pCoCoA: A precise congestion control algorithm for CoAP

Simone Bolettieri, Giacomo Tanganelli, Carlo Vallati, Enzo Mingozzi

***Abstract*— The Constrained Application Protocol (CoAP) is an IETF standard application protocol for the future Internet of Things (IoT). Since IoT devices are often interconnected by networks characterized by high packet error rates and low throughput, congestion control will be crucial to ensure proper and timed communication in these networks. Therefore, CoCoA+, an advanced congestion control algorithm for CoAP, is currently being specified by the IETF. In this work, we present a critical analysis of CoCoA+ performance and highlight some of its shortcomings and pitfalls. Two different scenarios are considered: one with an increasing traffic load due to an increasing number of CoAP requests, and another with an interfering traffic concurrently transmitted in the network characterized by a bursty pattern. In the former scenario, we show how CoCoA+ may be characterized by many spurious retransmissions at some offered loads close to congestion. In the latter, we show instead how the weak estimator is not particularly effective in adapting to changing traffic loads. In order to overcome such limitations, a number of modifications to CoCoA+ are proposed. The resulting solution, named precise Congestion Control (pCoCoA), is shown to reduce the number of retransmissions, while guaranteeing throughputs and delays comparable to those of CoAP and CoCoA+.**

***Index Terms*— CoAP, CoCoA, CoCoA+, Congestion Control**

I. Introduction

THE Internet of Things (IoT) is getting momentum both in academia, as a very active research topic, and in industry, with smart objects that are meant to thoroughly affect our lives. IoT systems are built on top of IoT devices that collect data and interact with the physical environment. IoT devices are usually cheap, battery-powered devices, with limited computation capabilities. Those devices are equipped with low power transceivers that, however, allow forming Low Power and Lossy Networks (LLN), characterized by possibly large packet error rates, and low throughput. Due to these characteristics, new communication protocols tailored for constrained devices operating in lossy networks are required. To this aim, IETF defined a communication protocol stack for IoT devices. In particular, the IETF CoRE Working Group standardized the Constrained Application Protocol (CoAP) [1], which allows applications to exchange information with IoT devices.

Considering the limited capabilities of nodes and the limited amount of bandwidth available in LLNs, congestion control is crucial to ensure proper and timely delivery of data. In traditional networks, application data is usually transported by the Transmission Control Protocol (TCP), which includes a congestion control mechanism. In LLNs, instead, CoAP employs the User Datagram Protocol (UDP), which does not provide any congestion control mechanism. For this reason, CoAP implements optional reliable data delivery through retransmissions and regulates the amount of data transmitted through a simple congestion avoidance mechanism. The definition of the latter, however, is not trivial: the shared nature of the wireless medium results in collisions of transmissions, and the limited buffer size of IoT devices often produces buffer overflows. Both phenomena can cause frequent packet losses that lead to further message retransmissions, which eventually lead to congestion.

Since CoAP defines only a basic congestion control mechanism, a new advanced congestion control have been proposed recently in [2], and further extended in [3]. Such algorithm, called CoCoA, is currently under standardization within the CoRE Working Group [8]. CoCoA exploits the measured round-trip time (RTT) between the client and the server to adjust the retransmission timeout and therefore avoid too frequent retransmissions. CoCoA has been originally evaluated against CoAP in [2] and [3], showing that it can improve the performance in terms of throughput, packet delivery ratio, and average delays.

In this work, we present an exhaustive performance evaluation of the CoCoA congestion control algorithm[1]. First, we present an evaluation of the algorithm considering two different scenarios, the first one with only CoAP traffic, and the second one in which CoAP traffic competes with interfering greedy traffic that is transmitted without rate control. An in-depth analysis of these simulation results allows us to highlight some novel critical issues of the algorithm. Consequently, we propose a set of modifications to the original algorithm to overcome these limitations. The resulting algorithm, named pCoCoA, is evaluated against CoAP, CoCoA+ and two more recent algorithms: 4-state-strong [7] and CoCoA-E [14]. Results demonstrated that pCoCoA effectively mitigates the issues of the original algorithm, reducing the overall number of retransmissions

C. Vallati, G. Tanganelli, and E. Mingozzi are with the Department of Information Engineering, University of Pisa, Largo Lucio Lazzarino, 2, 56122, Pisa Italy. (e-mail: g.tanganelli@iet.unipi.it, c.vallati@iet.unipi.it, enzo.mingozzi@unipi.it).

S. Bolettieri is with Institute of Informatics and Telematics (IIT)-CNR, 56124 Pisa, Italy (e-mail: s.bolettieri@iit.cnr.it).

[1] A preliminary performance evaluation has been presented in a conference paper [13]. In this initial contribution, we only considered a scenario with CoAP traffic with periodic rate.

while keeping throughputs and delays comparable to those obtained with CoAP, CoCoA, CoCoA-E and 4-state-strong. Moreover, pCoCoA can correctly handle scenarios in which CoAP traffic competes with interfering bursty traffic, whereas other algorithms, such as CoCoA-E, do not. To summarize, the contributions of this work are the following:

- An extended performance evaluation of CoCoA considering two different traffic scenarios: one adopting CoAP periodic traffic, and a second one with periodic traffic sources competing with interfering bursty traffic sources. The analysis of the CoCoA behavior allowed highlighting novel issues of the CoCoA algorithm.
- A novel congestion control algorithm derived from CoCoA named pCoCoA, defined to overcome the limitations highlighted. The proposed congestion control algorithm is evaluated against protocols from the literature.

The rest of the paper is structured as follows. In Section II a short description of CoCoA is provided. Section III highlights the related work. Section IV introduces the methodology adopted and analyzes the performance evaluation results. Section V summarizes the main issues identified by their in-depth analysis. In Section VI we present the proposed pCoCoA algorithm, while in Section VII we present the results of its performance evaluation. Finally, in Section VIII we draw the conclusions.

II. CoCoA+ CONGESTION CONTROL

The Simple Congestion Control/Advanced, also known as CoCoA, is an alternative congestion control algorithm for CoAP originally proposed in [2] and now currently under standardization within the activities of the CoRE Working Group [8]. Different versions of the algorithm have been proposed over time, including minor modifications that helped improving the performance. In this work we consider its last revision, CoCoA+ [3]. The algorithm is composed of the following three main functions:

- a policy to calculate the retransmission timeout (RTO),
- the back-off policy to set the RTO for retransmissions, and
- an ageing policy for the status information.

We briefly describe in the following the CoCoA+ algorithm yet with enough details to allow the reader understand the subsequent in-depth analysis. We refer to [3] for a comprehensive illustration of the algorithm and the rationale behind it.

A. RTO calculation

In order to calculate the RTO, CoCoA+ adopts the same algorithm used by TCP, which updates the RTO value based on measured Round Trip Times (RTTs). According to Karn's algorithm [9], the RTT measurements should be considered only when the message was correctly delivered without retransmissions. In the context of lossy networks, however, many transmissions are expected to experience retransmissions, thus reducing the probability of obtaining valid RTT measurements. For this reason, CoCoA+ considers also transactions that experienced retransmissions to obtain a more accurate RTT estimation. Specifically, two RTO values are calculated: a *strong* RTO, estimated using RTTs samples from transactions that did not required any retransmission, and a *weak* RTO, estimated using RTT values of transactions that required no more than two transmissions. Considering that it is not possible to know for an ACK to which transmission it belongs to, RTT samples are collected measuring the time between the first transmission and the arrival of the response, but only if two retransmissions at most are required.

