



## mlCoCoA: a machine learning-based congestion control for CoAP

Alper Kamil DEMİR<sup>1,\*</sup> , Fatih ABUT<sup>2</sup> 

<sup>1</sup>Department of Computer Engineering, Faculty of Engineering,

Adana Alparslan Türkeş Science and Technology University, Adana, Turkey

<sup>2</sup>Department of Computer Engineering, Faculty of Engineering, Çukurova University, Adana, Turkey

Received: 03.03.2020

Accepted/Published Online: 31.05.2020

Final Version: 25.09.2020

**Abstract:** Internet of Things (IoT) is a technological invention that has the potential to impact on how we live and how we work by connecting any device to the Internet. Consequently, a vast amount of novel applications will enhance our lives. Internet Engineering Task Force (IETF) standardized the Constrained Application Protocol (CoAP) to accommodate the application layer and network congestion needs of such IoT networks. CoAP is designed to be very simple where it employs a genuine congestion control (CC) mechanism, named as default CoAP CC leveraging basic binary exponential backoff. Yet efficient, default CoAP CC does not always utilize the network dynamics the best. As a result, CoCoA has been exposed to better utilize the IoT networks. Although CoCoA considers the network dynamics, the RTO calculation of CoCoA is based on constant coefficient values. However, our experiments show that these constant values, in general, do not achieve the best throughput. Inspired by these observations, we propose a new machine learning-based CC mechanism called as mlCoCoA that is a variation of CoCoA. Particularly, mlCoCoA sets retransmission timeout (RTO) estimation parameters of CoCoA adaptively by using a machine learning method. In this study, we applied support vector machines on a self-created dataset to develop new models for improving the throughput of the IoT network with dynamic selection of CoCoA coefficient values. We carried out extensive simulations in Cooja environment coupled with Californium. Our results indicate that compared to the performance of default CoAP CC and CoCoA mechanisms, mlCoCoA has merit in terms of improving the throughput of CoAP applications.

**Key words:** Internet of Things, CoAP, CoCoA, congestion control, machine learning

### 1. Introduction

The Internet of Things (IoT) is a vision of the next-generation internet widened with networks of constrained devices, embedded with small-sized and low power electronics, sensors, actuators, communication units, software, and network protocols [1–3]. Novel network protocols and algorithms, fitting the restrictions of constrained devices, are being developed and standardized due to limited resources of IoT devices. They are also compatible and operational with Internet Protocol (IP) suite. In this sense, the Constrained Application Protocol (CoAP) is developed by the Internet Engineering Task Force (IETF) to meet the needs of IoT applications. CoAP is a peculiar application layer web transfer protocol for constrained devices and constrained networks. Being lightweight and request/response-based RESTful protocol, CoAP is designed to be de facto standard protocol between devices in an IoT network. CoAP is designed to operate over a datagram-oriented transport such as

\*Correspondence: akdemir@atu.edu.tr

User Datagram Protocol (UDP). For being lightweight and simple transport protocol, UDP does not try to solve the congestion problem within the network. Nevertheless, congestion is also a supreme complication in IoT networks<sup>1</sup>.

When the queuing and storing size of physical devices constituting an IoT network are exceeded or generated traffic within a network gets close to network capacity, the congestion problem is inescapably perceived. Increased queuing delays and packet losses are amongst the exemplary outcomes of congestion within a network. Decreasing the network utilization, congestion definitely leads to a fact known as congestive collapse. Hence, congestion control and avoidance mechanisms need to be implemented within the IoT network. Therefore, the core CoAP specification leverages a default congestion control mechanism.

The CoAP is designed to be simple, and the core CoAP specification offers a default CoAP CC mechanism utilizing retransmission timeout (RTO) with binary exponential back off (BEB) for lost packets. As lost packets are retransmitted at exponentially growing time, default CoAP CC is very basic and insensitive to network dynamics. So, default CoAP CC can be categorized as conservative. For not adjusting its protocol operation to dynamic network conditions, default CoAP CC may consequently underperform. Thus, core CoAP specification is receptive to novel CC mechanisms leveraging dynamic network conditions. Accordingly, CoAP Simple Congestion Control/Advanced (CoCoA)<sup>2</sup> was proposed to improve the default CoAP CC. CoCoA harnesses round-trip time (RTT) measurements, adaptive RTO back off computations, and RTO aging methods to improve the performance of default CoAP CC.

Although CoCoA considers the network dynamics, the RTO calculation of CoCoA is based on constant coefficient values. However, our experimentation and the results of literature survey revealed that these constant values, in general, do not achieve the best throughput. Inspired by these observations, we designed an adaptive CoCoA, named as mlCoCoA that dynamically sets RTO-related coefficients of CoCoA using a machine learning method depending on the characteristics of the IoT network being considered. In order to compare the performance of mlCoCoA with default CoAP CC and CoCoA, we run extensive experiments. The experiments were conducted over Cooja simulator of Contiki OS [4]. Number of CoAP clients, packet size, and packet delivery ratio (PDR) were varied over 4×4 grid network topology. Each CoAP client sent back-to-back CoAP requests to CoAP servers for 3 min. CoAP clients used Californium [5] implementation of CoAP in Java programming language. On the other hand, each CoAP server used Erbium [6] implementation of CoAP in C programming language. Finally, the throughput of the network is used to compare mlCoCoA with default CoAP CC and CoCoA. The results show that compared to the performance of these two traditional CC mechanisms, mlCoCoA has merit in terms of improving the throughput of CoAP applications. The contributions of the paper can be summarized as follows:

- We propose a new machine learning-based CC mechanism for improving the throughput of the IoT network with dynamic selection of CoCoA coefficient values. To this end, a ground truth dataset containing 60 best throughput samples related to various experimental scenarios was created on a 4×4 grid network topology by varying number of clients, packet size, and physical layer packet delivery ratio (PDR). The throughput of each CoAP client was measured using both the default CoAP CC and CoCoA.

<sup>1</sup>Shelby Z, Hartke K, Bormann C (2014). Constrained Application Protocol, RFC 7252 [online]. Website <https://tools.ietf.org/html/rfc7252> [accessed 05 June 2020].

<sup>2</sup>Bormann C, Betzler A, Gomez C, Demirkol I. (2018). CoAP Simple Congestion Control/Advanced [online]. Website <https://tools.ietf.org/html/draft-ietf-core-cocoa-03> [accessed 05 June 2020].

- By using the created dataset, several models for predicting the values of CoCoA coefficients were built based on support vector machines (SVM). By using 10-fold cross-validation, the accuracy of the developed prediction models has been verified by calculating the root mean squared error (RMSE) values.
- We integrated all prediction models in an approach called mlCoCoA, which can predict and set the CoCoA coefficients according to a given triple of client number, packet size, and physical layer PDR in a 4×4 grid network topology.
- We evaluated and compared the performance of mlCoCoA with default CoAP CC and CoCoA in terms of achieved average throughput. The results show that compared to the performance of these two traditional approaches, mlCoCoA has merit in terms of improving the throughput of CoAP applications.

The rest of the paper is organized as follows. Section 2 briefly reviews the related work. Section 3 details the existing default CoAP CC and CoCoA mechanisms, and our proposed machine learning-based approach. The experimental setup and methodology are introduced in Section 4. Comparative performance evaluation results are presented in Section 5. Finally, conclusions are given in Section 6 along with future directions.

## 2. Related works

Preventing network congestion is a vastly studied area in computer networks. As IoT networks are being emerged recently, the solutions to obviate network congestion in IoT networks are just appearing. In this context, CoAP is designed in such a way that it involves a very simple and basic congestion control scheme, referred to as the default CoAP CC. In the base default CoAP CC specification, advanced mechanisms are left open for performance enhancement. Hence, in RFC 7252, it is emphasized that CoAP needs additional CC mechanisms to encourage experimentation for determining feasible solutions on resource-constrained devices. As far as we know, there exists only one research paper [7] on the survey of CC mechanisms for CoAP in IoT networks. In this section, we briefly review most of the existing CC mechanisms for CoAP in addition to [7].

In [8], the performance of the default CoAP CC along with a simple algorithm based on the measured value over time is evaluated. WiSHFUL architecture is used to run the experiments. Moreover, it is planned that WiSHFUL architecture will be exploited and will be used to extensively run further experiments in order to explore the ability of candidate CC schemes of CoAP. In [9], the authors proposed to use the combination of the RTO estimation algorithm<sup>3</sup> and a set of algorithm enhancements of Simple CoCoA<sup>4</sup>. The proposed enhancement utilizes ideas from the Eifel retransmission timer [10]. Evaluation results reveal that the proposed solution, we named Simple CoCoA-E, is slightly more aggressive than default CoAP CC. Moreover, in conditions of limited traffic fluctuations and when unpredictable events occur, it has been reported as more efficient. In [11], Simple CoCoA<sup>5</sup> along with a couple of optimizations is evaluated against default CoAP CC in terms of throughput and request/response exchange durations. The results show that Simple CoCoA is able to better exploit the available network capacity and is able to increase the throughput by 19%–112% at Flocklab testbed [12].

<sup>3</sup>Paxson V, Allman M, Chu J, Sargent M (2011). Computing TCP's Retransmission Timer [online]. Website <https://tools.ietf.org/html/rfc6298> [accessed 05 June 2020].

<sup>4</sup>Bormann C (2013). CoAP Simple Congestion Control/Advanced [online]. Website <https://datatracker.ietf.org/doc/draft-bormann-core-cocoa/00/> [accessed 05 June 2020].

