the networks. The propagation delay (or RTT) frequently fluctuates depending on the load and congestion situation in the network. The current CoAP ignores the changes in RTT owing to dynamic network conditions.

In summary, CoAP does not allow reliable burst data transfer. In addition, CoAP lacks a rate control, particularly for congestion control and avoidance. The simple congestion control of the current CoAP is insufficient for burst traffic and congestion control. This paper focuses on these shortcomings and proposes a new control mechanism for CoAP.

C. Shortcomings of CoAP

In this subsection, we present related studies with respect to the issues of the current CoAP. According to our survey, modifications for CoAP can be classified into three main groups: 1) enhancements for RTO computation, 2) proposals for burst data transfer, and 3) modifications for rate control.

1) Enhancements for RTO computation

Most studies focused on RTO modifications for CoAP [5]-[10]. The reason is that a fixed RTO value is unsuitable for IoT networks because of dynamic network conditions. The authors in [5] highlighted the need for RTO adjustment according to the variability of RTT. Dynamic update of RTO helps restrict the frequency of retransmissions. If a fixed RTO value is used for connections with low bandwidth and large delay (large RTT), RTO will quickly pass without receiving an ACK. In this case, the sender immediately retransmit the previous packet, although it would receive the delayed ACK packet in a later time. The retransmitted packet is redundant in this case. Moreover, the receiver receives a duplicated packet. This issue is undesirable, leading to a waste of scarce network bandwidth and increase of delay. Therefore, the authors in [5], [6] proposed to measure the round-trip time to adjust RTO. Owing to the variation of RTT, two estimators were proposed: a "*strong RTO estimator*" and a "*weak RTO estimator*". The authors proposed a modified CoAP called CoCoA [5], [6]. In addition, CoCoA used a variable backoff factor (VBF) instead of BEB mechanism of CoAP.

Another CoAP variant, CoCoA+, was proposed in [7]. CoCoA+ used a smaller RTT multiplicative factor to reduce the impact of weak RTO estimator of CoCoA. However, the adjustment of RTO depended much on RTT and it is difficult because of the frequent variation of RTT. The authors [7] suggested a probabilistic backoff factor (PBF). However, CoCoA+ was unable to select the correct RTO value for burst traffic. The retransmission can occur quickly in many network scenarios, especially in case of small RTT and burst traffic resulting in poor performance compared to the basic CoAP in various network conditions.

As indicated in [7]-[10], choosing a right RTO is a problem in dynamic network conditions because of the variability of RTT. If the RTO value is large, the sender cannot receive further ACKs. A dynamic scaling factor was proposed in [8] for estimating RTO. In [9], the authors proposed using a fuzzy logic system to compute RTO. The RTO value was computed using a smooth RTT estimation and flexible backoff mechanism. In [10], the authors proposed several modifications to the computation of RTO. The maximum mean deviation of the RTO was computed to avoid the impact of RTT variations and limit the overall RTO value.

2) Proposals for reliable burst data transfer

Until now, few studies have addressed the problem of burst traffic. The basic CoAP does not support burst traffic. The RFC 7252 [2] indicated this limitation for further development. Recently, an other RFC was proposed for burst transfer, but in blocks [11][12]. The mechanism in [11] proposed an option for transferring large payloads in a block-wise manner. A block-wise transfer mechanism was also proposed in [12]. However, these mechanisms are only either for separating large datagrams into blocks [11] or for unreliable data transfer [12]. The mechanism in [12] focused on the issue of flow control and error handling, that is, not on congestion control.

In [13], the authors shown the issue of CoAP for burst traffic owing to the wrong computation of RTO for packets in burst. Thus, the authors introduced the retransmission counter as an option field in the packet to estimate the RTT for every packet of burst traffic. The authors in [14][15] investigated the impact of RTO for video streaming applications. The papers shown that the RTO and RTT values have significant impact on the burst transfer of streaming data.

3) Modifications of CoAP for rate control

Several studies addressed the problem of concurrent transmission together with an RTO adjustment. In [14], the authors suggested a control mechanism for CoAP based on the TCP BBR (bottleneck bandwidth round-trip propagation time) protocol. A rate-based control mechanism, that is the BDP-CoAP [16], was proposed. This mechanism estimates the bottleneck bandwidth and round-trip propagation time to estimate the new RTO. The updated RTO is used to adjust the sending rate. The remaining issue of BDP-CoAP is the overestimation of the available bandwidth, which results in inefficient performance, particularly in burst traffic. Furthermore, this method regulates only the rate based on bottleneck bandwidth and RTO estimation.

The paper in [17] proposed a rate-based approach for regulating the sending rate of CoAP sources. The adjustment of the sending rate was based on throughput estimation using the link capacity. This mechanism required the knowledge of bandwidth allocation along the

connection path. However, it is difficult to estimate the link capacity under dynamic network conditions. Wrong allocation can lead to inaccurate allocation of the transmission rate.

Another rate-based scheme was proposed in [18]. This mechanism used probe packets to discover the bottleneck bandwidth and regulated the sending rate accordingly. The scheme allows burst data transfer and was able to distinguish congestion losses from wireless losses using probe packets. However, the authors indicated the difficulty in estimating the bottleneck bandwidth for updating the RTO and sending rate.

D. Summary

In Section II, we have provided an overview of the CoAP protocol, highlighted its shortcomings, and presented several studies on the related issues. As presented, most of studies focused on the issues of RTO computation instead of using the fixed RTO similar to the basic CoAP. However, these modifications affected only the retransmission, that is, only when congestion has occurred. Therefore, it is necessary to develop a suitable control mechanism for adjusting the sending rate before congestion occurs. The sending rate must be adjusted according to dynamic network conditions. The problem of reliable burst data transfer has still not fully addressed. Because the current CoAP does not support burst traffic, a new control mechanism is necessary to solve this issue.

In next section, we present the proposed control mechanism for improving burst data transfer using CoAP in IoT networks.

III. PROPOSED CONTROL MECHANISM

A. Analytical model

In this section, we develop an analytical model for CoAP with inflight packets. The purpose of this analytical model is to find a method for control the sending rate to avoid congestion. The aim of this model is for reliable burst data transfer using CoAP.