For each destination, a node maintains the following quantities:

- two smoothed mean RTT estimators: RTT_{strong} and RTT_{weak},
- two smoothed mean variance, called $RTTVAR_{strong}$ and $RTTVAR_{weak}$,
- two RTO estimators, called RTO_{strong} and RTO_{weak}, derived from the strong and weak RTT estimators, respectively, and
- a comprehensive RTO, called $RTO_{overall}$, which keeps track of both RTO_{strong} and RTO_{weak} changes.

Initially, the RTO estimators are initialized with a default value of 2s. The value of the other RTT_x and $RTTVAR_x$ parameters are initialized when the first corresponding RTT value R is measured as follows:

$$RTT_x \leftarrow R, \qquad RTTVAR_x \leftarrow \frac{R}{2} \tag{1}$$

Every time a new RTT sample R is measured, the corresponding strong or weak estimators (based on the number of retransmissions) are updated as follows:

$$RTT_x = (1-\alpha)RTT_x + \alpha R \tag{2}$$

$$RTTVAR_x = (1-\beta)RTTVAR_x + \beta|RTT_x - R| \tag{3}$$

$$RTO_x = RTT_x + K_x RTTVAR_x \tag{4}$$

$$RTO_{overall} = \lambda_x RTO_x + (1-\lambda_x)RTO_{overall} \tag{5}$$

where the following values are recommended: α=0.125, β=0.25, K_{strong} =4, K_{weak}=1, λ_{strong} =0.5, and λ_{weak} =0.25. $RTO_{overall}$ is used to set the initial RTO (RTO_{init}) for the next CON transmission. The actual value is selected using a dithering approach, i.e., RTO_{init} is randomly chosen from the interval [$RTO_{overall}$, 1.5·$RTO_{overall}$].

B. Backoff policy

CoCoA+ introduces a backoff mechanism to set the retransmission timeout. Compared to CoAP, in which the RTO is doubled, CoCoA+ computes the new RTO value for retransmissions according to a variable backoff factor (VBF) that depends on the initial RTO value RTO_{init}. Specifically, the new value of RTO for retransmissions RTO_{new} is evaluated as follows:

$$RTO_{new} = RTO_{previous} * VBF(RTO_{init}) \tag{6}$$

The VBF factor is set according to RTO_{init} to avoid frequent retransmissions in a short time when the RTO value is low, and to avoid long delays in retransmissions when the RTO value is large, instead.

C. Information aging

CoCoA+ introduces a mechanism to age RTO values when RTT updates are not received for a certain time. The rationale

TABLE I
CONGESTION CONTROL ALGORITHMS

Name	Backoff Method	RTT Estimators	RTO aging	Derived from
CoAP	BBF	None	No	None
CoCoA	VBF	Strong & Weak	Yes	LinuxRTO
CoCoA-S	VBF	Strong	Yes	CoCoA
CoCoA-E	VBF	Strong & Weak	Yes	CoCoA & Eifel
4-state-strong	VBF	Four estimators	Yes	CoCoA

is that the RTO estimation becomes obsolete after a certain time and should converge towards the initial value. Specifically, if $RTO_{overall}$ is larger than the base RTO defined in CoAP, which is set by default to 2s, and it is not updated for more than 30s, when a new measurement is obtained the following formula is applied:

$$RTO_{overall} = \frac{2 + RTO_{overall}}{2} \quad (7)$$

If $RTO_{overall}$ is, instead, less than 1s, and it is not updated for a time that is 16 times its actual value, $RTO_{overall}$ is reset to 1s.

III. RELATED WORK

Many recent works focused on evaluating the performance of CoCoA per se. In [4], for instance, CoCoA is evaluated in a typical large-scale IoT scenario in which GPRS is employed to connect IoT nodes. The authors compared CoCoA against other congestion control mechanisms defined for TCP applications. Results showed that CoCoA performs equally or better than TCP-based algorithms. Similar conclusions were drawn in [5], where the authors compared CoCoA to other TCP-based congestion control mechanisms on an emulated Zigbee network. In [6], the authors evaluated CoCoA by means of simulations considering different traffic patterns. Similar to our work, the authors highlighted the poor performance of CoCoA with bursty traffic due to the improper selection of the retransmission timeout value.

Recently, other works proposed modifications to CoCoA. In [7] the authors proposed '4-states-strong', a modification to CoCoA that introduces a more complex RTT estimator to distinguish between wireless losses and congestion losses. Results showed an improvement of the performance in networks where the loss rate is particularly high. In [14], instead, authors proposed CoCoA-E, a modified version of CoCoA based on the Eifel retransmission timer. Simulations showed an improvement of the RTO estimation under two different RTT functions: stair-step and saw-like. Both '4-states-strong' and CoCoA-E will be presented in details in Section VII.A, as they are adopted as term of comparison in the performance evaluation. Finally, in [16] the authors evaluate CoCoA-S, a version of CoCoA that uses only the strong RTO estimator. The variant is evaluated against other TCP congestion control algorithms such as the Linux RTO and the peak-hopper TCP RTO estimator. In such results, however, CoCoA is confirmed to deliver better performance compared to CoCoA-S and the other congestion control strategies, which are demonstrated to be unfit for CoAP.

To conclude, Table I summarizes the congestion control strategies recently proposed in literature, along with their main features.

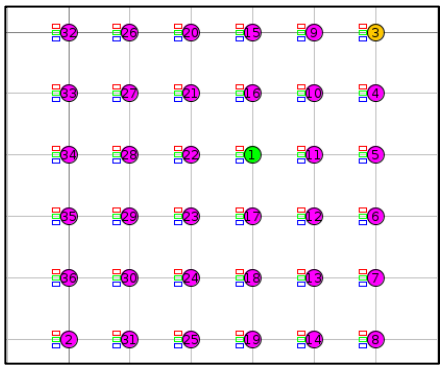


Fig. 1, Network topology

IV. PERFORMANCE EVALUATION

In this section, we analyze the results of the performance evaluation. The performance evaluation is carried out by simulation. To this aim, the Cooja platform is used, leveraging a ContikiOS implementation of CoCoA+ originally obtained from its authors, and then updated to be in line with the latest CoCoA+ specification [3]. Cooja motes are used to emulate wireless sensor devices. The relevant simulation parameters are summarized in Table II. In all scenarios, we consider a grid of 6x6 nodes, as reported in Fig. 1. In the network, one node is the RPL border router (node ID 1), one node act as a CoAP server (node ID 3), and, finally, the remaining nodes are CoAP clients. The linear distance between two nodes in a row is 10m.

In all scenarios, periodic CoAP traffic with CoCoA+ congestion control is considered to emulate a network of sensors that report their measurements at a fixed period. In the first evaluation scenario, only periodic traffic is transmitted. On the other hand, in the second scenario, periodic traffic is mixed with bursty CoAP traffic with default congestion control in order to evaluate CoCoA+ performance when greedy competing traffic is also present in the network.

TABLE II
SIMULATION PARAMETERS

Parameter	Value
L2 protocol	IEEE 802.15.4, 250 Kbps PHY rate
Channel model	Unit Disk GM, tx range = 10mt, interference range = 20mt
MAC buffer size	8 packets
MAC level max retransmissions	8
L3 protocol	6LoWPAN
Routing protocol	RPL
CoAP base ACK-TIMEOUT	3s
CoAP requests buffer size	4
Periodic traffic – Request period (T) and corresponding data rate	0.5s (6048 B/s), 1s (3025 B/s), 2s (1513 B/s), 3s (1010 B/s), 4s (757 B/s), 6s (505 B/s), 7s (434 B/s), 8s (379 B/s), 9s (337 B/s), 10s (304 B/s), 12s (253 B/s), 14s (217 B/s), 16s (191 B/s), 32s (96 B/s), 64s (49 B/s), 70s (45 B/s)
Bursty traffic – packet inter-arrival times	10s, 30s, 40s, 60s, 90s, 100s, 120s, 140s

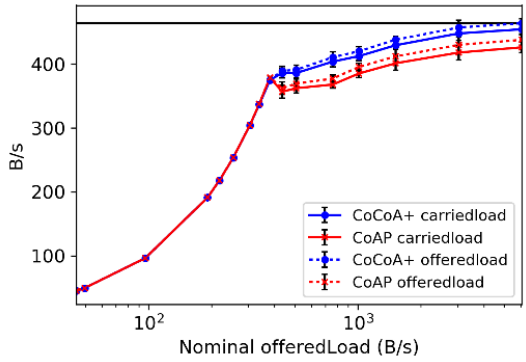


Fig. 2, Carried load vs. offered load.