<sup>5</sup>Bormann C (2014). CoAP Simple Congestion Control/Advanced [online]. Website <https://datatracker.ietf.org/doc/draft-bormann-core-cocoa/01/> [accessed 05 June 2020].

Default CoAP CC and Simple CoCoA are analyzed and evaluated in [13] on Cooja simulator with different traffic levels. Based on findings, CoCoA 4-state strong, an adaptation of Simple CoCoA using a 4-state estimator for variable back offs, is introduced. Moreover, it is inspected that CoCoA 4-state-strong improves throughput in highly lossy networks. Simple E-CoCoA [14] is proposed and evaluated against default CoAP CC<sup>6</sup>, Simple CoCoA-E<sup>7</sup> and Simple CoCoA<sup>8</sup>. E-CoCoA utilizes the retransmission count to estimate the RTO value and lower bound in round trip time variation. The retransmission count of option field of the CoAP is leveraged. Performance evaluation is carried out on a real testbed. The results indicate that Simple E-CoCoA significantly improves the efficiency and performs better than default CoAP CC, Simple CoCoA-E, and Simple CoCoA. Hartke<sup>9</sup> evaluated default CoAP CC, Simple CoCoA, and observe over emulated GPRS/UMTS links and IEEE 802.15.4 links of FIT IoT-Lab testbed [15] in terms of overall PDR, end-to-end delay, and MAC layer packet drops performance metrics. Evaluation results show that Simple CoCoA performs better than or slightly similar to default CoAP CC in contrast to observe in all considered scenarios.

A comparative study of default CoAP CC, Simple CoCoA, Linux RTO [16], and Peak-Hopper RTO [17] is presented in [18]. The emulation results with netem [19] in constant and bust traffic scenarios indicate that default CoAP CC is more efficient at higher congestion levels. In [20], the performance of default CoAP CC and Simple CoCoA is compared where Cooja is used as a simulator on dumbbell, chain, and hospital topologies. It is exposed that Simple CoCoA outperforms default CoAP CC in terms of average delay, packet loss, goodput, and throughput. Further, a comparison of default CoAP CC, Simple CoCoA, Linux RTO, Peak-Hopper RTO, Basic RTO, and Simple CoCoA-S is studied in [21]. GPRS and IEEE 82.15.4 links of Flocklab testbed with continuous and bust traffic scenarios were used for performance evaluation in terms of throughput, settling time, and fairness. The results revealed that Simple CoCoA consistently outperformed the default CoAP CC in all evaluated scenarios. Being too simple, default CoAP CC underperformed other approaches.

CoCoA+ [22] proposes enhancements to address the deficiencies of Simple CoCoA. Default CoAP CC, Simple CoCoA, and CoCoA+ are compared over Cooja simulator in constant (continuous), global event and mixed traffic scenarios, and grid, dumbbell, and chain topologies. Although performing well in a variety of scenarios, it is observed that Simple CoCoA often performs considerably worse than default CoAP CC. In addition, it has been reported that CoCoA+ outperforms both Simple CoCoA and default CoAP CC in most of the considered scenarios and topologies. In [23], comparison of CoCoA+ and default CoAP CC is investigated, and shortcomings of CoCoA+ are declared by using Cooja simulator for continuous and global event traffic scenarios in grid network topologies. The results reveal that CoCoA+ can perform significantly worse than default CoAP CC under busy traffic and few client conditions due to improper selection of the RTOs. Moreover, it is observed that CoCoA+ shows large variations in RTO values.

CoAP-R [24], an adaptive rate-based approach regulating the sending rate of CoAP clients, is suggested where performance evaluation is carried out over Cooja simulator. The proposed approach is evaluated by means of simulations considering a scenario in which traffic is generated in bursts. Simulation results demonstrate that

<sup>6</sup>Shelby Z, Hartke K, Bormann C (2014). Constrained Application Protocol, RFC 7252 [online]. Website <https://tools.ietf.org/html/rfc7252> [accessed 05 June 2020].

<sup>7</sup>Bormann C, Betzler A, Gomez C, Demirkol I (2018). CoAP Simple Congestion Control/Advanced [online]. Website <https://tools.ietf.org/html/draftietfcorecocoa-03> [accessed 05 June 2020].

<sup>8</sup>Bormann C (2016). CoAP Simple Congestion Control/Advanced [online]. Website <https://datatracker.ietf.org/doc/draft-bormann-core-cocoa/03/> [accessed 05 June 2020].

<sup>9</sup>Hartke W (2015). Observing resources in the constrained application protocol [online]. Website <https://tools.ietf.org/html/rfc7641> [accessed 05 June 2020].

CoAP-R assures a fair allocation of network resources. Data collection delays are decreased by 40% when compared to default CoAP CC.

A precise CC algorithm for CoAP, named pCoCoA [25], is presented with a critical analysis of default CoAP CC, CoCoA+, CoCoA 4-state-strong, and Simple CoCoA-E over Cooja simulator by generating periodic and interfering bursty traffic in a grid network topology. Simulation results reveal that pCoCoA lowers the number of retransmissions, at the same time assuring throughputs and delays as well as those of default CoAP CC and CoCoA+.

CCCLA [26], a cognitive approach for congestion control using a game of learning automata, is one of the first adaptive works acting to congestion by using a game of learning automata (LA). Particularly, a team of LA is assigned to a group of effective controllable parameters for congestion control. Maximizing the whole network performance is achieved by trying to learn the best parameter. A group of LA, acting independently, is assigned to each node in the network. Nevertheless, all nodes receive the same feedbacks from the environment. Cooja simulation results indicate that CCCLA performs better than default CoAP CC, CoCoA, and TCP-Siam in terms of packet delivery ratio and average delay.

Inspired by backpressure routing [27], backpressure congestion control for CoAP [28] proposes detecting and alleviating network congestion by devising CC policies based on backpressure control and controlling the injection of data into the network. The proposed schemes are cross-layer and fully decentralized where they are amenable to implementation onto existing protocol stacks of IoT devices. The simulation results in network simulator 3 (ns-3) [29] show that the proposed schemes, namely Griping, Deaf and Fuse, perform satisfactorily in unidirectional and upstream flows and bidirectional web-based CoAP flows. However, the proposed schemes require modifications to existing protocol stack.

In our previous study [30], we presented a comparison of default CoAP CC and CoCoA in terms of throughput by varying number of clients where each client continuously sends back-to-back traffic to servers residing in a  $1 \times 6$ ,  $3 \times 6$  and  $5 \times 6$  grid network topology operated with nullMAC or carrier sense multiple access (CSMA). It turned out that CoCoA is not always better than default CoAP CC in our grid topologies and MAC protocol setups. We concluded that developing new CoAP CC mechanisms are open to research. Finally, in a follow-up study [31], we proposed new models for predicting the average throughput in a  $4 \times 4$  grid CoAP-based IoT network. The main motivation behind this study was that the ahead knowledge of the CoAP throughput in an IoT network can, e.g., be useful for better network capacity planning. To develop the data-driven prediction models, a ground truth dataset was created that includes data related to 60 different experimental scenarios conducted using the default CoAP CC and CoCoA mechanisms. SVM and multiple linear regression (MLR) have been used to build to models. Differently from our previous works and the rest of studies in literature, this paper proposes a new machine learning-based CC mechanism for improving the achievable throughput of IoT networks with dynamic selection of CoCoA coefficient values.

### 3. CC mechanisms for CoAP

In this section, we first give a brief overview of the default CoAP CC and CoCoA mechanisms. Then, we present the details of the proposed mlCoCoA mechanism.

#### 3.1. Default CoAP CC

CoAP defines four types of messages, namely confirmable (CON), reset (RST), nonconfirmable (NON), and acknowledgement (ACK) messages. Upon a CON message is transmitted by a client, an ACK message is

required from the server. At most, a CON message is retransmitted four times before the transmission is determined as failed. In this process, the first initial value of retransmission timeout (RTO) is set to a random value between 2 and 3 s to prevent synchronization issues. The client postulates that the CON message is lost when RTO expires, and no ACK is received from the server. The client retransmits the CON message again and doubles the RTO value. This doubling method is known as binary exponential back off (BEB) algorithm. When an ACK message is not needed, similar to CON, a NON message is transmitted. A RST message is sent when a specific message (i.e. CON or NON) is received, but some context is not clear to properly process the specific message. Thus, neither NON nor RST messages are utilized for congestion control.

### 3.2. CoCoA

Since default CoAP CC disregards network traffic dynamics, CoCoA brings, along with adaptive RTO calculation, a variable back off factor (VBF) and RTO aging methods. An RTO, called as  $RTO_{overall}$ , is computed for a server adaptively by using RTT measurements and RTT variations (RTTVAR). RTO, RTT, and RTTVAR are computed by applying the exponentially weighted moving average (EWMA) algorithm. Besides, CoCoA requires strong and weak parameters for RTO, RTT, and RTTVAR ( $RTT_{strong}$  and  $RTT_{weak}$ ). Strong estimator keeps measured RTT value when no retransmission occurs. On the contrary, weak estimator keeps measured RTT value from retransmitted requests. The measured RTT is obtained from delivering the initial request and collecting the response. Following equations present RTTVAR, RTT, and RTO calculations where  $X$  represents the strong or weak estimators upon receiving a new RTT ( $RTT_{X\_new}$ ).

$$RTTVAR_X = (1 - \beta) \times RTTVAR_X + \beta \times |RTT_X - RTT_{X\_new}| \quad (1)$$

$$RTT_X = (1 - \alpha) \times RTT_X + \alpha \times RTT_{X\_new} \quad (2)$$

The default values for  $\alpha$  and  $\beta$  are 0.25 and 0.125, respectively. Thereupon, any update of  $RTT_{X\_new}$  results with an update of  $RTO_X$  as

$$RTO_X = RTT_X + K_X \times RTTVAR_X. \quad (3)$$

The default values for  $K_{strong}$  and  $K_{weak}$  are 4 and 1, respectively. Lastly,  $RTO_{overall}$ , which represents the ultimate RTO value used for a server, is calculated as

$$RTO_{overall} = \gamma_X \times RTO_X + (1 - \gamma_X) \times RTO_{overall}, \quad (4)$$

where  $\gamma_{strong}$  and  $\gamma_{weak}$  are 0.5 and 0.25, respectively.