As shown in Figure 1, the sequence of CON and ACK packets can be described by a discrete-time model. The discrete time model was commonly used to build analytical models in computer networks. Examples are the models in [19], [20]. In [19], Kleinrock analyzed the congestion control for TCP using queueing systems based on a discrete-time model. In [20], Keshav used a discrete-time model to illustrate a TCP conversation over a series of network nodes in the end-to-end path. The discrete-time model will be appropriate for any control system because the control decisions can be made in discrete-time intervals. Thus, we use the discrete-time model for CoAP transactions. However, the model for CoAP differs from the model for TCP (e.g., in [19], [20]) under various aspects. First, TCP model used congestion window to describe the throughput and delay as a function of the window. The control decision was given to increase or decrease the window size. In contrast, our CoAP model uses inflight packets and adjusts the sending rate. The CoAP model interprets the delivery rate and packet delay as a function of the inflight packets. Second, TCP model used triple ACKs as a signal for packet losses. In contrast, our CoAP considers an ACK loss as packet loss, as indicated in RFC 7252 [2]. Third, TCP model had the goal to control the window size, whereas our CoAP model aims to control the sending rate.

Figure 2 presents the sequence of sending and receiving periods between a sender and receiver in our discrete-time model for CoAP. Each period *k* denotes a round-trip time (RTT), and *T(k)* denotes the time duration of period *k*. In this figure, a CoAP sender can send several inflight packets during each period *k*. The discrete-time model is suitable to interpret the transaction between the sender and receiver because the rate adjustment will be performed in a discrete-time manner. That is, the decision on rate control is at the time of sending a CON packet.

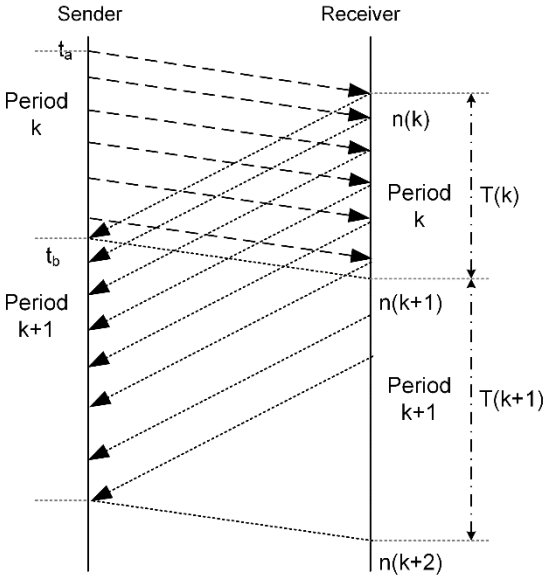

Figure 2. Periods for burst sending

Let *λ(k)* denote the sending rate during period *k*, *μ(k)* be the delivery rate at the receiver in period *k*, *T(k)* be the time duration of period *k*. The amount of data packets (the inflight packets) being transmitted in period *k* can be computed as follows:

$$L(k) = \lambda(k) \times T(k) \quad (1)$$

Among the transmitted packets *L(k)*, there are *μ(k)xT(k)* packets that have been processed by the receiver (i.e., server has received the packets and answered with ACKs). Let *n(k)* denote the instantaneous number of packets that arrived at the destination waiting for processing, the accumulative number of packets during the next period *(k+1)* will be *n(k+1)*. We have:

$$n(k+1) = n(k) + \lambda(k) \times T(k) - \mu(k) \times T(k) \quad (2)$$

From (1) and (2), we have:

$$n(k+1) = n(k) + L(k) - \mu(k) \times T(k) \quad (3)$$

From (3), we have:

$$\mu(k) = \frac{1}{T(k)}(L(k) + n(k) - n(k+1)) \quad (4)$$

Using (1) and (4), we can have:

$$\lambda(k) = \mu(k) + \frac{\Delta n}{T(k)} \quad (5)$$

Where: $\Delta n = n(k+1) - n(k)$ (6)

Δn represents the amount of increased or decreased packets between the periods, which depends on the sending rate of the sender and processing capability of the receiver. According to [20], we define a utilization factor ρ as follows:

$$\rho = \frac{\lambda}{\mu} \quad (7)$$

The system (i.e., the network) is stable if $\rho \leq 1$, this means $\mu \geq \lambda$. That is, the sending rate must be less than or equal to the delivery rate in a stable condition. In other words, the delivery rate must be greater than or equal to the sending rate to avoid congestion. This means that the minimal delivery rate $\mu(k)$ at step k must be equal to the sending rate $\lambda(k-1)$ at step $k-1$. We can rewrite (5) as follows:

$$\lambda(k) = \lambda(k-1) + \frac{\Delta n}{T(k)} \quad (8)$$

The quotient $\frac{\Delta n}{T(k)}$ in (8) represents an amount of increased or decreased packets at each period *k*. The smallest increase will be one if *Δn* is equal to *one*. If we do not want to make the control more aggressive, we can choose the value *Δn* = 1 for the increase of sending rate in case of no congestion. Thus, we can rewrite (8) as follows:

$$\lambda(k) = \lambda(k-1) + \frac{1}{T(k)} \quad (9)$$

The increase of *one* packet per *T(k)* is reasonable owing to the possible large amount of inflight packets at this moment.

On the other hand, Jain [21] showed a possibility to describe the network as a black box. The senders treat the network as a black box and interact with the receiver using requests and responses. In [20], Kleinrock showed that it is possible to model the connection from the senders to receiver in form of a physical pipe. The diameter of the pipe describes the maximum bottleneck bandwidth for all the flows. The pipe length describes the propagation delay. Intuitively, the delivery rate must be less than or equal to the maximum bottleneck bandwidth to avoid congestion.

Assume that the pipe can be described in a cartesian coordinate system where the x-axis represents the propagation delay, y-axis represents the diameter of the pipe. Let Y denote the portion of the diameter used by a CoAP flow, X be the propagation delay of the flow. The product of X and Y will represent the allowed inflight packets for such flow. Thus, we can define a function to represent the number of inflight packets for each flow. Because of the non-linear characteristics of the parameters, we must use an exponential function. Using the exponential function is the best way to model a non-linear variable. We define an utility function U(L) for inflight packets as follows:

$$U(L) = \mu(L)^{-\alpha} T(L) \quad (10)$$

where $\mu(L)$ is a function of L representing the delivery rate at the receiver, L is the number of inflight packets, T(L) is the delay function of L, and α is a control factor, $\alpha > 0$.

The utility function U(L) represents the relationship between the delivery rate and packet delay with the variable L (that is the inflight packets). The delivery rate is defined by the ratio of the packet number received at the destination and a time unit. This ratio corresponds to the receiving rate of the flow.