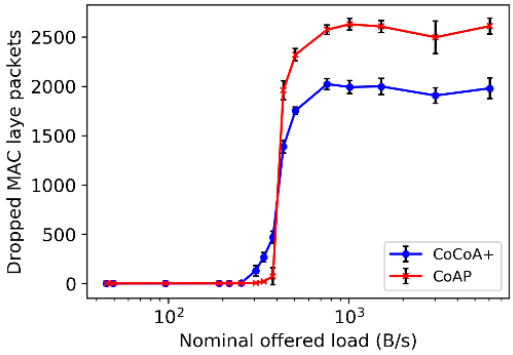


Fig. 3, Overall in-network MAC buffer overflows.

A. *Periodic traffic scenario*

In this scenario, all CoAP clients periodically send confirmable POST requests towards the server with ID 3. Each request has a size of 95B. For each successfully received request, the server replies with a response piggybacked onto a CoAP ACK message. All requests are addressed to an IPv6 global address, which forces messages to be routed through the RPL border router in order to reach the server. Before starting to transmit CoAP requests, nodes wait for 60s to let the RPL topology stabilize, and then for a further random time to avoid synchronization effects. Finally, nodes start sending requests with a common period T, ranging from 70s to 0.5s, in order to inject in the network an increasing amount of data. Each experiment has a duration of 800s. In order to obtain statistically sound results, twelve independent replicas for each scenarios are run, and metrics of interest are then estimated for each scenario along with a 95% confidence interval.

Fig. 2 shows the aggregate carried load and actual offered load as a function of the nominal offered load for both CoAP and CoCoA+. The carried load is defined as the overall amount of data successfully delivered at the server. The actual offered load, instead, is defined as the overall amount of generated data that is transmitted at least once, i.e., that is not discarded at the client because of a buffer overflow at the application layer.

From Fig. 2 we can first observe that for both CoAP and CoCoA+ the respective carried and actual offered loads are very close to each other even when the carried load stops increasing linearly (at a nominal offered load of approximately 380 B/s), i.e., congestion has started in the network. This means that in both cases the server receives almost all the requests transmitted by clients, and therefore the main bottleneck that causes congestion is located at the client, and not in the network. This is expected, as only one outstanding request for each node is allowed when N_{START} is set to one. When the offered load increases, the network traffic increases resulting in longer times to complete a transaction (considering that the client retransmits a request until the ACK is received), which, consequently, results in more frequent buffer overflows at the application layer. However, if we focus on the results obtained when congestion has started, we can also notice that the carried load keeps increasing, though at a much slower rate. This can be explained by considering that congestion is not evenly distributed among all nodes. In particular, nodes farther from the server start discarding requests earlier, while nodes closer to the server, instead, experience shorter delays to complete a transaction, consequently they experience application layer overflows at higher offered loads. Although omitted here for the sake of brevity, this is confirmed by analyzing the carried load per node: the increase of the carried load, after congestion has started, is mainly due to the three clients closest to the server (nodes 4, 9, and 10).

By comparing CoAP and CoCoA+ carried loads from Fig. 2, we can then observe that their performance differ under congestion: as expected, CoCoA+ always outperforms the CoAP simple congestion control mechanism by dynamically adapting retransmission timeouts. In order to get a better insight into this result, we consider a few additional relevant metrics as a function of the nominal offered load. In particular, Fig. 3 shows the overall MAC buffer overflows in the network, i.e., the overall amount of requests discarded at the MAC layer in the network. Fig. 4 shows instead the average number of transmissions (at the client) for each successful transaction. Finally, Fig. 5 shows the average number of ACK messages received at the client for each successful transaction (including duplicates).

These results highlight that three different offered load intervals can be identified to compare the performance of CoCoA+ and CoAP congestion control: a non-congestion interval, with offered loads up to 250 B/s, a pre-congestion interval, with offered loads ranging between 250 B/s and 380 B/s, and, finally, a congestion interval, with offered loads above 380 B/s.

In particular, in the non-congestion interval, corresponding to one request every 12s, or more, per node, the network is essentially operating out of congestion, no buffer overflow occurs and therefore the two algorithms behave in the same way. On

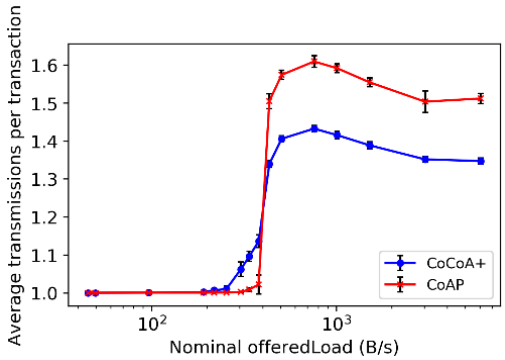


Fig. 4, Average number of transmissions per successful transaction.

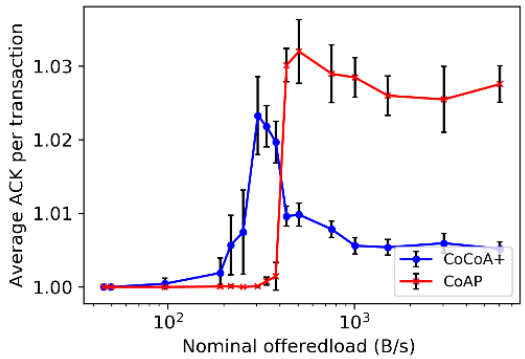


Fig. 5, Average number of ACKs per successful transaction.

the other hand, in the congestion interval, corresponding to one request every up to 8s per node, the network is running under substantial congestion in most of its nodes, and the actual offered load stops increasing linearly with the nominal offered load. In this case, the amount of buffer overflows rapidly increases together with packet delays. A fixed RTO value is therefore not efficient, as its value eventually becomes close or even lower than round-trip times. For this reason, the timeout expires more frequently and often unnecessarily, leading to the transmission of additional traffic that further congests the network. CoCoA+ shows instead its effectiveness in adapting to increasing RTTs by using larger RTO values and therefore reducing the amount of retransmissions per transaction (see Fig. 4). The amount of unnecessarily retransmitted requests is also drastically reduced, as shown in Fig. 5.

Finally, we compare the performance of CoCoA+ and CoAP in the pre-congestion interval, which corresponds to request arrival periods per node between 8s and 12s. This case is interesting since both CoAP and CoCoA+ achieve the same carried load, but they clearly behave differently. In particular, we observe from Fig. 4 that CoCoA+ triggers a number of request retransmissions that increases linearly with the offered load, while retransmissions with CoAP are still negligible. However, considering that the carried load is practically the same, such retransmitted requests with CoCoA+ are unnecessary and they only contribute to increase the traffic load in the network. In fact, either such spurious retransmissions are dropped before reaching the server, as shown by Fig. 3, or they produce duplicate ACKs that are sent back to the client, as shown by Fig. 5. In this range of offered loads CoCoA+ requires therefore more network resources than CoAP to achieve the same carried load, or, said alternatively, congestion is reached earlier than CoAP. As a matter of facts, with a request period of 8s, the carried load with CoCoA+ is even slightly lower than CoAP. Such behavior is consistently exhibited by all nodes in the network - see Fig. 6, which shows the carried load per node for both protocols.