$RTO_{overall}$  is then used to decide the initial RTO ( $RTO_{init}$ ). Similar to default CoAP CC, CoCoA selects the  $RTO_{init}$  value from the interval  $[RTO_{overall}, 1.5 \times RTO_{overall}]$ . Upon timeout, contrary to BEB used in default CoAP CC, CoCoA backs off depending on  $RTO_{init}$  by using VBF. If  $RTO_{init}$  is below 1 s, a larger back off factor, i.e. 3, is applied for retransmission. If  $RTO_{init}$  is above 3 s, a smaller back off factor, i.e. 1.5, is chosen for retransmissions. Otherwise, if  $RTO_{init}$  is between 1 s and 3 s, a moderate back off factor corresponding to BEB of default CoAP CC, i.e. 2, is applied for retransmission.

In addition, CoCoA leverages an aging method, called as RTO aging, to avoid obsolete  $RTO_{overall}$  estimates that may have turned out to be invalid for an extended period of time. If  $RTO_{overall}$  is less than 1 s or more than 3 s, and no new RTT measurement is applied for 16 or 4 times the current RTO respectively, the RTO value is modified to the default initial value.

### 3.3. mlCoCoA: the proposed mechanism

CoCoA calculates RTO with predefined constant coefficient values of  $\alpha$ ,  $\beta$ ,  $K_{strong}$ ,  $K_{weak}$ ,  $\gamma_{strong}$ , and  $\gamma_{weak}$ . We claim that these predefined coefficient constants used by CoCoA can adaptively be set with the help of a machine learning (ML) method to improve the achievable throughput of an IoT network. We first calculated the throughput of CoCoA clients by varying combinations of  $\alpha$ ,  $\beta$ ,  $K_{strong}$ ,  $K_{weak}$ ,  $\gamma_{strong}$  and  $\gamma_{weak}$  constants under ranging number of clients, packet size, and physical layer packet delivery ratio (PDR) on a  $4 \times 4$  grid network topology.

For each triple combination of the number of clients, packet size, and physical layer PDR, we found the  $\alpha$ ,  $\beta$ ,  $K_{strong}$ ,  $K_{weak}$ ,  $\gamma_{strong}$ , and  $\gamma_{weak}$  values resulting with the highest throughput. In practice,  $\alpha$ ,  $\beta$ ,  $K_{strong}$ ,  $K_{weak}$ ,  $\gamma_{strong}$ , and  $\gamma_{weak}$  values of CoCoA resulting with the highest throughput can be searched from infinite number of value domain. However, this is not possible as it requires an exhaustive search. Thus, a more feasible method needs to be chosen. In our experiments, we determined a simple method, the details of which is given in Section 4.2., that generates a finite number of values for  $\alpha$ ,  $\beta$ ,  $K_{strong}$ ,  $K_{weak}$ ,  $\gamma_{strong}$ , and  $\gamma_{weak}$ .

Once the dataset containing the ground truth values of  $\alpha$ ,  $\beta$ ,  $K_{strong}$ ,  $K_{weak}$ ,  $\gamma_{strong}$ , and  $\gamma_{weak}$  for each triple of number of clients, packet size, and physical layer PDR is created, any machine learning method can be applied on the dataset to train and create the prediction models. Afterwards, by feeding these models with blind input data, which represents a particular network configuration including ranging number of client size, packet size, and PDR; the values of  $\alpha$ ,  $\beta$ ,  $K_{strong}$ ,  $K_{weak}$ ,  $\gamma_{strong}$ , and  $\gamma_{weak}$  can be predicted. Finally, mlCoCoA can initialize and run the traditional CoCoA with these predicted coefficient values to improve the achievable throughput of the IoT network. Figure 1 illustrates the concept and components of mlCoCoA.

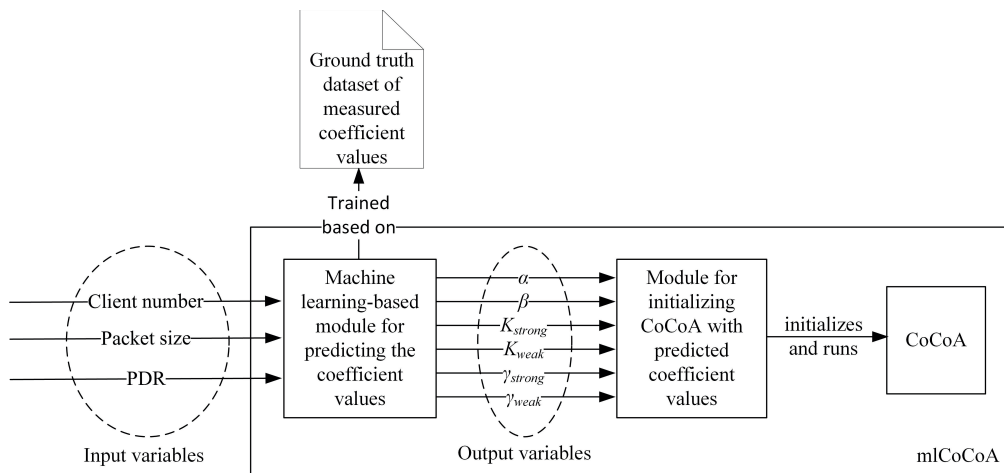


Figure 1. Concept and components of mlCoCoA.

From that point of view, CoCoA's RTO calculation resembles TCP's RTO calculation where RTT and RTTVAR are calculated by using a low-pass filter. Moreover, the CoCoA's coefficients are similar to TCP's coefficient. This is a well-accepted method for RTO calculation. Hence, we also adapt the same low-pass filter mechanism for RTO calculation in mlCoCoA similar to CoCoA's RTO calculation. However, we dynamically assign the coefficients of mlCoCoA by leveraging a machine learning method. By doing so, we expect that mlCoCoA estimates RTOs more accurately.

In our current implementation, the learning phase from the training set is statically achieved offline, and the predicted coefficient values are manually passed to mlCoCoA. However, the future implementations of mlCoCoA might leverage dynamic learning, where these parameters can be recomputed and placed in mlCoCoA in run-time based on continuous measurements of client number, packet size and PDR. Finally, it is noteworthy that mlCoCoA does not need to be run on IoT server nodes with limited capacity of memory, energy and processing power. It only runs on CoAP clients residing over the Internet with plentiful resources. This is just the opposite of classical client-server model.

## 4. Experimental setup and methodology

### 4.1. Experimental environment

We conducted our experiments over Cooja simulator of ContikiOS. Cooja simulates an IoT network of constrained devices running Contiki operating system. One of the key features of Cooja is the emulation of the off-the-self constrained devices. Emulation of hardware specification and processing capabilities of constrained devices are also supported. The compiled binary image file of a constrained device (i.e. node) can be uploaded into simulated nodes in Cooja. During the simulation, as if real, the uploaded code is then executed by simulated nodes.

Our experiments consist of a varying number of clients and servers. The number of clients is always the same as the number of servers. The clients, residing on the same PC, are programmed with Californium [5] implementation of CoAP where CoCoA and mlCoCoA can be chosen as preferred CC mechanisms. The servers, residing on the same PC, running ContikiOS and networked in Cooja simulator, are programmed with Erbium [6] implementation of CoAP.

Each client runs on Ubuntu as a process communicating with servers by using CoAP on top of UDP, IP and, Ethernet. Each server is a node of TMote Sky motes from Moteiv<sup>10</sup>. Servers are hosted on Ubuntu operating system within Cooja simulator. The network stack of servers to communicate with clients is formed with CoAP, UDP, uIPv6, RPL, SICSlowpan, CSMA based on nullRDC and IEEE 802.15.4 physical layer with a data transmission rate of 250 kbps in the 2.4 GHz radio band. Figure 2 depicts Contiki network layers used in this study.

Erbium CoAP	
UDP	
uIPv6 / Contiki RPL	
SICSlowpan	
CSMA+ NullRDC	CSMA+ ContikiMAC
IEEE 802.15.4 PHY	

Figure 2. Contiki OS network layers.