From (10), we have

$$\log(U(L)) = \log(T(L)) - \alpha\log(\mu(L) \quad (11)$$

We take the differential for both sides to obtain:

$$\frac{dU(L)}{U(L)} = \frac{dT(L)}{T(L)} - \alpha\frac{d\mu(L)}{\mu(L)} \quad (12)$$

The utility function U(L) will be maximum if its derivative is equal to *zero*. That is,

$$\frac{dU(L)}{U(L)} = \frac{dT(L)}{T(L)} - \alpha\frac{d\mu(L)}{\mu(L)} = 0 \quad (13)$$

Thus,

$$\alpha\frac{d\mu(L)}{\mu(L)} = \frac{dT(L)}{T(L)} \quad (14)$$

The quotient $\frac{d\mu(L)}{\mu(L)}$ represents the relative variation of the delivery rate, whereas the quotient $\frac{dT(L)}{T(L)}$ represents the relative variation of the packet delay with the number of inflight packets *L*. The value α represents the relative variation ratio of both presented quantities.

The utility function increases with the delivery rate and packet delay. This function reaches a maximum at a point described by (14) according to the number of inflight packets. Subsequently, the function decreases. This is the case of congestion when the number of inflight packets becomes too large. The goal of control is to limit the number of inflight packets just before a maximum point of the utility function. The meaning of control factor α is as follows:

- If $\alpha < 1$, the increase speed of the delay variation is faster than the increase speed of the delivery rate variation. The target of control will be in the direction of lower delay.

- If $\alpha > 1$, the increase speed of the delay variation is slower than the increase speed of the delivery rate variation. The target of control will be in the direction of higher delivery rate.

- If $\alpha = 1$, the packet delay increases corresponds to the delivery rate. The target of control is to maintain the balance between delivery rate and packet delay.

Let B(L) denote the number of inflight packets at the end of the period k. We consider two cases: 1) the case without packet loss, and 2) the case with packet loss.

In the case of no packet loss, all transmitted packets L will arrive at the destination within period k. The number of received packets will be B(L). At the maximum point of U(L), we can determine the delivery rate $\mu(L)$ as follows:

$$\mu(L) = \frac{L}{T(L)} \quad (15)$$

Thus, from (14), we can have

$$\alpha\frac{d\mu(L)}{dL}\frac{dL}{\mu(L)} = \frac{dT(L)}{T(L)} \quad (16)$$

$$\alpha \frac{1}{T(L)} \frac{dL}{(L/T(L))} = \frac{dT(L)}{T(L)} \quad (17)$$

$$\alpha \frac{dL}{L} = \frac{dT(L)}{T(L)} \quad (18)$$

$$L = \alpha T(L) \frac{dL}{dT(L)} \quad (19)$$

Owing to the assumption that no packet loss happens, from (1), we can deduce that

$$\frac{dL}{dT(L)} = \lambda(L) \quad (20)$$

Thus, from (19) and (20), we can obtain

$$B(L) = L = \alpha \times T(L) \times \lambda(L) \quad (21)$$

Equation (21) indicates an amount of inflight packets B(L) at the maximum point of the utility function P(L) in case of no packet loss.

Now, we consider the case of packet loss. Suppose that a packet loss occurs owing to congestion during period k. Let B(L) denote the number of inflight packets at a maximum of the utility function P(L) in case of packet loss. The delivery rate $\mu(L)$ at the time of packet loss can be determined as follows:

$$\mu(L) = \frac{B(L)}{T(L)} \quad (22)$$

By substituting $\mu(L)$ into (11), we have

$$\log(U(L)) = (1+\alpha)\log(T(L)) - \alpha\log(B(L)) \quad (23)$$

Again, the utility function U(L) will be maximum if its derivative is equal to *zero*. That is,

$$\frac{dU(L)}{U(L)} = (1+\alpha)\frac{dT(L)}{T(L)} - \alpha\frac{dB(L)}{B(L)} = 0 \quad (24)$$

Thus, we can compute B(L) as follows:

$$B(L) = \frac{\alpha}{(1+\alpha)} T(L) \frac{dB(L)}{dT(L)} \quad (25)$$

As presented previously, the sending rate must be less than or equal to the delivery rate. Therefore, the maximal sending rate just before the packet loss can be determined as follows:

$$\lambda(L) = \frac{dB(L)}{dT(L)} \quad (26)$$

By substituting (26) into (25), we get

$$B(L) = \frac{\alpha}{(1+\alpha)} T(L) \lambda(L) \quad (27)$$

By comparing (27) and (21) we can conclude that B(L) in case of packet loss is less than B(L) in case of no packet loss by a factor of $\frac{\alpha}{(1+\alpha)}$ at the maximum utility function U(L). If we choose $\alpha = 1$, we have

$$\frac{\alpha}{(1+\alpha)} = 0.5 \quad (28)$$

That is, B(L) in case of packet loss is a half of B(L) in case of no packet loss at the maximum of utility function. This means that the sending rate must be adjusted to maintain a half of inflight packets to obtain the maximum for the utility function in case of packet loss. Because the number of inflight packets is the same before and after packet loss, the sending rate must be reduced to a half in case of packet loss.

In conclusion, we can have the following control mechanism:

- In case of no packet loss, the CoAP sender can increase the sending rate by *one* as follows:

$$\lambda(k) = \lambda(k-1) + \frac{1}{T(k)} \quad (29)$$

- In case of packet loss, i.e., when congestion occurs, the CoAP sender must decrease the sending rate by half as follows:

$$\lambda(k) = 0.5 \times \lambda(k-1) \quad (30)$$

where $\lambda(k)$ is the sending rate at step k, $\lambda(k-1)$ is the sending rate at the previous step $k-1$, $T(k)$ is a round-trip time that is measured at step k, and k is the time when the sender receives an ACK.

The expressions (29) and (30) present the proposed rate control mechanism for CoAP in this paper.

B. A rate control mechanism for CoAP

We propose a rate control mechanism for CoAP based on the analytical model developed in the previous section. The control mechanism is mainly implemented at the CoAP sender. On the CoAP receiver, there are only functions for receiving CON packets and sending ACK packets to the senders.

The operation of this control mechanism is described using *four* states in the next paragraphs. The pseudocodes for the key algorithms in the states are presented in the figures 3, 4, 5, and 6 in this subsection.

We use the following notations.

- R_{start}: the start sending rate of a CoAP sender during the startup state.

- R: the computed sending rate for the CoAP sender.