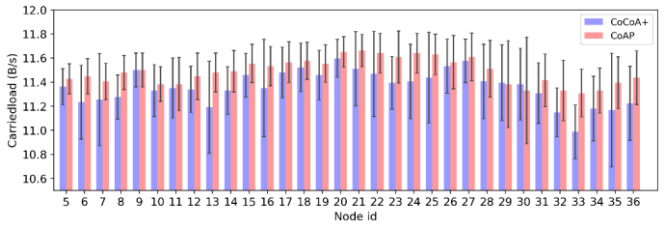


Fig. 6, CoAP vs. CoCoA+ carried load per node, T = 8s.

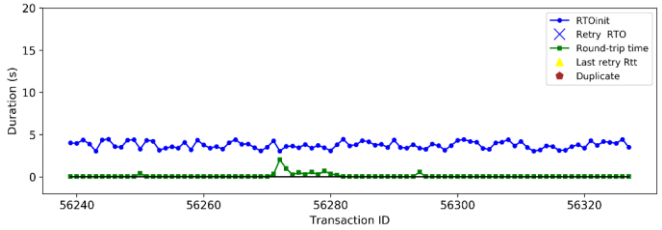


Fig. 7, CoAP operation, node ID 10, T = 8s.

In order to understand this behavior, we provide an insight into the operation of both the algorithms in a selected time interval, in the case of a request period equal to 8s. Fig. 7 and Fig. 8 show the values of all the relevant parameters controlling the transmission of requests over subsequent transactions in CoAP and CoCoA+, respectively. Specifically, in both figures values reported with blue circle marks represent the RTO_{init} values, while green square marks represent the measured RTTs. More precisely, in case of a retransmission, the RTT value is calculated as the delay between the first request transmission and the arrival of an ACK (this is due to the RTT ambiguity problem [5]); in this case, the measured delay between the last retransmission and the ACK reception is also reported using the yellow triangle mark. Finally, in both figures we report with blue cross marks the RTO values calculated as a result of a back-off because of a retransmission, and highlight spurious retransmissions with brown small circle marks.

From Fig. 7 we can observe that RTO values randomly chosen by CoAP in the predetermined range are large enough to cope with RTT variations, thus causing a negligible number of unnecessary fluctuations. On the other hand, from Fig. 8 we can observe that, with CoCoA+, when a series of similar RTT values is sampled, the contribution of RTTVAR to the RTO computation vanishes and, consequently RTO values eventually get very close to actual RTTs. When this occurs, small RTT variations cause spurious retransmissions, which, besides being unnecessary, potentially further exacerbate the issue, since they contribute to increase RTTs in the network, and therefore trigger additional retransmissions.

B. Interfering Bursty Traffic

In this second scenario, we consider five nodes (with IDs 2, 8, 15, 32, and 34, respectively) that transmit bursty traffic that interferes with periodic requests sent to the server by the remaining nodes. In particular, interfering nodes send CoAP requests according to an ON-OFF pattern. The OFF period is set to 120s, while the ON period is varied ranging from 10s to 140s in the different experiments. During the ON periods, requests are sent to the server back to back using CoAP congestion control with an RTO equal to 1s max, and one retransmission at

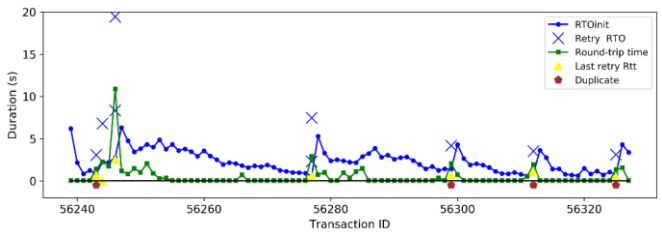


Fig. 8, CoCoA+ operation, node ID 10, T = 8s.

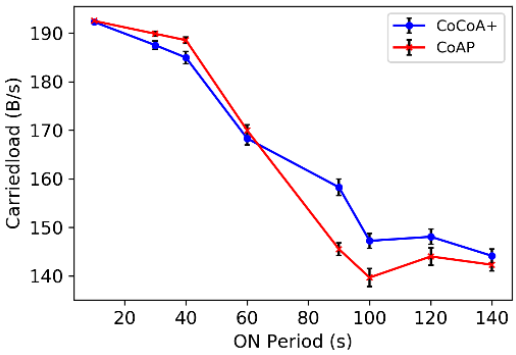


Fig. 9, Carried load – periodic traffic.

most (i.e., MAX_RETRANSMIT is equal to one). Such configuration leads to interfering nodes that are very aggressive during their ON phase. Periodic requests from the other CoAP clients are sent with a constant period set to T = 14s, which ensures that the network operates far from congestion when all interfering nodes are in the OFF phase. Each experiment lasts 1000s. For each scenario twelve independent replicas are run.

The goal of this scenario is to analyze the behavior of CoCoA+, as compared to CoAP, in a non-steady context when the background traffic varies back and forth between light and high loads. Fig. 9 shows the aggregate carried load for CoAP clients with constant periodic traffic as a function of the duration of the ON period. On the other hand, Fig. 10 shows the average number of transmissions at the sender per successful transaction for the same clients.

From Fig. 9 we can observe that CoAP slightly outperforms CoCoA+ in terms of carried load when the ON period is short (up to 60s). This seems to be in contrast with the results shown in Fig. 2, i.e., CoCoA+ performs either equal to CoAP at light loads, and better at high loads. However, this behavior can be explained by the fact that CoCoA+ fails to adapt to congestion when congestion periods (ON periods) are not long enough. In fact, for ON periods longer than 60s, the carried load with CoCoA+ starts decreasing slower than CoAP, becoming then higher than the latter.

However, Fig. 10 shows that in all scenarios CoCoA+ employs on average a larger number of transmissions per transaction than CoAP. Recall that in the previous scenario we observed instead (see Fig. 4) that at steady high loads CoCoA+ succeeds in reducing retransmissions with respect to CoAP by dynamically adapting to congestion. This behavior can be ex-

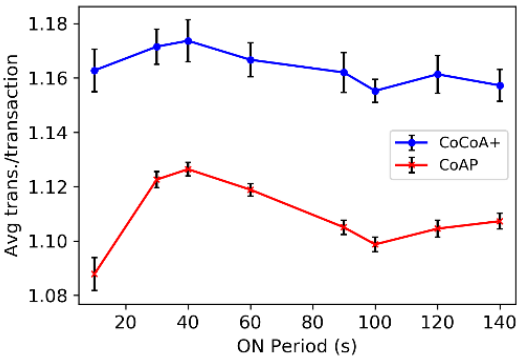


Fig. 10, Transmissions per transaction – periodic traffic.

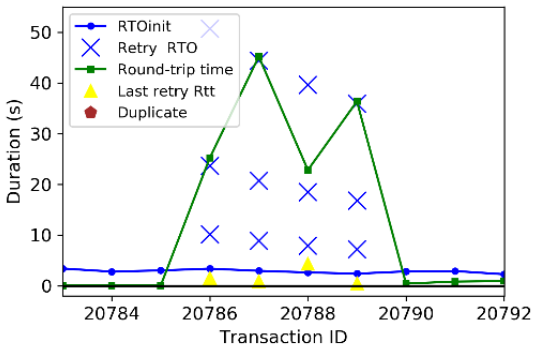


Fig. 11, Weak estimator failing to update RTO_{init}.

plained by considering how RTO_{init} is updated in case of retransmissions, i.e., when the weak estimator plays a major role. If more than two retransmissions occur, the weak estimator is not updated, and therefore RTO_{init} is not updated as well. In addition, if an RTT is sampled after a retransmission that is close to the previous RTO_{init} value, the weight associated to the weak estimator is too small to affect the next RTO_{init}, which therefore does not increase fast to respond to network congestion.