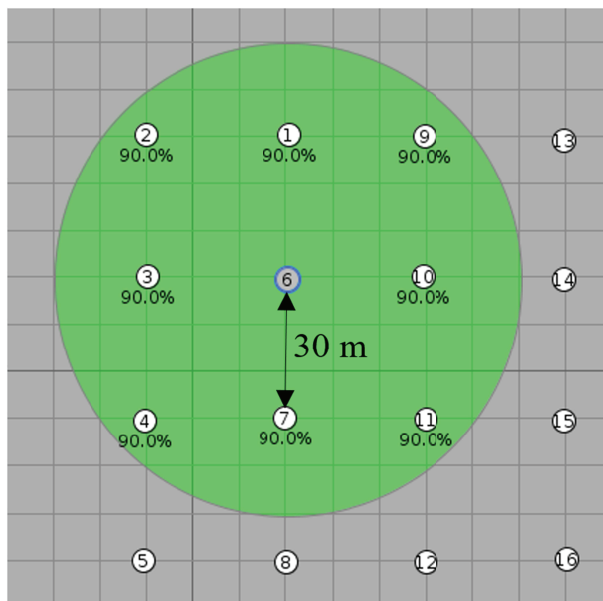
### 4.2. Experimental scenario

During the experiments, clients send CoAP CON requests to servers whereas servers reply with CoAP ACK responses containing piggy-backed responses. In our experiments, the number of clients is varied as 3, 6, 9, 12,

<sup>10</sup>Corporation M (2006). TMote Sky: Ultra low power IEEE 802.15.4 compliant wireless sensor module [online]. Website <http://www.eecs.harvard.edu/konrad/projects/shimmer/references/tmote-skydatasheet.pdf> [accessed 05 June 2020].



and 15. The size of CoAP responses, piggy-backed in CoAP ACK, is ranged as 12, 24, 36, and 48 byte. Finally, PDR of the IoT network that consists of servers is set up with 90%, 95%, and 100% values. These variations let us create 60 ( $5 \times 4 \times 3$ ) different experimental cases. In all experimental cases, the topology of the IoT network is a  $4 \times 4$  grid network topology that is constructed by servers. Figure 3 illustrates the emulated  $4 \times 4$  grid network topology and the simulation parameters.



#### Overview of simulation parameters:

Packet sizes: 12, 24, 36, and 48 Byte

PDRs: 90%, 95%, and 100%

Number of clients: 3, 6, 9, 12, and 15

Congestion control: Default CoAP CC, CoCoA, and mlCoCoA

MAC layer: Carrier Sense Multiple Access (CSMA) and nullRDC

Experiments: In total, 60 different experimental scenarios each of which has been repeated 3 times and has a duration of 3 minutes.

**Figure 3.** Emulated  $4 \times 4$  grid network topology and simulation parameters.

Each client randomly picks a server residing at Cooja simulator in an experimental case. Concurrently, each client starts continuously sending CoAP CON requests, named as continuous traffic, to the picked server for 120 s. Continuous traffic means that clients send back-to-back CoAP CON requests to corresponding servers. Upon receiving a response from the corresponding server, the client immediately sends another CoAP CON request. Continuous traffic is justified as one of common traffic in IoT networks [21, 25].

In an experimental case, each client runs CoAP with CoCoA configured with varying the values of  $\alpha$ ,  $\beta$ ,  $\gamma_{strong}$ ,  $\gamma_{weak}$ ,  $K_{strong}$ , and  $K_{weak}$  parameters of CoCoA. These variations let us explore the performance of CoCoA with different parameter settings. As expected, these parameters can be configured with infinite values. Hence, we narrowed down these value ranges. Particularly, at first,  $\alpha$  and  $\beta$  are varied from 0.10 to 0.90 stepped with 0.05 (in total 9 values for  $\alpha$  and 9 values for  $\beta$ ) where  $K_{strong}$ ,  $K_{weak}$ ,  $\gamma_{strong}$ , and  $\gamma_{weak}$  are set as 4, 1, 0.5 and 0.25, respectively. This enabled us to run CoCoA with 81 different combinations of  $\alpha$ ,  $\beta$ ,  $K_{strong}$ ,  $K_{weak}$ ,  $\gamma_{strong}$ , and  $\gamma_{weak}$  for each experimental case. In each experimental case, each combination was executed 3 times and the average throughput is calculated. Then, for each experimental case,  $\alpha$  and  $\beta$  values combined with  $K_{strong}$ ,  $K_{weak}$ ,  $\gamma_{strong}$ , and  $\gamma_{weak}$ , resulting in the highest throughput, are found.

In the second step, each experimental case is run with the best values of  $\alpha$  and  $\beta$  found in the previous step along with 4 and 1 values for  $K_{strong}$  and  $K_{weak}$ , respectively, where  $\gamma_{strong}$  is varied from 0.35 to 0.65 stepped with 0.05 (in total 7 values) and  $\gamma_{weak}$  is varied from 0.10 to 0.40 stepped with 0.05 (in total 7 values). This lets us run CoCoA with 49 different combinations of  $\alpha$ ,  $\beta$ ,  $K_{strong}$ ,  $K_{weak}$ ,  $\gamma_{strong}$ , and  $\gamma_{weak}$  for each

experimental case. Again, in each experimental case, each combination is executed 3 times and the average throughput is calculated. Afterward, for each experimental case, the best  $\gamma_{strong}$  and  $\gamma_{weak}$  combined with  $\alpha$ ,  $\beta$ ,  $K_{strong}$  and  $K_{weak}$ , resulting in the highest throughput, are determined.

Finally, each single experimental case is run with the best values of  $\alpha$ ,  $\beta$ ,  $\gamma_{strong}$  and  $\gamma_{weak}$  extracted in the previous steps where the values of  $K_{strong}$  and  $K_{weak}$  are diverged from 1 to 7 stepped with 1 (in total 7 values for  $K_{strong}$ , and 7 values for  $K_{weak}$ ). With this, CoCoA is run with 49 distinct combinations of  $\alpha$ ,  $\beta$ ,  $K_{strong}$ ,  $K_{weak}$ ,  $\gamma_{strong}$ , and  $\gamma_{weak}$  for each experimental case. In total, 179 (81 + 49 + 49) combinations of  $\alpha$ ,  $\beta$ ,  $K_{strong}$ ,  $K_{weak}$ ,  $\gamma_{strong}$ , and  $\gamma_{weak}$  are examined for each triple combination of the number of clients, packet size, and physical layer PDR. Each triple combination is executed 3 times, and the average throughput is calculated. Thereupon, the best values of  $\alpha$ ,  $\beta$ ,  $K_{strong}$ ,  $K_{weak}$ ,  $\gamma_{strong}$ , and  $\gamma_{weak}$  resulting with the highest throughput are extracted and form our ground truth dataset.

By using the created ground truth dataset, 6 separate models have been created for predicting the values of  $\alpha$ ,  $\beta$ ,  $K_{strong}$ ,  $K_{weak}$ ,  $\gamma_{strong}$ , and  $\gamma_{weak}$  from the triple predictor variables of client number, packet size, and PDR. SVM has been used to build the prediction models. The values of cost ( $C$ ), epsilon ( $\epsilon$ ) for the  $\epsilon$ -insensitive loss function, and the type of kernel function are the major parameters impacting the performance of the SVM-based models. The radial basis function has been utilized as the kernel function, which requests the optimization of the function parameter gamma ( $g$ ). The optimal values of the three model parameters  $C$ ,  $\epsilon$ , and  $g$  have been determined using the grid search technique. The performance of the prediction models has been assessed by calculating the RMSE, the formula of which is shown in Eq. (5).

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (Y - Y')^2} \quad (5)$$

In Eq. 5,  $Y$  is the measured coefficient value,  $Y'$  represents the predicted coefficient value, and  $n$  represents the number of samples in a test subset. To evaluate the generalization error of the prediction models, 10-fold cross-validation has been used. Consequently, for each fold, the training data included 54 samples, while the test data included 6 samples.

With the help of the prediction models, the optimal values of  $\alpha$ ,  $\beta$ ,  $K_{strong}$ ,  $K_{weak}$ ,  $\gamma_{strong}$ , and  $\gamma_{weak}$  for our 4×4 grid network topology have been estimated. Then, we run mlCoCoA with these estimated coefficient values to measure the achieved throughput. For comparison purposes, default CoAP CC and CoCoA were also run using the same network topology and conditions. The throughput metric has been utilized to compare the performance of mlCoCoA with default CoAP CC and CoCoA. In our experiments, a response to a CoAP request is carried directly in an ACK message. Thus, the arrival of an ACK message implies that the corresponding response has successfully been received by the client. Consequently, at the client side, we define the throughput as the average number of CoAP ACK messages received per second. We classify the throughput of mlCoCoA as “outperformed” when the percentage increase rate of mlCoCoA throughput compared to default CoAP CC or CoCoA throughput is higher than 5%. Conversely, we classify the mlCoCoA throughput as “underperformed” when the percentage increase rate of default CoAP CC or CoCoA throughput compared to mlCoCoA throughput is higher than 5%. Finally, the mlCoCoA throughput is categorized as “comparable” if it is within ±5% of the corresponding default CoAP CC or CoCoA throughput.

## 5. Results and discussion

This section is organized in 2 subsections. In the first subsection, the training and validation results of the created models for predicting the values of  $\alpha$ ,  $\beta$ ,  $K_{strong}$ ,  $K_{weak}$ ,  $\gamma_{strong}$ , and  $\gamma_{weak}$  on our  $4 \times 4$  grid network topology are presented. In the second subsection, by using the coefficient values predicted in the previous step, the performance of mlCoCoA is compared with default CoAP CC and CoCoA in terms of achieved throughput.