- *nACK*: the number of received packets during the startup state.

- *RTT*: the measured round-trip time.

- *T*: the time to send the next packet.

The function SendNextPacket is for packet sending. In case of packet loss, this function first attempts to retransmit the lost packets before sending another packet.

1) Startup state

At the startup, the CoAP sender uses a default start rate R_{start}. This start rate is only valid during two RTTs. Subsequently, it is replaced by the computed sending rate in the steady state.

As indicated in Figure 3, the sender transmits packets within the first while loop. The inter-packet interval is T. When the first ACK is received, the sender measures RTT and performs the second while loop to count the received ACKs for the sent packets.

After two RTTs, the sender computes the sending rate as follows:

$$R = min(R_{start}, max(1, nACK) / 2*RTT) \quad (31)$$

The sender then enters the steady state.

```
1: Function Startup(R_start)
2:     State = Startup
3:     T = t_0 + 1/R_start
4:     nACK = 0
5:     ACK = false
6:     while ( ACK == false ) do
7:         if (( t >= T ) & ( t <= T_max ) ) then
8:             packet ← SendNextPacket( )
9:             T = t + 1/R_start   ; t = now
10:        else if ( t > T_max ) then
11:            TransactionFailed( )
12:            return
13:        endif
14:    enddo
15:    RTT ← CalculateRTT( )
16:    while ( t <= 2*RTT ) do
17:        if (ACK == true) then
18:            nACK = nACK + 1
19:        endif
20:    enddo
21:    R = min (R_start, max ( 1, nACK ) / 2*RTT )
22:    State = Steady
23: end Function
```

Figure 3. Startup State

2) Steady state

Figure 4 presents the steady state. This state starts with a sending rate *R* computed at the previous step. If there is no packet loss, the sender checks the time to send the next packet. If any ACK is received, RTT is updated accordingly. Otherwise, packet loss will be checked.

Packet loss can be detected in two cases: 1) If the sender does not receive an ACK after timeout (RTO), it assumes packet loss owing to congestion; 2) If the sender detects a gap in the packet sequence numbers, several packets have been lost. The gap may include retransmitted packets that were not successful after the maximal number of retransmissions. The function CheckLoss is used for packet loss detection.

If the sender detects packet loss, it changes to the detect state. If no packet loss is detected, the sender checks for the time to adjust the sending rate. The rate updating period is one RTT. The sending rate *R* is adjusted using (29).

3) Detect state

Figure 5 presents the detect state. The CoAP sender enters this state if packet loss is detected, i.e., congestion occurs. The CoAP sender immediately reduces the sending rate by a half using (30). Then, the sender performs a while loop for a time of one RTT. Within this loop, it first checks for the time to send the next packet. As indicated above, this function retries to retransmit the lost packets before sending the next packet. If any ACK is received within one RTT, the sender recovers the previous sending rate and return to the steady state. Otherwise, the sender assumes that the network is still congested. A heavy congestion situation has occurred. Thus, the sender changes to the backoff state.

```
1: Function Steady(R)
2:     State = Steady
3:     T = t_0 + 1 / R
4:     T_n = t_0 + RTT
5:     ACK = false
6:     Loss = false
7:     while ( Loss == false ) do
8:         if ( t >= T ) then
9:             packet ← SendNextPacket( )
10:            T = t + 1 / R
11:        endif
12:        if (ACK == true) then
13:            RTT = CalculateRTT( )
14:        else
15:            Loss = CheckLoss( )
16:            if ( Loss == true ) then
17:                State = Detect
18:            endif
19:            return
20:        endif
21:        if ( t >= T_n ) then
22:            R = min ( R + 1 / RTT, R_max )
23:            T = t + 1 / R
24:            T_n = t + RTT
25:        endif
26:        t = now
27:        Loss = CheckLoss()
28:    enddo
29: end Function
```

Figure 4. Steady State

```
1: Function Detect(R)
2:     State = Detect
3:     R_old = R
4:     R_new = R / 2
5:     T = t_0 + 1 / R
6:     T_n = t_0 + RTT
7:     ACK = false
8:     while ( t <= T_n ) do
9:         if ( t >= T ) then
10:            packet ← SendNextPacket()
11:            T = t + 1 / R
12:        endif
13:        if (ACK == true) then
14:            RTT = CalculateRTT()
15:            R = R_old
16:            Loss == false
17:            State = Steady
18:            ACK = false
19:            return
20:        endif
21:        t = now
22:    enddo
23:    if (ACK == false) then
24:        Loss == true
25:        State = Backoff
26:        return
27:    endif
28: end Function
```

Figure 5. Detect State

4) Backoff state

Figure 6 presents the backoff state. During this state, the sender continues to check for the receiving ACK within a while loop. In this loop, the sender first checks for the time to send the next packet. Again, the function SendNextPacket tries to retransmit the lost packets before sending another packet. If no ACK is received within the maximum transaction time as defined in [2], the transaction is considered to have failed. The sender then is required to restart. If any ACK is received, the sender updates RTT and returns to the steady state. Otherwise, the sender reduces the sending rate again by a half to perform a slower transmission.

```
1: Function Backoff
2:     State = Backoff
3:     R_new = R / 2
4:     T = t_0 + 1 / R
5:     T_n = t_0 + RTT
6:     ACK = false
7:     while ( ACK == false ) do
8:         if ( t >= T ) & ( t <= T_max ) then
9:             packet ← SendNextPacket( )
10:            T = t + 1 / R
11:        else if ( t > T_max ) then
12:            TransactionFailed( )
13:            return Startup( )
14:        endif
15:        if (ACK == true) then
16:            RTT ← CalculateRTT( )
17:            State = Steady
18:            return Steady()
19:        endif
20:        if ( t >= T_n ) then
21:            R_new = R / 2
22:            T = t + 1 / R
23:            T_n = t + RTT
24:        endif
25:    t = now
26:    enddo
27: end Function
```

Figure 6. Backoff State

IV. SIMULATION RESULTS

In this section, we present the simulation results for the proposed control mechanism using the Network Simulator NS-3.36 [22]. For the convenience, the proposed mechanism is called mCoAP. All simulation scenarios used a star network topology, as shown in Figure 7.