Such circumstances are both frequently observed during ON periods with CoCoA+. Two examples are provided in Fig. 11 and Fig. 12. In particular, Fig. 11 shows that, although a sequence of four transactions that required at least one retransmission occurred, RTO_{init} is not updated by CoCoA+, and its small value contributes to increase the number of retransmissions. On the other hand, in Fig. 12 it is shown how the RTO_{init} is adapted, but not enough to avoid spurious retransmissions corresponding to duplicate ACKs.

V. CoCoA+ Shortcomings

The performance analysis presented in Sect. IV allowed us to identify some weaknesses of CoCoA+ in its current definition. We summarize our conclusions in the following, also providing additional considerations based on the gained experience.

A. RTO too close to RTT

When a sequence of similar RTTs is sampled, the RTTVAR variable, which measures the mean variance, tends to vanish. It follows that RTO values get close to the measured RTT. The problem is exacerbated when such RTTs have a very small value, for instance because of a period in which the network is lightly loaded that leads to small RTOs. However, even when the network is stable, RTTs can fluctuate, consequently the

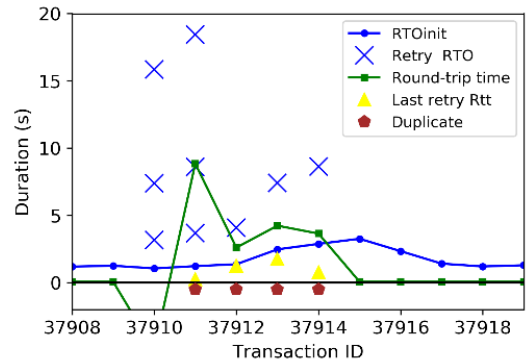


Fig. 12, Weak estimator failing to adapt RTO_{init}.

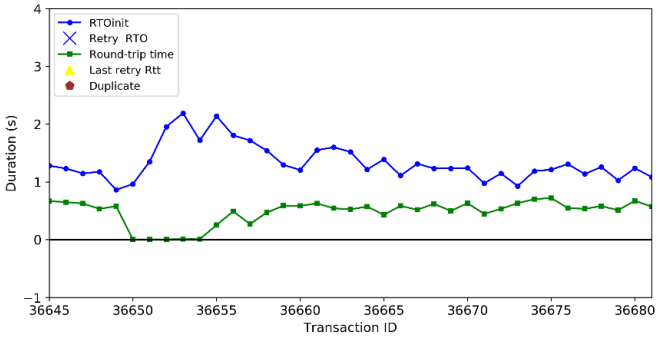


Fig. 13, Example of RTO increase as response to RTT decrease.

probability of spurious retransmissions increases as RTOs get close to small RTTs. The RTTVAR component was intended to maintain information about network variability: when the traffic pattern is bursty (e.g., an alarm triggered by sporadic events), it might be worthwhile to avoid RTTVAR resets.

B. *Lack of weak estimator update*

The rule to avoid the weak estimator update when more than two retransmissions occur has been introduced in CoCoA+ to reduce the impact of the weak RTO on the final RTO. However, this countermeasure can work well only under steady conditions when the network load is constant or changes very slowly. On the other hand, if competing traffic is bursty, a more timeliness adaptation to the current network load is required.

C. *Insufficient weak estimator weight*

Setting K_{weak} to 1 also limits the impact of the weak RTO. This works well under slowly varying traffic load, but not when competing sporadic bursty traffic is transmitted in the network. Our results showed that if the sampled RTT after a retransmission is very close to the RTO, the weak estimator is not able to increase the RTO value effectively. Such event is particularly frequent when spurious retransmissions occur. In this case, even though the received response corresponds to the first transmission, the weak estimator manages the sample as if it were a response to the retransmitted request. However, its weight is smaller than the strong estimator, and therefore it does not change significantly the RTO value, and this may cause a sequence of spurious retransmissions.

D. *RTO peaks in response to RTT decrease*

The algorithm used to calculate the RTO suffers of the following issue: if the RTT rapidly decreases, the RTO, contrarily, increases. As also shown in [14], this can be explained considering that the difference between the smoothed mean RTT and the sampled RTT becomes negative while RTTVAR is updated using the absolute value of such difference, thus resulting in an artificial increase of the RTO. Fig. 13 shows an example of this behavior observed in one simulation run of our experiments.

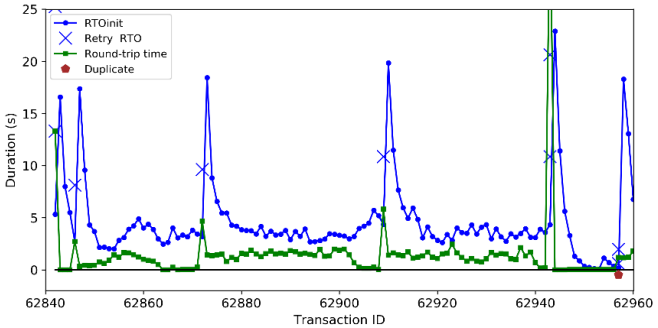


Fig. 14, Example of RTO_{init} spikes.

E. *Double algorithms, double histories*

The analysis of the RTO traces has highlighted some RTO_{init} spikes in some experiments. Fig. 14 shows an example of such behavior: a small increase in the RTT causes a steep increment of the RTO value. The reason for this is due to the lack of an aging mechanism for the weak estimator. If the weak estimator has grown to a large value at some time in the past, it will influence the RTO whenever a retransmission occurs (even if the retransmission is spurious), irrespectively of how much time has passed since then. To avoid such behavior, the weak estimator should be aged as well, or it should be discarded in favor of an exact retransmission response estimation.

F. *Excessive RTO growth*

In CoCoA+ the RTO_{init} value can grow very large. If retransmissions occur, the back-off further increases such values, leading to very large waiting times. In fact, only the initial RTO is limited to 60 seconds. During our experiments some transactions, although a few ones, required a waiting time in between 200 and 300 seconds.

VI. CoAP Precise Congestion Control

The following section presents a set of modifications to the CoCoA+ algorithm that aim at overcoming the issues highlighted. The proposed solution is based on two main elements: (i) a mechanism to precisely link requests to responses even in case of retransmissions; (ii) a set of modifications to the RTO estimation algorithm. Specifically, the resulting *precise Congestion Control* algorithm, p*CoCoA* for short, differs from CoCoA+ in the following points:

- It eliminates the use of the weak estimator, introducing a transmission count option to match each ACK message with the corresponding CON message even in case of retransmissions;
- It introduces a different way to initialize the variable and to update *RTTVAR* and the only smoothed round-trip estimator, *SRTT*;
- It employs a dynamic method to limit the minimum RTO, thus reducing spurious retransmissions.

In the following, we present in detail the proposed modifications. First, we introduce two alternative solutions to implement a precise mapping between CON and ACK messages when retransmissions occur, then we present the precise Congestion Control algorithm and the RTO calculation procedure adopted.