### 5.1. Results of models in predicting the coefficient values

Table 1 gives the descriptive statistics of the measured and predicted values of  $\alpha$ ,  $\beta$ ,  $K_{strong}$ ,  $K_{weak}$ ,  $\gamma_{strong}$ , and  $\gamma_{weak}$ . Table 2 shows the training and validation results (i.e. the RMSE values) for all prediction models developed by using SVM. All following discussions refer to validation results. However, similar observations also apply to training results.

**Table 1.** Descriptive statistics of the measured and predicted coefficients.

Target variable	Minimum	Maximum	Mean	Standard deviation
$\alpha$ (measured)	0.10	0.40	0.22	$\pm 0.10$
$\alpha$ (predicted)	0.13	0.29	0.21	$\pm 0.04$
$\beta$ (measured)	0.10	0.40	0.26	$\pm 0.10$
$\beta$ (predicted)	0.13	0.40	0.26	$\pm 0.07$
$K_{strong}$ (measured)	1.00	7.00	4.52	$\pm 1.78$
$K_{strong}$ (predicted)	0.99	6.30	4.45	$\pm 1.48$
$K_{weak}$ (measured)	1.00	7.00	2.23	$\pm 1.83$
$K_{weak}$ (predicted)	1.31	3.35	2.28	$\pm 0.40$
$\gamma_{strong}$ (measured)	0.35	0.65	0.51	$\pm 0.11$
$\gamma_{strong}$ (predicted)	0.40	0.60	0.51	$\pm 0.07$
$\gamma_{weak}$ (measured)	0.10	0.40	0.27	$\pm 0.10$
$\gamma_{weak}$ (predicted)	0.25	0.30	0.27	$\pm 0.02$

**Table 2.** Averages of 10-fold training and validation results for  $\alpha$ ,  $\beta$ ,  $K_{strong}$ ,  $K_{weak}$ ,  $\gamma_{strong}$ , and  $\gamma_{weak}$  prediction models.

Predictor variables	Target variable	RMSE (Training)	RMSE (Validation)
Number of clients, packet size, physical layer PDR	$\alpha$	0.079	0.091
	$\beta$	0.071	0.076
	$K_{strong}$	0.504	1.797
	$K_{weak}$	1.670	1.796
	$\gamma_{strong}$	0.045	0.099
	$\gamma_{weak}$	0.096	0.098

Among all developed prediction models, the relatively lower RMSEs have been observed in predicting the values of  $\alpha$ ,  $\beta$ ,  $\gamma_{strong}$ , and  $\gamma_{weak}$ . In more detail, the model for prediction of  $\beta$  yields the lowest error (RMSE = 0.076). The second lowest RMSE has been yielded by the model for prediction of  $\alpha$  (RMSE = 0.091). The RMSEs of the models for prediction of  $\gamma_{strong}$  and  $\gamma_{weak}$  have been found to be comparable to each other

(i.e. RMSE for  $\gamma_{strong} = 0.099$ , RMSE for  $\gamma_{weak} = 0.098$ ). These models occupy the third rank in terms of achieved prediction error. The relatively highest RMSEs have been observed for prediction of  $K_{strong}$  and  $K_{weak}$  because the gap between the minimum and maximum values of the  $K_{strong}$  and  $K_{weak}$  is the highest (i.e. between 1 and 7) compared to that of the other coefficient values, which vary between 0.10 and 0.90. The  $K_{strong}$  and  $K_{weak}$  prediction models give RMSE values of 1.797 and 1.796, respectively. It can be concluded that all developed models produce RMSEs within limits of acceptable accuracy, and can be used to predict the values of  $\alpha$ ,  $\beta$ ,  $K_{strong}$ ,  $K_{weak}$ ,  $\gamma_{strong}$ , and  $\gamma_{weak}$  under ranging number of clients, packet size, and physical layer PDR on a  $4 \times 4$  grid network topology.

The Wilcoxon signed-rank test, the details of which is given in [32], has been utilized to determine whether the differences between measured and predicted values of  $\alpha$ ,  $\beta$ ,  $K_{strong}$ ,  $K_{weak}$ ,  $\gamma_{strong}$ , and  $\gamma_{weak}$  obtained by SVM are statistically significant or not. In this case, the value of  $n$  is set to 60 samples whereas the two-sided level of significance, i.e.  $\lambda$ , is used as 0.05. Depending on the target coefficient being considered, the values of  $W$  range from 764 to 912. Since the sample size is greater than 20, the table of critical values for  $W$  cannot be utilized. Alternatively, the statistical analysis can be conducted using the normal distribution approximation. In this case, the required calculations for different cases yield  $z$  values ranging from  $-1.11$  to  $-0.02$ , which in turn yield  $P$  values in the range of 0.27 and 0.98 for  $\lambda = 0.05$ . All  $P$  values are bigger than  $\lambda$ , therefore the null hypothesis is accepted. In conclusion, the test results reveal that there is a statistically insignificant difference between measured and predicted values of  $\alpha$ ,  $\beta$ ,  $K_{strong}$ ,  $K_{weak}$ ,  $\gamma_{strong}$ , and  $\gamma_{weak}$  prediction models. Table 3 shows the values of  $W$ ,  $z$ , and  $P$  for each coefficient. Furthermore, Figure 4 shows the scatter plots for actual vs. predicted values of each coefficient using SVM and 10-fold cross-validation.

**Table 3.**  $W$ ,  $z$ , and  $P$  values for each target variable.

Target variable	$W$ value	$z$ value	$P$ value
$\alpha$	862	-0.39	0.70
$\beta$	912	-0.02	0.98
$K_{strong}$	775.5	-0.83	0.41
$K_{weak}$	764	-1.11	0.27
$\gamma_{strong}$	903	-0.09	0.93
$\gamma_{weak}$	873	-0.31	0.76

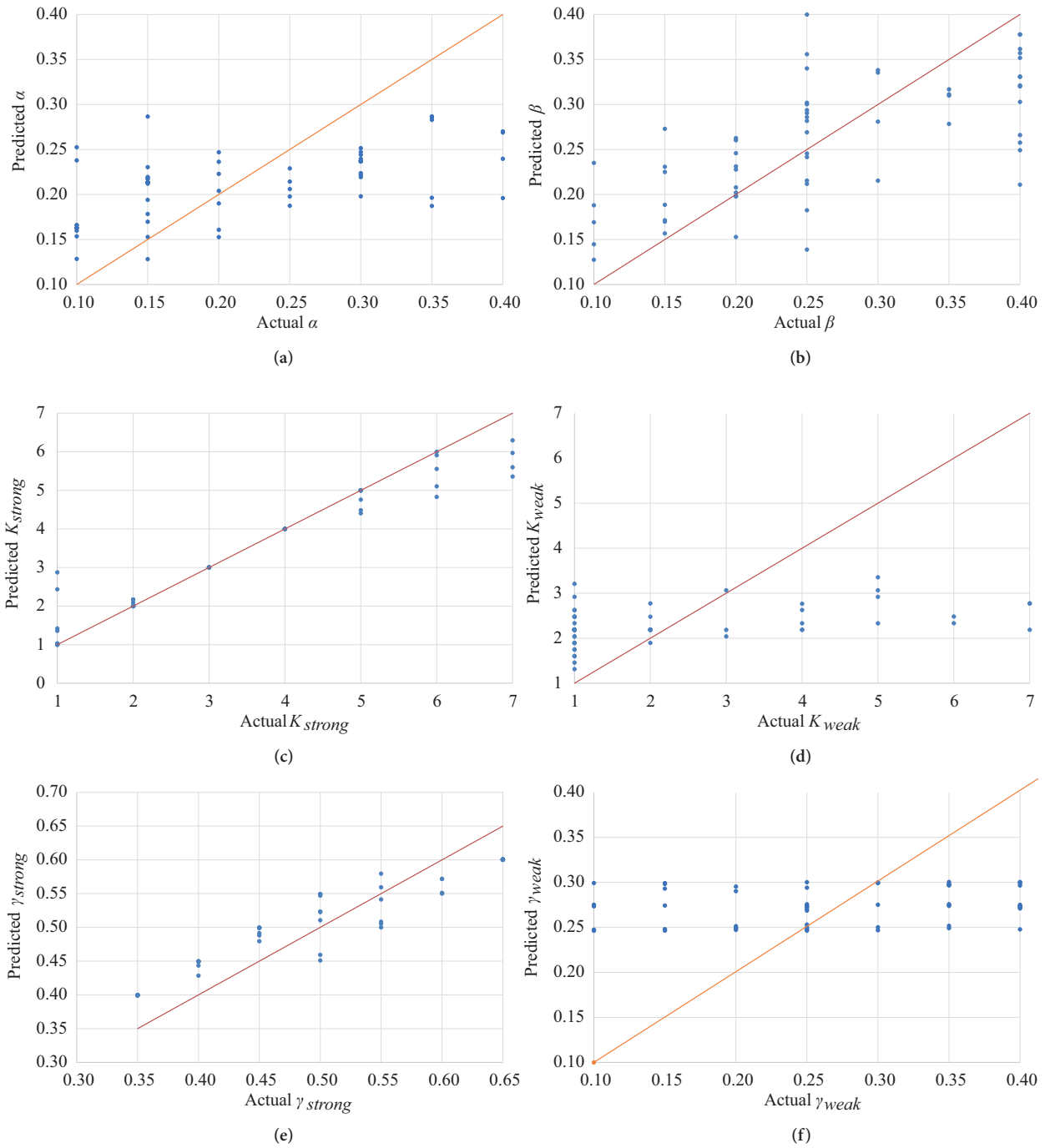
## 5.2. Comparing the throughput of mlCoCoA with default CoAP CC and CoCoA

After an exhaustive search of the related literature, we were able to find four major studies that proposed alternative CC mechanisms designed for IoT networks, i.e. the default CoAP CC<sup>11</sup>, CoCoA<sup>12</sup>, pCoCoA [25] and CCCLA [26]. However, to the best of our knowledge, the source codes of pCoCoA and CCCLA have never been publicly released so that the performance of mlCoCoA could only be compared against the one achieved by using the default CoAP CC and CoCoA mechanisms.