We used 10 flows for CoAP and 10 flows for mCoAP. All CoAP senders were implemented at the wireless nodes of a Wi-Fi network. This Wi-Fi network used a base station (BS) and was connected to the Internet through a gateway. All senders are connected to a central server in Internet. The Wi-Fi network was established using the standard parameters of IEEE 802.11 in NS-3 [22]. The bottleneck link (link between the BS and the gateway) had a bandwidth of 250 Kbps with link delay of 64 ms. The link bandwidth between the gateway and server was 1 Mbps with link delay of 70 ms. These parameters were used to create dynamic congestion situation in the experiments. Other values for bandwidth and delay may be possible. However, the purpose of these experiments was to show the feasibility of the proposed control mechanism and compare with the original CoAP. Thus, such parameters are sufficient to simulate various congestion scenarios.

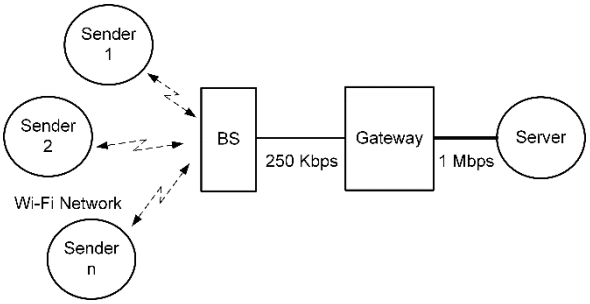

Figure 7. Simulation Model

The proposed control mechanism mCoAP was compared to the original CoAP using the following metrics: delay, throughput, number of sent packets, number of ACKs, number of packet losses, number of retransmissions, and number of duplications. All measured values were computed using average values for all *ten* flows. The simulation time was 300 seconds for all experiments. We conducted *ten* times for each experiment to perform confident measurements.

Figure 8 shows the average delay comparison for mCoAP and CoAP. As indicated, congestion occurred during the interval from 170 s to 290 s owing to the accumulated number of transmitted packets from *ten* flows. CoAP had low delay (around 900 ms) during 0s to 160 s. Subsequently, the delay in CoAP was increased rapidly. Packet delay was approximately 30 s during 170 s and 250 s. The reason is that many packets cannot be received at server because of timeout. Thus, CoAP retransmitted these packets using doubled RTO at each retransmission attempt (see Table 1 for the number of retransmissions).

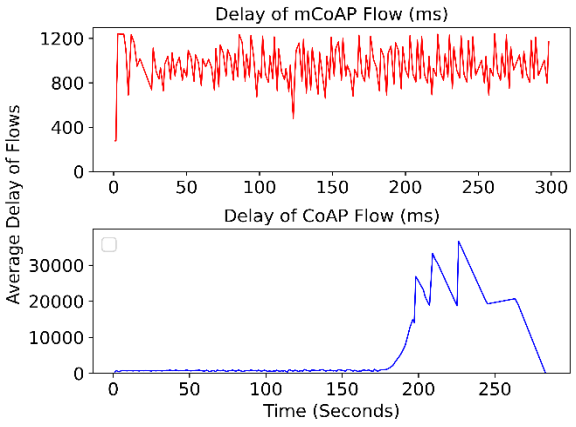

Figure 8. Average Delay Comparison

In contrast, mCoAP adjusted a suitable sending rate, thus, no packet was required to be retransmitted. Therefore, mCoAP can maintain a low average delay of approximately 790 ms to 1200 ms, as indicated in the figure. These results indicated that mCoAP controlled congestion better than CoAP. Thus, mCoAP can maintain a lower delay.

Figure 9 shows an average throughput comparison of 10 mCoAP flows and 10 CoAP flows. As indicated, mCoAP and CoAP had approximately the same average throughput during the time from 0s to 170 s, where the network was still not congested. In the time of congestion (from 170 s to 290 s), the throughput of CoAP decreased quickly. Meanwhile, mCoAP still retained a high average throughput of approximately 0.711 Kbps (for each flow). The throughput peak during 0 to 20 s indicates that mCoAP started with a high sending rate to estimate the bottleneck bandwidth during the startup state. These results indicated that mCoAP has better throughput performance than that of the CoAP in case of congestion.

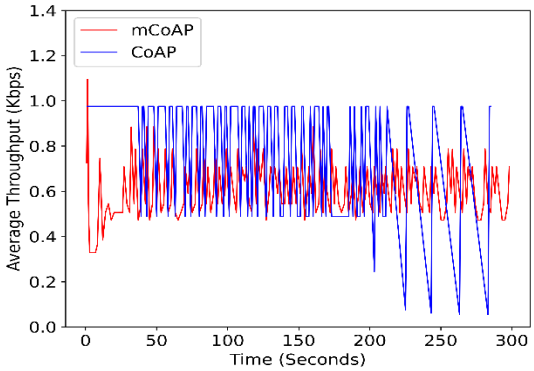

Figure 9. Average Throughput Comparison

Figure 10 shows the average delay for the *three* CoAP flows (randomly selected from 10 flows). The results indicated that the average delay was approximately the same for all CoAP flows. High delay was during congestion (from 170 s to 290 s).

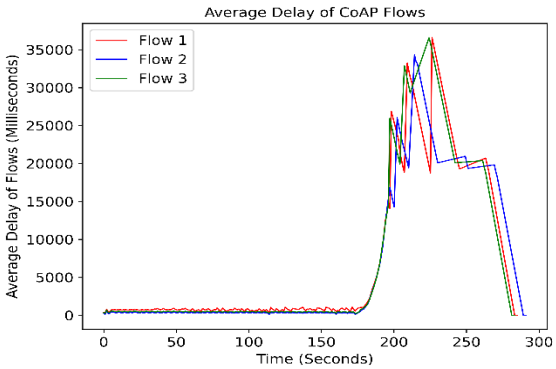

Figure 10. Average Delay of three CoAP Flows

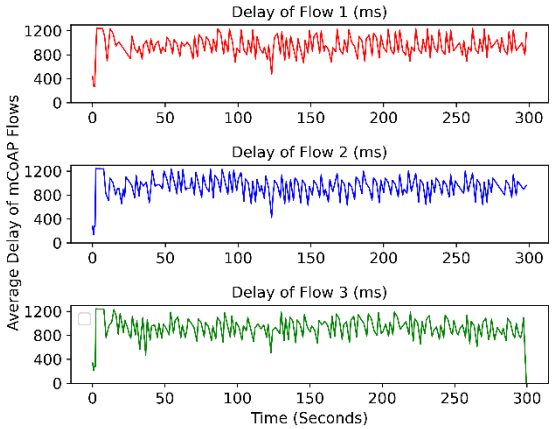

Figure 11. Average Delay of three mCoAP Flows

Figure 11 shows the average delay for *three* mCoAP flows (randomly selected from 10 flows). The results indicated that the average delay was approximately equivalent for all mCoAP flows. A delay variation was approximately 920 ms, even in the time interval of congestion (from 170 s to 290 s).