A. *CON-ACK precise mapping*

The current CoAP and CoCoA+ implementations cannot match a specific ACK with its CON when retransmissions occur. As a result, it is impossible to distinguish between transactions completed with only one transmission and transactions that required instead multiple retransmissions. In order to overcome such limitation two solutions are proposed: i) introduce a

```
   function SendAndReceive(CON)
1.  TC = 1
2.  CON.TC = TC
3.  if(CON.dest not in known_dest)
4.     RTO = CoAP_DEFAULT_RTO
5.  else
6.     RTO = known_dest[CON.dest]
7.  RTO = RTO + rand()%RTO/2
8.  ACK = NULL
9.  timestamp[] = Array(MAX_ATTEMPS)
10. while(TC < MAX_ATTEMPS and ACK == NULL)
11.    timestamp[TC] = system_time()
12.    ACK = send(CON)
13.    t = start_timer(RTO)
14.    wait_until(ACK ≠ NULL or t.expired)
15.    if(t.expired)
16.       TC = TC + 1
17.       CON.TC = TC
18.       RTO = VBF()
19. if(ACK ≠ NULL)
20.    R = system_time() − timestamp[ACK.TC]
21.    if(TC ≠ ACK.TC)
22.       ACK.spurious = True
23. else
24.    return
25.    RTO = RTOComputation(R, ACK.spurious)
26.    known_dest[CON.dest] = RTO
```

Algorithm 1: pCoCoA algorithm

specific CoAP option, or ii) modify the Message ID semantic.

In the first case, a new CoAP option called transmission counter (TC) is introduced. This option is used to link each ACK message to the corresponding transmitted CON message even in case of retransmissions. Specifically, the sender of a CON message sets the TC option to 1 on the first transmission, whereas for each subsequent retransmission the value is incremented. The receiver echoes the CON's TC option in the ACK. It is worth to note that the TC option is updated on every retransmission in order to allow to identify which specific CON transmission had generated the ACK.

Based on the TC value in the ACK message, the sender can link the latter to the corresponding CON message. If the sender stores a timestamp for each transmission, it can estimate the round-trip time precisely. Furthermore, this mechanism allows to detect *spurious retransmissions*, i.e. unnecessary retransmission of CON messages that are not lost but simply delayed in the network, by comparing the TC value of the last retransmission with the TC value received in the ACK message. This allows the sender to implement specific policies to react, i.e. by using different factors in the update of the RTO by estimation. The TC option is classified as elective: in this case, when the receiver does not recognize the option, the latter can be silently ignored and an ACK without the TC option can be sent back. The communication is still guaranteed but the advanced congestion control features are disabled.

An alternative solution to avoid the overhead introduced by the transmission of a dedicated CoAP option is to rely on existing fields of the CoAP message, and specifically on the Message ID. The MID is a 16-bit unsigned integer used to detect message duplication and to match messages of type ACK/RST to messages of type CON/NON. To support our approach, MID could be split into a 14-bit *id* and a 2-bit *sequence number*. The *id* sub-field would retain the role of the original MID, thus being the same in all retransmitted CON messages, whereas the *sequence number* would be incremented at each retransmission (supporting then up to four retransmissions). Duplicate message detection is therefore preserved, and, in addition, the sender is able to match each received ACK message to its corresponding CON message among the multiple transmitted ones, in order to precisely estimate the RTO.

Without loss of generality, in the following we adopted the first approach.

B. Precise Congestion Control algorithm

The pseudocode of the proposed congestion control algorithm is presented in Algorithm 1. The code summarizes all the operations to be performed in sequence at each phase of a CoAP transaction: the transmission of a CON, the retransmission, and the reception of an ACK.

At the beginning of each transaction, the internal state is initialized. In particular, the RTO value is initialized to the default RTO value, if the destination is unknown, (line 4) otherwise is set to the last value measured for the least transaction towards the same destination (line 6). In both cases, a dithering technique is applied (line 7) to avoid synchronization effects. After the transmission of the CON, the sender stores the timestamp and sets a retransmission timer based on the current RTO value (line 11-13). The sender waits until the retransmission timer expires or an ACK is received. If the transmission timer expires without receiving an ACK, the CON message is retransmitted. Before retransmitting, however, the sender updates the CON's TC option (line 16) and the RTO using the Variable Back-off (VBF) formulae adopted by CoCoA+ (line 18).

Each CON message is retransmitted until an ACK is received or the maximum number of attempts, MAX_ATTEMPS, is reached (line 10). In the latter case, the CON message is discarded and the procedure ends. When an ACK is received, instead, the algorithm computes the RTT value (*R)* as the difference between the timestamp of the ACK and the timestamp of the corresponding CON (line 20). Subsequently, the value of the TC option received with the ACK is compared with the TC of the last sent CON (line 21). If there is a difference between these two values, a spurious transmission occurred. In order to take this into account in future computations, the spurious flag is set (line 22). Once the RTT is computed, the *RTOComputation* algorithm is executed to update the RTO for future transmissions to the same destination (line 25-26).

C. RTO Computation algorithm

The proposed RTO computation mechanism is inspired by the algorithm included in the Linux TCP congestion control [10], which however cannot be directly applied here since the underlying layer is UDP.

Specifically, the Linux TCP algorithm exploits the estimation of the maximum mean deviance of the RTO ($mdev_{max}$) to avoid issues caused by sudden RTT variations. In detail, $mdev_{max}$ is used to limit the RTO. To this aim, the RTO is maintained constant within a time window proportional to the

function RTOCalculation(*R, spurious*)
1. $SRTT = (1-\alpha)SRTT + \alpha R$
2. **if**$(R < (SRTT - RTTVAR))$
3. $\quad RTTVAR = (1-\gamma)RTTVAR + \gamma|SRTT - R|$
4. **else**
5. $\quad$ **if**$(|SRTT - R| > RTTVAR)$
6. $\quad\quad RTTVAR = (1-\beta)RTTVAR + \beta|SRTT - R|$
7. $\quad$ **else**
8. $\quad\quad RTTVAR = (1-\alpha)RTTVAR + \alpha|SRTT - R|$
9. **if**$(R > SRTT)$
10. $\quad$ **if**$(RTTVAR > mdev_{max}$ for 3 consecutive times)
11. $\quad\quad mdev_{max} =$ average of the last 3 $RTTVAR$
12. $\quad$ **else if**$(RTTVAR < mdev_{max}$ for 8 consecutive times)
13. $\quad\quad mdev_{max} = (1-\beta)mdev_{max} + \beta RTTVAR$
14. **if**(*spurious*)
15. $\quad k = 6$
16. **else**
17. $\quad k = 4$
18. $SRTO = SRTT + \max(k\ RTTVAR,\ mdev_{max})$
19. $RTO_{init} = (1-\delta)SRTO + \delta RTO_{init}$

Algorithm 2: RTOCalculation algorithm

number of TCP segments currently waiting for acknowledgment. Considering that by default CoAP adopts a stop-and-wait approach, as NSTART is set to 1 by default, the algorithm must be modified to deal with the fact that CoAP has at maximum only one outstanding request at a time. The resulting RTO computation algorithm is shown in Algorithm 2. When the first RTT sample R is received, the algorithm performs the initialization of the variables as follows:

$$SRTT \leftarrow R,\ \ RTTVAR \leftarrow \frac{R}{2} \tag{8}$$

$$mdev_{max} \leftarrow max\left(\frac{R}{2}, 250ms\right) \tag{9}$$

$$RTO \leftarrow SRTT + mdev_{max} \tag{10}$$

The formula (10) that initializes the RTO value has been inferred from the results of a set of preliminary experiments, omitted here for the sake of brevity, which aimed at evaluating several possible initialization strategies.

After the first RTT measurement, whenever a new value R is measured, the steps of the *RTOCalculation* function are executed. Its definition employs the following constants whose values are set according to the results available from the literature, $\alpha = \frac{1}{8}$[11], $\beta = \frac{1}{4}$[11], $\gamma = \frac{1}{32}$[10], $\delta = \frac{1}{2}$[3].