Table 4 through Table 8 show the throughput results of mlCoCoA compared to default CoAP CC and CoCoA under different congestion scenarios with various client numbers. Figure 5 through Figure 9 represent

<sup>11</sup>Shelby Z, Hartke K, Bormann C (2014). Constrained Application Protocol, RFC 7252 [online]. Website <https://tools.ietf.org/html/rfc7252> [accessed 05 June 2020].

<sup>12</sup>Bormann C (2014). CoAP Simple Congestion Control/Advanced [online]. Website <https://datatracker.ietf.org/doc/draft-bormann-core-cocoa/01/> [accessed 05 June 2020].



**Figure 4.** Scatter plots for actual vs. predicted values of  $\alpha$  (a) ,  $\beta$  (b),  $K_{strong}$  (c),  $K_{weak}$  (d),  $\gamma_{strong}$  (e), and  $\gamma_{weak}$  (f) using SVM and 10-fold cross-validation.

the percentage increase rates in throughput of mlCoCoA compared to default CoAP CC and CoCoA under different congestion scenarios with various clients.

Figure 5 shows the percentage increase rates in throughput of mlCoCoA compared to default CoAP CC and CoCoA under lowly congested scenario with 3 clients. In general, mlCoCoA outperforms both default

**Table 4.** Comparison of default CoAP CC, CoCoA and mlCoCoA in terms of throughput under lowly congested scenario with 3 clients.

ID	PS (byte)	PDR (%)	Default CoAP CC throughput (Avg. Msgs/s)	CoCoA throughput (Avg. Msgs/s)	mlCoCoA throughput (Avg. Msgs/s)
1	12	90	1.65	1.83	2.09
2	12	95	1.84	2.05	1.99
3	12	100	2.89	2.75	3.08
4	24	90	0.54	1.25	1.25
5	24	95	1.8	1.82	2.00
6	24	100	1.15	1.66	1.75
7	36	90	2.03	1.66	2.15
8	36	95	1.93	1.68	1.69
9	36	100	2.88	2.84	2.89
10	48	90	0.57	0.41	0.66
11	48	95	2.41	2.44	2.97
12	48	100	2.41	2.57	2.50

**Table 5.** Comparison of default CoAP CC, CoCoA and mlCoCoA in terms of throughput under lowly congested scenario with 6 clients.

ID	PS (byte)	PDR (%)	Default CoAP CC throughput (Avg. Msgs/s)	CoCoA throughput (Avg. Msgs/s)	mlCoCoA throughput (Avg. Msgs/s)
13	12	90	1.15	1.12	1.18
14	12	95	1.22	1.26	1.34
15	12	100	1.53	1.66	1.51
16	24	90	0.95	1.07	1.16
17	24	95	1.32	1.15	1.44
18	24	100	1.55	1.40	1.38
19	36	90	0.84	1.00	1.17
20	36	95	1.24	1.23	1.21
21	36	100	1.47	1.46	1.32
22	48	90	0.67	0.89	1.07
23	48	95	1.40	1.46	1.50
24	48	100	1.66	1.73	1.70

CoAP CC and CoCoA despite very minor exceptions. Unusually, in a case where PDR equals 95% and packet size equals 36 byte, default CoAP CC outperforms mlCoCoA. Totally, in 11 out of 12 cases, mlCoCoA has higher throughput than default CoAP CC. In cases where PDR equals to 100% and packet size equals to 36; and PDR equals to 100% and packet size equals to 48, both default CoAP CC and CoCoA have comparable throughput. In cases where PDR equals to 95% and packet size equals to 12 byte; and PDR equals to 100% and packet size equals to 48 byte, CoCoA slightly outperforms mlCoCoA. Totally, in 10 out of 12 cases, mlCoCoA

**Table 6.** Comparison of default CoAP CC, CoCoA and mlCoCoA in terms of throughput under moderately congested scenario with 9 clients.

ID	PS (byte)	PDR (%)	Default CoAP CC throughput (Avg. Msgs/s)	CoCoA throughput (Avg. Msgs/s)	mlCoCoA throughput (Avg. Msgs/s)
25	12	90	0.83	0.89	0.90
26	12	95	0.86	0.89	0.73
27	12	100	1.2	1.10	0.99
28	24	90	0.8	0.67	0.79
29	24	95	0.95	0.76	0.87
30	24	100	1.03	1.06	0.88
31	36	90	0.62	0.71	0.77
32	36	95	0.92	0.73	0.73
33	36	100	1.1	1.00	1.02
34	48	90	0.74	0.77	0.82
35	48	95	0.96	0.92	0.88
36	48	100	0.95	0.89	0.97

**Table 7.** Comparison of default CoAP CC, CoCoA and mlCoCoA in terms of throughput under heavily congested scenario with 12 clients.

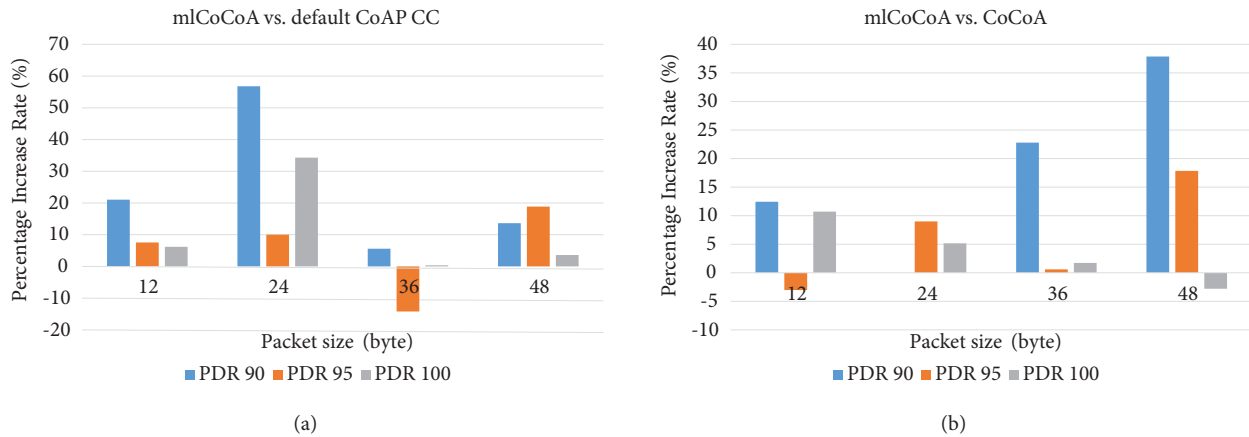
ID	PS (byte)	PDR (%)	Default CoAP CC throughput (Avg. Msgs/s)	CoCoA throughput (Avg. Msgs/s)	mlCoCoA throughput (Avg. Msgs/s)
37	12	90	0.56	0.53	0.55
38	12	95	0.67	0.66	0.70
39	12	100	0.94	0.73	0.80
40	24	90	0.49	0.52	0.59
41	24	95	0.73	0.64	0.62
42	24	100	0.87	0.81	0.80
43	36	90	0.6	0.53	0.62
44	36	95	0.67	0.64	0.65
45	36	100	0.79	0.69	0.68
46	48	90	0.52	0.49	0.54
47	48	95	0.72	0.64	0.66
48	48	100	0.75	0.70	0.67

has higher throughput than CoCoA. The overall result indicates that mlCoCoA achieves the best in lowly congested scenarios.

The percentage increase rates in throughput of mlCoCoA compared to default CoAP CC and CoCoA under lowly congested scenario with 6 clients are presented in Figure 6. mlCoCoA outperforms both default CoAP CC and CoCoA other than cases where PDR equals to 100%. Mostly, when PDR equals to 100%, default CoAP CC achieves higher throughput than mlCoCoA; and when packet size equals 12, both default CoAP CC

**Table 8.** Comparison of default CoAP CC, CoCoA and mlCoCoA in terms of throughput under heavily congested scenario with 15 clients.