Table I shows a performance evaluation of mCoAP and CoAP. The results indicated that the number of sent packets, number of ACKs and retransmitted packets, number of received packets with ACK, and number of successful received packets in mCoAP were higher than those in CoAP. Note that the number of successful received packets is defined as the difference between the sent packets and duplicated retransmitted packets. The number of packet losses is the sum of lost packet and duplicated packets because the duplicated packets are useless and will be discarded.

As shown in Table I, the average delay of 10 mCoAP flows was 922.41 ms that was less than 4175.40 ms of 10 CoAP flows. The average throughput in mCoAP was 0.7111 Kbps that was higher than 0.5333 Kbps of CoAP in the same competing condition and bottleneck bandwidth in these experiments.

Table I. Performance evalution of mCoAP and CoAP

Average for 10 flows	**mCoAP**	CoAP
Number of sent packets	219	163
Number of ACKs and retransmitted packets	216	162
Number of retransmitted packets	0 (0,00%)	33 (20,53%)
Number of duplicated packets	0 (0,00%)	29 (17,93%)
Number of received packets with ACK	216 (100,00%)	159 (98,21%)
Successful received packets	216 (100,00%)	134 (82,20%)
Packet Losses	0 (0,00%)	32 (19,73%)
Average delay	922,41 ms	4175,40 ms
Average throughput	0,7111 Kbps	0,5353 Kbps

We changed simulation condition to create a heavy congestion situation. In these simulation scenarios, the link bandwidth between the gateway and the server was 1 Mbps, but the link delay was 120 ms. This link delay was double compared to the previous simulation scenarios. The round-trip delay was larger than the previous resulting in higher likelihood of congestion. Under this condition, we indicated a heavy congestion situation for all flows competing the common bottleneck bandwidth.

Figure 12 shows the results of delay comparison of mCoAP and CoAP using these simulation scenarios. As indicated, the flows faced to congestion immediately at the startup. The delay increased rapidly in CoAP. In contrast, mCoAP can bound the delay during a certain time interval, that is, from 0s to 100 s. Subsequently, the delay increased quickly in mCoAP owing to heavy congestion. However, the results indicated that mCoAP can handle heavy congestion better than CoAP.

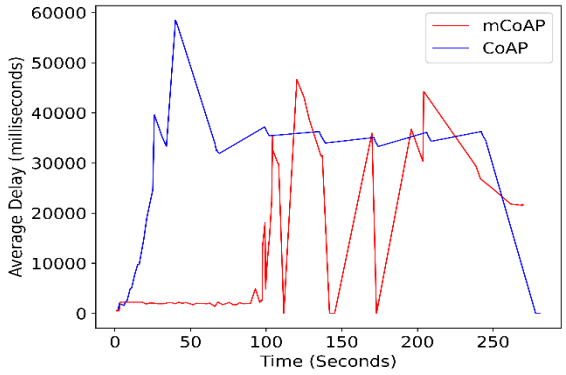

Figure 12. Average Delay in Heavy Congestion

Figure 13 presents an average throughput comparison of mCoAP and CoAP under heavy congestion. Owing to the variation of bottleneck bandwidth, the throughput of both mCoAP and CoAP fluctuated. Nevertheless, the fluctuation was approximately a baseline that was less than 1 Kbps. For mCoAP, there was some peaks because mCoAP detected packet losses and changed the operation state. At the moment of state change, mCoAP tried to keep the highest throughput as possible. Thus, the peaks represented the quotient of high number of inflight packets and the short change time duration.

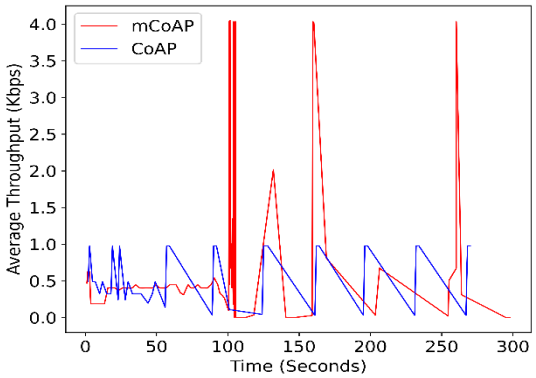

Figure 13. Average Throughput in Heavy Congestion

Figure 14 shows the average delay of three CoAP flows (randomly selected from 10 flows). The results indicated that flows had high equivalent packet delay. The high delay values were because of the RTO backoff mechanism of CoAP. The initial RTO of 2000 ms was doubled for each retransmission.

Figure 15 shows the average delay of three mCoAP flows (randomly selected from 10 flows). As indicated in the figure, the average delay of flows was small for all flows during the first-time interval from 0s to 100 s. The reason is that mCoAP tried to adjust the rate to mitigate congestion as explained above. Thus, mCoAP can bound the small delay during a certain interval. Owing to heavy congestion, the number inflight packets increased accumulatively. Therefore, the delay increased quickly when congestion became more serious. Again, the delay values in mCoAP were high owing to the RTO backoff as same as in CoAP.

Table II presents a performance evaluation of mCoAP and CoAP under heavy congestion condition. The comparison metrics are the number of sent packets, number of ACKs and retransmitted packets, number of retransmitted packets, number of duplicated packets, number of received packets with ACK, number of successful received packets, average delay, and average throughput.