The algorithm updates *SRTT* and *RTTVAR* adopting two smoothed filters. Differently from LinuxRTO, which updates *RTTVAR* through an additional filter, *RTTVAR* is updated directly considering two different policies (line 5) to slow down its decrease when significant fluctuations occur. To this aim, when the difference between SRTT and R is greater than the current *RTTVAR* value, *RTTVAR* is updated using the parameter β (line 6), otherwise it is updated using the parameter α (line 8). When α is employed, the historic *RTTVAR* value has a greater weight compared to the case in which β is employed, thus avoiding a fast decrease of *RTTVAR*.

The maximum variability that is experienced is tracked using $mdev_{max}$. Such variable, however, is updated only if RTTVAR increases due to an RTT sample greater than the smoothed RTT (line 9), caused by a sudden increase of the network delay. This choice is motivated by the fact that when the network delay increases, the RTO should be increased consequently. However, in order to limit the influence of sporadic peaks in the RTT estimation, a specific aging mechanism for $mdev_{max}$ is introduced. In particular, $mdev_{max}$ is increased only if its value is smaller than R*TTVAR* for three consecutive times (line 10-11). In such case, $mdev_{max}$ is updated to the average value of the last three *RTTVAR* samples. If $mdev_{max}$, instead, is higher than *RTTVAR* for eight consecutive times, its value is updated using the usual smoothing algorithm to facilitate $mdev_{max}$ to converge to *RTTVAR* (line 12-13). This policy has been designed based on a set of preliminary experiments that are not reported here for the sake of brevity. The rationale behind taking into account both $mdev_{max}$ and *RTTVAR* is that they change over time differently, i.e. *RTTVAR* decreases rapidly when multiple similar RTT values are sampled in a short period, while $mdev_{max}$, requires instead more time to decrease, thus limiting the decrease of the final RTO value.

Finally, the *RTO* value is updated following a two-step procedure as in the CoCoA+ algorithm. First, the estimator $SRTO$ (line 18) is updated, then the new RTO_{init} is computed through a smoothed average of the new $SRTO$ and the old RTO_{init} values (line 19). The computation of $SRTO$ depends on the value of the spurious flag and also includes a technique to limit the minimum $SRTO$ value. The latter mechanism, adopted also by the work in [12], increases the weight of *RTTVAR* when a spurious transmission occurs to allow the $SRTO$ estimator to grow faster, thus limiting successive spurious transmissions.

VII. pCoCoA Performance Evaluation

The pCoCoA algorithm has been evaluated by means of simulations adopting the same methodology presented in Section IV. Similarly, both traffic patterns, i.e. periodic traffic and interfering bursty traffic, respectively, are considered to compare the proposed algorithm against CoAP, CoCoA+ and two new recent algorithms derived from CoCoA+: 4-state-strong [7] and CoCoA-E [14], respectively. In the following, we first introduce the 4-state-strong and CoCoA-E, adopted as term of comparison in the performance evaluation, then we present the results with periodic traffic and interfering bursty traffic.

A. 4-state-strong and CoCoA-E algorithms

The **4-state-strong** algorithm, presented in [7], exploits a 4-state estimator to improve throughput even in lossy networks. Specifically, the 4-state-strong algorithm defines four states, namely 1, 2, 3, 4; each CoAP transaction is assigned one of these states depending on the number of times its message has been retransmitted. Specifically, every time a message is retransmitted, the corresponding state is increased, while if the transaction is completed within the RTO the state is decreased. Each state has a different VBF associated, thus allowing to compute the RTO according to the recent history of transactions. Differently from CoCoA+, which has two different estimators (weak and strong), the 4-state-strong has four different estimators one per each state. The rationale behind this is that if losses are intermittent, e.g. due to wireless interference, less RTT samples should be included in the RTO estimation, otherwise the RTO estimation would be highly affected by channel losses with less correlation with network congestion. With 4-

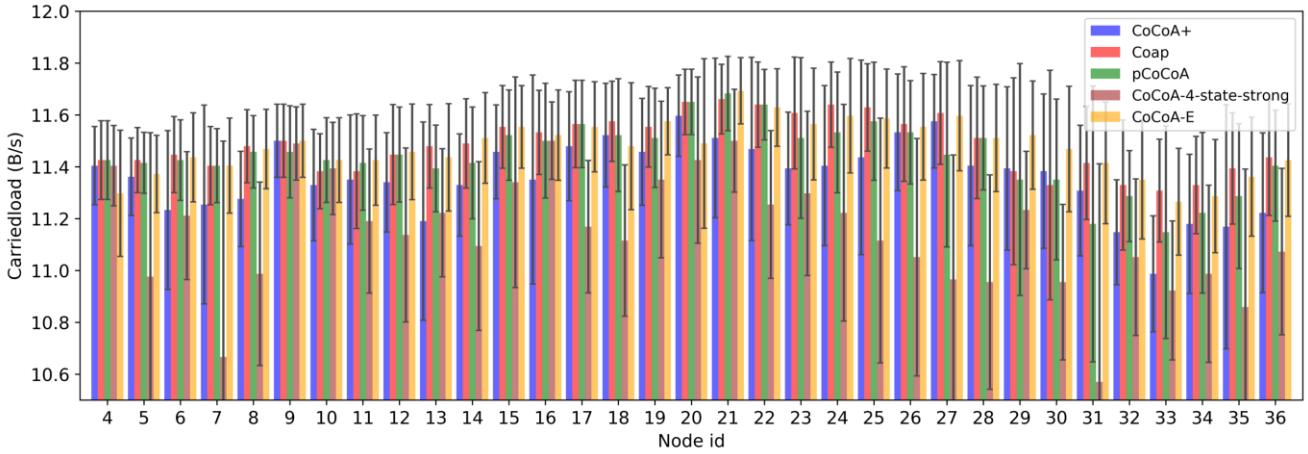


Fig. 15, CoAP vs. CoCoA+ vs. pCoCoA carried load per node, T = 8s.

state-strong, instead, when the number of retransmissions is due to congestion a greater percentage of the measured RTT is included in the final RTO in order to reduce the transmission rate. Specifically, the RTO is computed as follows:

$$RTO_{overall} \leftarrow w * RTO_{obtained} + (1 - w)RTO_{overall} \quad (11)$$

where w is a weight which depends on the transaction state.

The **CoCoA-E** algorithm, presented in [14], takes a different approach that involves the re-definition of the estimators weights and gains. Indeed, CoCoA+ has been designed based on the retransmission timer of TCP, which exploits three different parameters: α, β, and K. In [15] the authors concluded that the estimator gains α and β are too high and the weight K is too low for networks with high load. Specifically they cause SRTT and RTTVAR to decay too rapidly and, consequently, RTO to be set aggressively. For this reason, in CoCoA-E α and β are replaced by a single coefficient γ defined as follows:

$$\gamma_t \leftarrow \frac{RTT}{RTO} \quad (12)$$

$$\gamma \leftarrow \begin{cases} \gamma_t & \gamma_t \leq 0.5 \\ 1 - \gamma_t & otherwise \end{cases} \quad (13)$$

Eq. (13) is introduced to avoid the rapid increase of the RTO due to sporadic losses. When γ is greater than 0.5, RTT is greater than ½ RTO, in this case Eq. (13) avoids a step increment of the RTO, thus avoiding to include in the RTO estimation sporadic losses. In addition, CoCoA-E exploits the strong and weak estimators adopted in CoCoA+, however, with a different strategy for the overall RTO computation. Specifically, if the last RTO value is calculated based on the weak estimator, the same formula adopted in CoCoA+ is used Eq (17) (b), otherwise, if the last RTO is calculated based on the strong estimator, the overall RTO is set equal to the strong RTO value Eq (17) (a). Summarizing:

$$RTT_x = (1 - \gamma)RTT_x + \gamma R \quad (14)$$

$$RTTVAR_x = (1 - \gamma)RTTVAR_x + \gamma|RTT_x - R| \quad (15)$$

$$RTO_x = RTT_x + K_x RTTVAR_x \quad (16)$$

$$RTO_{overall} = \begin{cases} RTO_{strong} & (a) \\ \lambda RTO_{weak} + (1 - \lambda)RTO_{overall} & (b) \end{cases} \quad (17)$$

where $\lambda = 0.5$.