ID	PS (byte)	PDR (%)	Default CoAP CC throughput (Avg. Msgs/s)	CoCoA throughput (Avg. Msgs/s)	mlCoCoA throughput (Avg. Msgs/s)
49	12	90	0.49	0.50	0.55
50	12	95	0.62	0.52	0.56
51	12	100	0.75	0.69	0.72
52	24	90	0.49	0.51	0.52
53	24	95	0.54	0.52	0.51
54	24	100	0.64	0.50	0.54
55	36	90	0.48	0.46	0.51
56	36	95	0.56	0.52	0.52
57	36	100	0.65	0.58	0.57
58	48	90	0.50	0.47	0.48
59	48	95	0.55	0.56	0.54
60	48	100	0.58	0.52	0.46



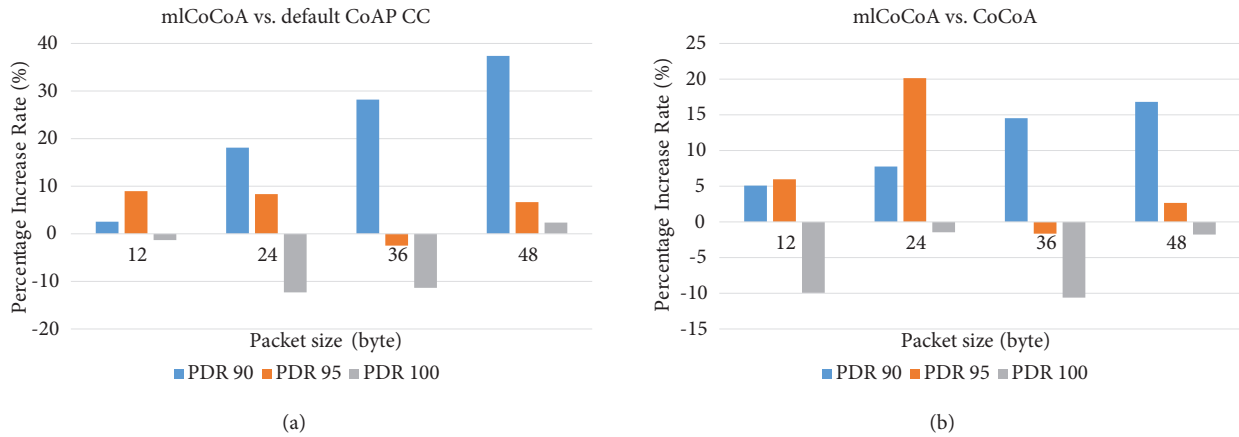
**Figure 5.** Percentage increase rates in throughput of mlCoCoA compared to default CoAP CC (a) and CoCoA (b) under lowly congested scenario with 3 clients.

and mlCoCoA have comparable throughput. Totally, in 8 out of 12 cases, mlCoCoA has higher throughput than default CoAP CC. On the other hand, in 7 out of 12 cases, mlCoCoA has higher throughput than CoCoA. When PDR equals to 100%, CoCoA achieves better throughput than mlCoCoA.

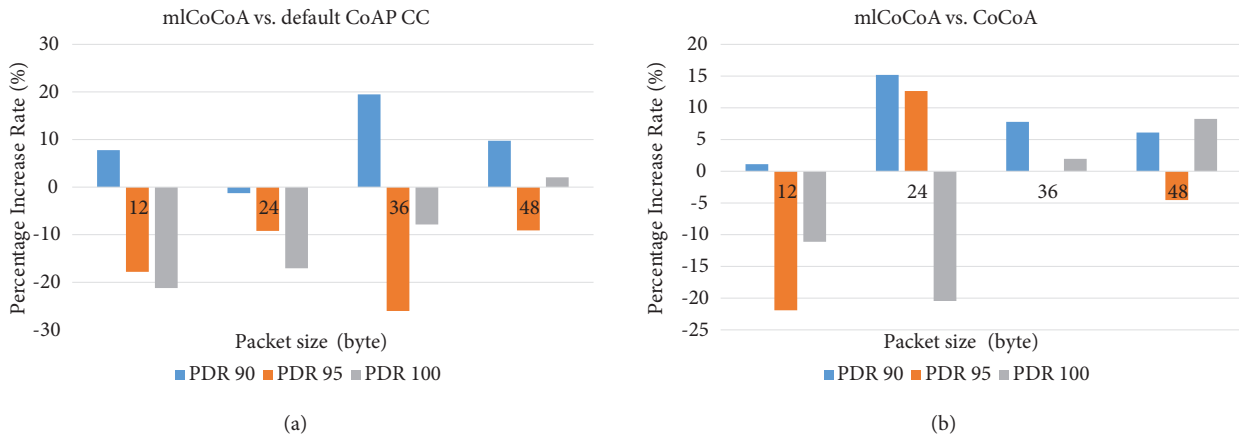
Figure 7 shows the percentage increase rates in throughput of mlCoCoA compared to default CoAP CC and CoCoA under moderately congested scenario with 9 clients. When the PDR is 95% and 100%, the network gets more congested. As a result, default CoAP CC achieves better throughput than mlCoCoA. On the other hand, in general, there is no clear superiority between mlCoCoA and CoCoA in terms of throughput improvement.

Figure 8 illustrates the percentage increase rates in throughput of mlCoCoA compared to default CoAP CC and CoCoA under heavily congested scenario with 12 clients. Again, when the PDR is 95% and 100%, the





**Figure 6.** Percentage increase rates in throughput of mlCoCoA compared to default CoAP CC (a) and CoCoA (b) under lowly congested scenario with 6 clients.

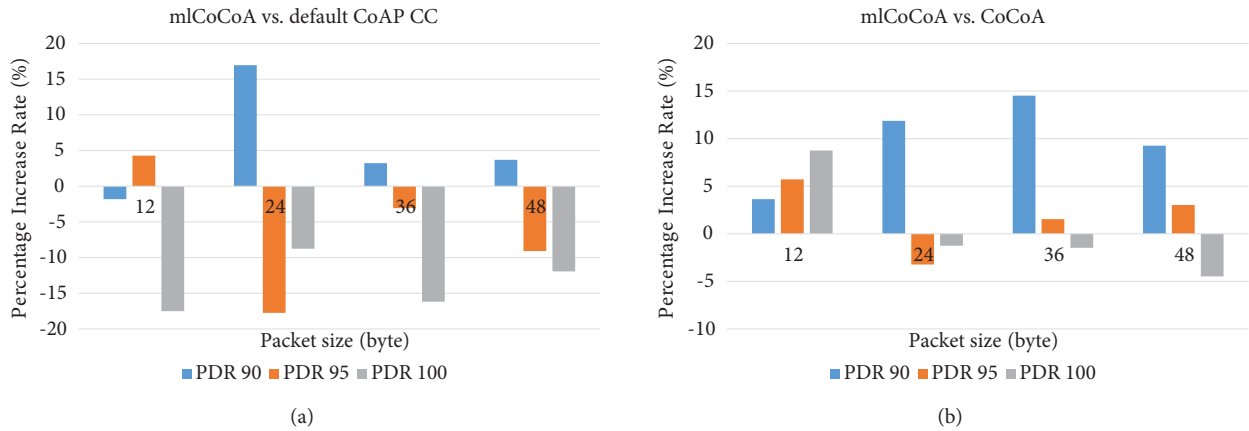


**Figure 7.** Percentage increase rates in throughput of mlCoCoA compared to default CoAP CC (a) and CoCoA (b) under moderately congested scenario with 9 clients.

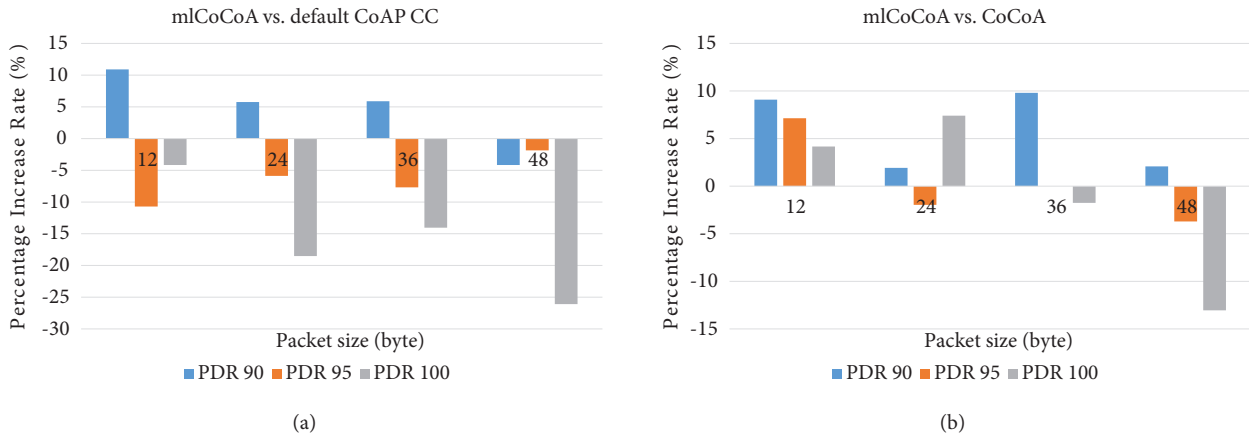
network gets more congested. Hence, default CoAP CC achieves better throughput than mlCoCoA. On the other hand, in general, mlCoCoA outperforms CoCoA in terms of throughput improvement under a heavily congested network.

Figure 9 gives the percentage increase rates in throughput of mlCoCoA compared to default CoAP CC and CoCoA under heavily congested scenario with 15 clients. Again, default CoAP CC outperforms mlCoCoA in highly congested scenarios where PDR is 95% and 100%. On the other hand, mlCoCoA performs significantly better than CoCoA aside from minor exceptions.

In conclusion, our experimental results show that mlCoCoA outperforms default CoAP CC at light and moderate traffic loads. We believe that the reason for this is the fact that the default CoAP CC backs off in an exponential fashion. This is a very conservative approach. As a result, default CoAP CC underutilizes the network when the traffic is low or moderate. However, in heavy traffic loads, mlCoCoA and default CoAP CC compete equally. We believe that this is due to aggressive Variable Backoff Factor (VBF) of mlCoCoA. This behaviour calculates immature RTOs increasing the network traffic further in heavy traffic loads.



**Figure 8.** Percentage increase rates in throughput of mlCoCoA compared to default CoAP CC (a) and CoCoA (b) under heavily congested scenario with 12 clients.