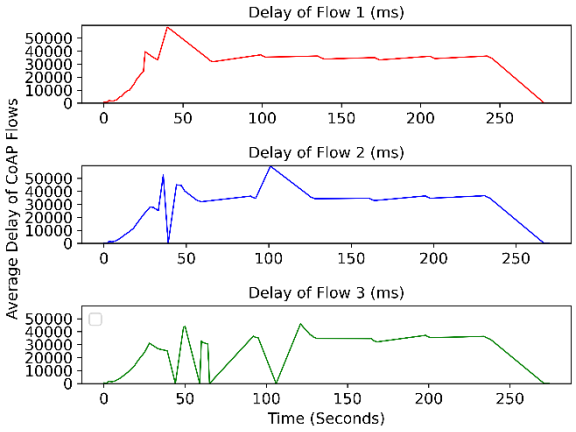

Figure 14. Average Delay of three CoAP Flows in Heavy Congestion

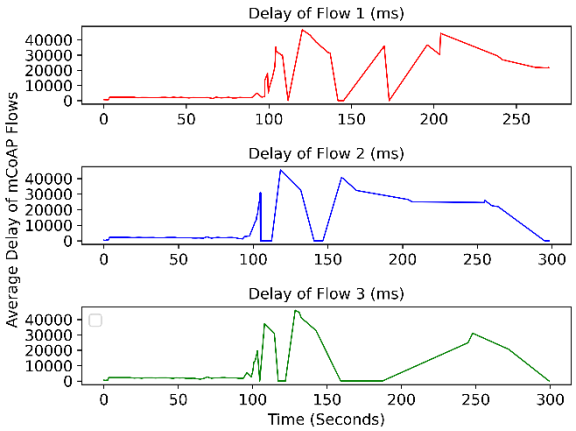

Figure 15. Average Delay of three mCoAP Flows in Heavy Congestion

Table II. Performance evalution of mCoAP and CoAP in heavy congestion

Averages for 10 Flows	**mCoAP**	CoAP
Number of sent packets	77	51
Number of ACKs and retransmitted packets	74	49
Number of retransmitted packets	35 (47,24%)	48 (97,55%)
Number of duplicated packets	27 (36,88%)	40 (82,62%)
Number of received packets with ACK	63 (85,33%)	44 (89,57%)
Successful received packets	36 (48,45%)	4 (6,95%)
Packet Losses	38 (51,55%)	45 (93,05%)
Average delay	10902,82 ms	30665,06 ms
Average throughput	0,2459 Kbps	0,1519 Kbps

As shown in Table II, the number of sent packets, number of ACKs and retransmitted packets, number of

received packets with ACK, and number of successful received packets of mCoAP were higher than those of CoAP. The number of duplicated packets of 27 (36.88%) in mCoAP was less than 40 (82.62%) in CoAP. The number of packet losses of mCoAP was 38 (51.55%), whereas it was 45 (93.05%) in CoAP. The average delay in mCoAP flows was 10902.82 ms, which was lower than 30665.06 ms in CoAP. The average throughput in mCoAP was 0.2459 Kbps that was higher than 0.1519 Kbps of CoAP under the same competing and network condition.

Note that, we defined the number of successful received packets as the difference of the number of sent packets and number of duplicated retransmitted packets. The number of packet losses was defined as the sum of lost packets and duplicated packets because the duplicated packets are useless and will be discarded.

Summary: Section IV presents the simulation results using two simulation scenarios: light congestion and heavy congestion. The results indicated that the proposed mechanism, that is the mCoAP, is feasible. In addition, mCoAP can process congestion better than the original CoAP under various dynamic network conditions.

V. CONCLUSION

As presented in this paper, a reliable burst data transfer is typically required for many applications in IoT networks. Without proper control mechanism, the transfer of such burst data may lead to congestion in the network. Congestion causes large packet delay, high packet loss rate, low throughput, and a lot of duplicated retransmissions. Although CoAP has been standardized for data transmission in IoT networks, it still has several shortcomings because of the simple design as a lightweight protocol for IoT applications. As indicated in [2], an enhancement for CoAP is encouraged. The development of a suitable control mechanism for reliable burst data transfer with CoAP is necessary.

In this paper, we briefly reviewed the design of CoAP, presented its shortcomings and related work. Particularly, we identified two key issues of the original CoAP, including: the lack of support for reliable burst traffic, and the lack of rate adjustment mechanism. That is why CoAP required a high number of retransmissions and duplicated retransmissions in case of burst traffic. The CoAP sources shown a large delay and poor performance for burst data transfers.

Subsequently, we proposed a new analytical model for CoAP using burst traffic. We developed a rate control mechanism based on this model for CoAP to support reliable burst data transfer in IoT networks. Two groups of simulation scenarios were proposed: light congestion and heavy congestion. The simulation results indicated that the design of a CoAP rate control mechanism for burst traffic is feasible. The proposed mechanism can process congestion better than the original CoAP under various dynamic network conditions. In addition, the proposed control mechanism outperformed the CoAP in terms of delay, throughput, retransmission, duplication, packet loss, and number of successful received packets.

Further studies can investigate the bottleneck bandwidth and possibility to determine the bottleneck to detect congestion early under dynamic network conditions.

REFERENCES

[1] C. Gomez, A. Archia-Moret, J. Crowcroft, et.al., "TCP in the Internet of Things: from ostracism to prominence," IEEE Internet Computing, vol. 22, Issue 1, pp. 29-41, Jan./Feb. 2018.

[2] RFC 7252, "The Constrained Application Protocol (CoAP)," available: https://rfc-editor.org/info/rfc7252.

[3] M.A. Tariq, M. Khan, M.T.R. Khan, D. Kim, "Enhancements and Challenges in CoAP–A Survey," Sensors, vol. 20, 2020, 6391. DOI: 10.3390/s20216391, pp. 1-29, Nov. 2020.

[4] H. Haile, K. Grinnemo, S. Ferlin, et.al., "End-to-end congestion control approaches for high throughput and low delay in 4G/5G cellular networks," Computer Networks, vol. 186, Feb. 2021.

[5] A. Betzler, C. Gomez, I. Demirkol, J. Paradells, "CoAP congestion control for the internet of things," IEEE Commun. Mag., vol. 54, no. 7, pp. 154–160, Jul. 2016.

[6] C. Bormann, A. Betzler, C. Gomez, and I. Demirkol, "CoAP Simple Congestion Control/Advanced", Internet-Draft, Feb. 2018. Accessed: Jul. 24, 2021 [Online]. Available: https://tools.ietf.org/id/draft-bormann-core-cocoa-03.txt.

[7] A. Betzler, C. Gomez, I. Demirkol, J. Paradells, "CoCoA+: An advanced congestion control mechanism for CoAP,". Ad Hoc Netw., vol. 33, pp. 126–139, Oct. 2015.

[8] S. Deshmukh, V.T. Raisinghani, "AdCoCoA–Adaptive Congestion Control Algorithm for CoAP," in Proc. of 11th IEEE Int. Conf. on Computing, Communication and Networking Technologies (ICCCNT), Kharagpur, India, Jul. 2020, pp. 1-7.

[9] P. Aimtongkham, P. Horkaew, C. So-In, "An Enhanced CoAP Scheme Using Fuzzy Logic with Adaptive Timeout for IoT Congestion Control," IEEE Access, vol. 9, pp.58967-58981, Apr. 2021.