**Figure 9.** Percentage increase rates in throughput of mlCoCoA compared to default CoAP CC (a) and CoCoA (b) under heavily congested scenario with 15 clients.

On the other hand, mlCoCoA outperforms CoCoA in almost every traffic load scenario because mlCoCoA adjust its RTO better than CoCoA. However, there are a few cases where CoCoA outperforms mlCoCoA. This may be caused by the fact that the results are averaged with only 3 experiments. We believe that if we increase the number of experiments, mlCoCoA always performs better than CoCoA. However, because Cooja was crashing down after a certain time, we had to limit the number of experiments to 3. Moreover, a few negative cases might be diminished by developing more accurate coefficient prediction models with the help of other promising machine learning methods.

### 6. Conclusion and future work

IoT is an emerging area that requires more attention and concrete work. Much the same, congestion problem exists within IoT networks. There has been some research to tackle the congestion problem of IoT networks within IETF in the context of the application layer of the Internet. Based on this, default CoAP CC, a simple yet efficient solution, is engaged in solving congestion complications within IoT networks. However, studies showed that default CoAP CC does not always utilize the network dynamics the best to solve the congestion.

As a result, CoCoA has been proposed to better utilize IoT networks. Although CoCoA considers the network dynamics, the RTO calculation of CoCoA is based on constant coefficient values. However, our experiments show that these constant values, in general, do not achieve the best throughput. Inspired by this observation, we proposed a new mechanism, called mlCoCoA where RTO calculation of CoCoA is identified based on the characteristics of an IoT network with the help of machine learning methods. The results show that compared to default CoAP CC and CoCoA throughput, mlCoCoA has merit in terms of improving the achieved throughput in an IoT network.

In the future, we plan to evaluate mlCoCoA on more complex mesh topologies using further extended training dataset to generalize its promising potential. Other candidate potential predictors of CoCoA throughput, such as the average delay, round-trip time, jitter, or number of hops, can be integrated into our prediction models. Furthermore, there exist other promising methods, such as deep learning and artificial neural networks, which can be leveraged for improving the prediction accuracy of mlCoCoA over default CoAP CC and CoCoA. Finally, it is noteworthy that the approach proposed in this study can, in general, be applied to any other protocols, such as TCP. By dynamically selecting the values of coefficient values of TCP with the help of machine learning methods, we plan to investigate whether also the network utilization of TCP can further be improved.

## References

- [1] Atzori L, Iera A, Morabito G. The internet of things: a survey. *Computer Networks* 2010; 54 (15): 2787-2805.
- [2] Whitmore A, Agarwal A, Da Xu L. The internet of things—a survey of topics and trends. *Information Systems Frontiers* 2015; 17 (2): 261-274.
- [3] Al-Fuqaha A, Guizani M, Mohammadi M, Aledhari M, Ayyash M. Internet of things: a survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys & Tutorials* 2015; 17 (4): 2347-2376.
- [4] Dunkels A, Gronvall B, Voigt T. Contiki - a lightweight and flexible operating system for tiny networked sensors. In: *29th IEEE International Conference on Local Computer Networks*; Tampa, FL, USA; 2004. pp. 455-462.
- [5] Kovatsch M, Lanter M, Shelby Z. Californium: scalable cloud services for the Internet of things with CoAP. In: *IEEE International Conference on the Internet of Things (IoT)*; Cambridge, MA, USA; 2014. pp. 1-6.
- [6] Kovatsch M, Duquennoy S, Dunkels A. A low-power CoAP for Contiki. In: *IEEE International Conference on Mobile Ad-Hoc and Sensor Systems*; Valencia, Spain; 2011. pp. 855-860.
- [7] Pramanik A, Luhach AK, Batra I, Singh U. A systematic survey on congestion mechanisms of CoAP based Internet of things. In: *Advanced Informatics for Computing Research*; Jalandhar, India; 2017. pp. 306-317.
- [8] Vallati C, Righetti F, Tanganelli G, Mingozzi E, Anastasi G. ECOAP: experimental assessment of congestion control strategies for CoAP using the WISHFUL platform. In: *IEEE International Conference on Smart Computing*; Sicily, Italy; 2018. pp. 423-428.
- [9] Balandina E, Koucheryavy Y, Gurtov A. Computing the retransmission timeout in CoAP. In: Balandin S, Andreev S, Koucheryavy Y (editors). *Internet of Things, Smart Spaces, and Next Generation Networking. Lecture Notes in Computer Science, Vol 8121*. Berlin, Germany: Springer, 2013, pp. 352-362.
- [10] Ludwig R, Sklower K. The Eifel retransmission timer. *ACM SIGCOMM Computer Communication Review* 2000; 30 (3): 17-27.
- [11] Betzler A, Gomez C, Demirkol I, Kovatsch M. Congestion control for CoAP cloud services. In: *IEEE Emerging Technology and Factory Automation (ETFA)*; Barcelona, Spain; 2014. pp. 1-6.
- [12] Lim R, Ferrari F, Zimmerling M, Walser C, Sommer P et al. FlockLab: a testbed for distributed, synchronized tracing and profiling of wireless embedded systems. In: *International Conference on Information Processing in Sensor Networks, Part of CPSWeek*; New York, NY, USA; 2013. pp. 153-165.

- [13] Bhalerao R, Subramanian SS, Pasquale J. An analysis and improvement of congestion control in the CoAP Internet-of-things protocol. In: IEEE Annual Consumer Communications & Networking Conference (CCNC); Las Vegas, NV, USA; 2016. pp. 889-894.
- [14] Lee JJ, Kim KT, Youn HY. Enhancement of congestion control of Constrained Application Protocol/Congestion Control/Advanced for Internet of Things environment. *International Journal of Distributed Sensor Networks* 2016; 12 (11): 1-13.
- [15] Fleury E, Mitton N, Noël T, Adjih C. FIT IoT-LAB: The largest IoT open experimental testbed. *ERCIM News* 2015; 101 (14): 1-20.
- [16] Sarolahti P, Kuznetsov A. Congestion control in Linux TCP. In: *USENIX Annual Technical Conference*; Berkeley, CA, USA; 2002. pp. 49-62.
- [17] Ekstrom H, Ludwig R. The peak-hopper: a new end-to-end retransmission timer for reliable unicast transport. In: *IEEE International Conference on Computer Communications*; Hong Kong, China; 2004. pp. 2502-2513.
- [18] Jarvinen I, Daniel L, Kojo M. Experimental evaluation of alternative congestion control algorithms for constrained application protocol (CoAP). In: *IEEE 2nd World Forum on Internet of Things (WF-IoT)*; Milan, Italy; 2015. pp. 453-458.
- [19] Hemminger S. Network emulation with netem. In: *6th Australia's National Linux Conference*; Sydney, Australia; 2005. pp. 18-23.
- [20] Hasan HM, Ahmed AI. A comparative analysis for congestion mechanism in CoAP and CoCoA. *Engineering and Technology Journal* 2018; 36 (8A): 867-877.
- [21] Betzler A, Gomez C, Demirkol I, Paradells J. CoAP congestion control for the Internet of things. *IEEE Communications Magazine* 2016; 54 (7): 154-160.
- [22] Betzler A, Gomez C, Demirkol I, Paradells J. CoCoA+: an advanced congestion control mechanism for CoAP. *Ad Hoc Networks* 2015; 33: 126-139.
- [23] Ancillotti E, Bruno R. Comparison of CoAP and CoCoA+ congestion control mechanisms for different IoT application scenarios. In: *IEEE Symposium on Computers and Communications*; Heraklion, Crete, Greece; 2017; pp. 1186-1192.
- [24] Ancillotti E, Bruno R, Vallati C, Mingozzi E. Design and evaluation of a rate-based congestion control mechanism in CoAP for IoT applications. In: *IEEE 19th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*; Chanina, Greece; 2018; pp. 14-15.
- [25] Bolettieri S, Tanganelli G, Vallati C, Mingozzi E. pCoCoA: a precise congestion control algorithm for CoAP. *Ad Hoc Networks* 2018; 80: 116-129.
- [26] Gheisari S, Tahavori E. CCCLA: a cognitive approach for congestion control in internet of things using a game of learning automata. *Computer Communications* 2019; 147: 40-49.
- [27] Jiao Z, Zhang B, Li C, Mouftah HT. Backpressure-based routing and scheduling protocols for wireless multihop networks: a survey. *IEEE Wireless Communications* 2016; 23 (1): 102-110.
- [28] Castellani AP, Rossi M, Zorzi M. Back pressure congestion control for CoAP/6LoWPAN networks. *Ad Hoc Networks* 2014; 18: 71-84.
- [29] Riley GF, Henderson TR. The ns-3 network simulator. In: Wehrle K, Güneş M, Gross J (editors). *Modeling and Tools for Network Simulation*. Berlin, Germany: Springer, 2010, pp. 15-34.
- [30] Demir AK, Abut F. Comparison of CoAP and CoCoA congestion control mechanisms in grid network topologies. *Gümüşhane University Journal of Natural Sciences* 2018; 1: 153-160.
- [31] Demir AK, Abut F. Data-Driven modelling and prediction of CoAP throughput in a grid network topology. *El-Cezeri Journal of Science and Engineering* 2020; 7 (1): 295-303.
- [32] Taheri SM, Hesamian G. A generalization of the Wilcoxon signed-rank test and its applications. *Statistical Papers* 2012; 54: 457-470.