[10] S. Bolettieri, G. Tanganelli, C. Vallati, E. Mingozzi, "pCoCoA: A precise congestion control algorithm for CoAP," Ad hoc Network, vol. 80, pp.116-139, Nov. 2018.

[11] C. Bormann, Z. Shelby, "Block–Wise Transfers in the Constrained Application Protocol (CoAP)," Accessed: Jun. 24, 2021 [Online]. Available: https://rfc-editor.org/info/rfc7959.

[12] M. Boucadair, J. Shallow, "Constrained Application Protocol (CoAP) Block-Wise Transfer Options Supporting Robust Transmission," Internet-Draft, May 2021. Accessed: Jun. 20, 2021 [Online]. https://tools.ietf.org/id/draft-ietf-core-new-block-14.

[13] J.J. Lee, K.T. Kim, H.Y. Youn, "Enhancement of congestion control of Constrained Application Protocol/Congestion Control/Advanced for Internet of Things environment," Int. J. of Distributed Sensor Networks, vol. 12 (11), pp. 1-13, Nov. 2016.

[14] W.U. Rahman, Y.S. Choi, K. Chung, "Performance Evaluation of Video Streaming Application Over CoAP in IoT," IEEE Access, vol. 9, pp.39852-39861, Apr. 2019.

[15] J.H. Jung, M. Gohar, S.J. Koh, "CoAP–Based Streaming Control for IoT Applications," Electronics, vol. 9 (8), 2020, 1320, DOI: 10.3390/electronics9081320, pp. 2-19, Aug. 2020.

[16] E. Ancillotti, R. Bruno, "BDP–CoAP: Leveraging Bandwidth-Delay Product for Congestion Control in CoAP," in Proc. of 5th IEEE World Forum on Internet of Things (WF-IoT), Ireland, Apr. 2019, pp. 656-661.

[17] E. Ancillotti, R. Bruno, C. Vallati, E. Mingozzi, "Design and Evaluation of a Rate–Based Congestion Control Mechanism in CoAP for IoT Applications," in Proc. 19th IEEE Int. Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM), Greece, Jun. 2018, pp. 14–15.

[18] D.H. Hoang, T.T.D. Le, "RCOAP: A Rate Control Scheme for Reliable Bursty Data Transfer in IoT Networks," IEEE Access, vol. 9, 2021, doi: 10.1109/ACCESS.2021.3135435, pp. 169281-169298.

[19] L. Kleinrock, "Internet congestion control using the power metric: Keep the pipe justfull, but no fuller," Ad Hoc Networks, (2018), 05-015, pp.1-16.

[20] S. Keshav, "A Control-theoretic Approach to Flow Control," in ACM SIGCOMM, Computer Communication Review, Vol.21, Issue 4, Sept. 1991, pp 3–15 (https://doi.org/10.1145/115994.115995).

[21] R. Jain, "A delay-based approach for congestion avoidance in interconnected heterogeneous computer networks," CM SIGCOMM Computer Communication Review, Volume 19, Issue 5, Oct. 1989, pp 56–71, (https://doi.org/10.1145/74681.74686).

[22] NS-3 Network Simulator, version 3.36, available: https://www.nsnam.org/

MỘT CƠ CHẾ ĐIỀU KHIỂN TRUYỀN CHÙM DỮ LIỆU TIN CẬY TRONG MẠNG IoT

Tóm tắt: Mạng IoT đang được ứng dụng rộng rãi trong nhiều lĩnh vực như công nghiệp, y tế, nông nghiệp, môi trường. Nhiều ứng dụng đòi hỏi thường xuyên truyền một lượng lớn dữ liệu đã thu thập từ các thiết bị IoT để gửi về một máy chủ trung tâm. Nếu không có cơ chế điều khiển phù hợp, mạng sẽ rất dễ xảy ra tắc nghẽn. Giao thức CoAP (Constrained Application Protocol) đã được đề xuất để truyền dữ liệu trong mạng IoT. Bài báo này phân tích các hạn chế của CoAP và chỉ ra CoAP chưa có cơ chế điều khiển tốc độ phát và không hỗ trợ truyền chùm dữ liệu tin cậy. Để cải tiến CoAP, chúng tôi xây dựng một mô hình giải tích cho truyền chùm dữ liệu tin cậy với CoAP có sử dụng điều khiển tốc độ. Dựa vào mô hình đã xây dựng, bài báo đề xuất một cơ chế điều khiển tốc độ mới cho phép truyền chùm dữ liệu tin cậy với thông lượng cao, độ trễ thấp và cải thiện việc điều khiển chống tắc nghẽn cho CoAP trong mạng IoT.

Từ khóa: Điều khiển tốc độ, truyền chùm dữ liệu, điều khiển chống tắc nghẽn, giao thức CoAP, mạng IoT.

Le Thi Thuy Duong, received the B.S. degree in telecommunications and electronic engineering from Technical University of Hanoi, Vietnam, in 2002, and the M.S. degree from Technical University of Hanoi, Vietnam, in 2008. Since 2005, she has been a lecturer with the Faculty of Information Technology at University of Civil Engineering, Hanoi, Vietnam. Currently, she is a senior lecturer. She is pursuing the Ph.D. degree at Posts and Telecommunications Institute of Technology, Hanoi, Vietnam. Her current research interests include computer and communication systems, wireless sensor networks, QoS mechanisms, and network performance.

Hoang Dang Hai, received the Diplom Ing. (M.Eng.) degree in technical cybernetics and automation from Technical University of Ilmenau, Germany, in 1984. He received the Dr.-Ing. degree and the Dr.-Ing.habil. degree in telematics and communication systems from Technical University of Ilmenau, Germany, in 1999 and in 2002, respectively. Since 2009, he has been an Associate Professor at Posts and Telecommunications Institute of Technology, Hanoi, Vietnam. His current research interests include information security, communication protocols, communication systems, QoS mechanisms, and control systems.

Pham Thieu Nga, received the Diplom Ing. (M.Eng.) degree in technical cybernetics and automation in 1987, and the Dr.-Ing. degree in Informatics and Automation in 2000 from Technical University of Ilmenau, Germany. Since 2006, she has been a Senior Lecturer at the Faculty of Information Technology at University of Civil Engineering, Hanoi, Vietnam. Her current research interests include fuzzy control, fuzzy optimization, fuzzy decision, expert systems, wireless sensor networks, IoT networks, system techniques, and control systems.