B. *Periodic traffic scenario*

In this section, we report the results obtained with the periodic traffic pattern as described in Section IV.A. Fig. 15 shows the carried load per node when the request period is set to 8s. As can be seen, the pCoCoA and CoCoA-E algorithms perform always better than CoCoA+ and guarantee a performance close to CoAP. The 4-state-strong algorithm, instead, can be compared to CoCoA+ in terms of average performance. However, its results are influenced by a considerably high variance across different runs. Due to their similarities, both CoCoA+ and 4-state-strong suffer from spurious retransmissions that affect their performance. This is confirmed by the data reported in Fig. 16, which shows the average number of retransmissions. Both the CoCoA+ and the 4-state-strong algorithms achieve a similar carried load but with a significantly different number of retransmissions. pCoCoA, instead, reduces spurious retransmissions by leveraging the $mdev_{max}$, variable to compute the lower bound for the computation of RTO. This mechanism reduces the fast decrease of RTO that can occur when multiple similar RTT samples are obtained, thus cutting the number of spurious retransmissions. Indeed, in Fig. 16, the pCoCoA algorithm results in a number of retransmissions that is considerably lower than CoCoA+ and 4-state-strong. The CoCoA-E algorithm achieves a carried load close to pCoCoA but with less retransmissions, this is because the usage of γ allows the RTO to converge to the actual RTT more faster and, under constant periodic traffic, to remain close to the effective RTT value even when retransmissions occur. This, however, does not happen when bursty traffic is considered as shown in the following.

In Fig. 17, we show a detail of the behavior over time of CoCoA+, 4-state-strong, CoCoA-E and pCoCoA algorithms. Specifically, we report on the y-axis the Node IDs and on the x-axis the time. For each retransmission performed by a node, a colored marker is shown. Different colors for the markers are adopted to distinguish adjacent markers from different nodes. Due to a different and more conservative initialization of the state variables, the pCoCoA algorithm shows a reduction of the number of retransmissions at the beginning of the simulation.

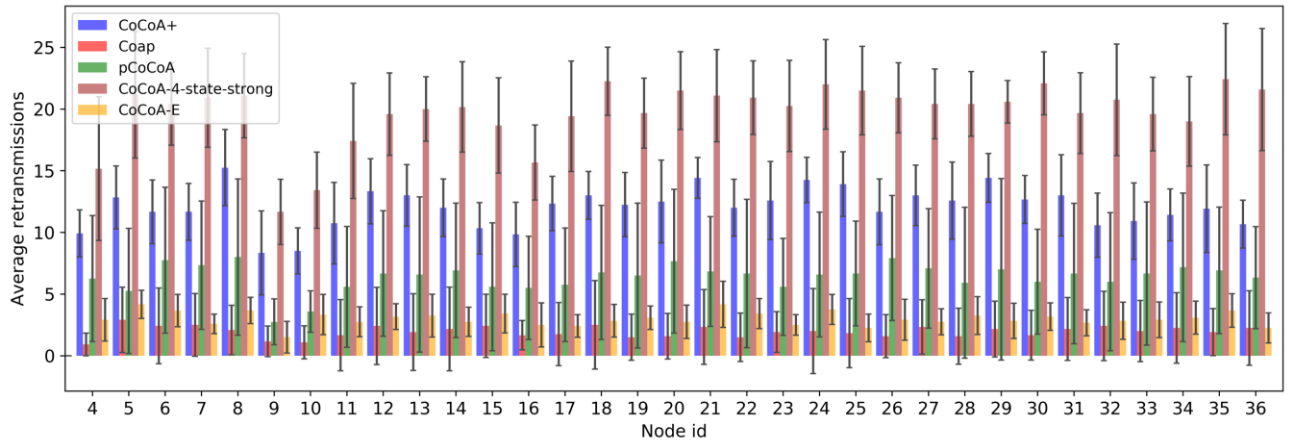
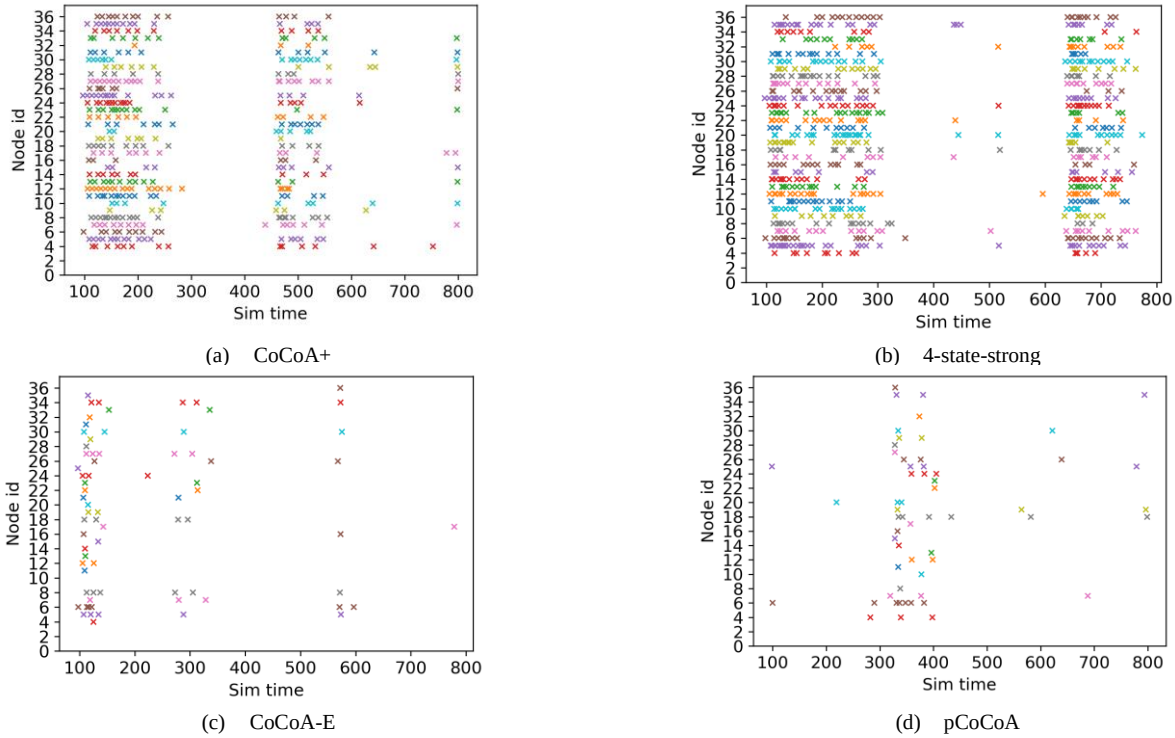


Fig. 16, Node average retransmissions, T = 8s.

On the other hand, CoCoA+ and 4-state-strong require more time to correctly estimate the RTO, thus at the beginning the latter underestimates the RTT value causing spurious retransmissions. The CoCoA-E algorithm, instead, initializes the state variables like CoCoA and thus exhibits a set of retransmissions at the beginning, but subsequently it rapidly converges to the correct RTO. Specifically, CoCoA+ uses the first RTT measurement to set its internal variables. However, if the first RTT sample is too low, several unnecessary retransmissions might be triggered due to the fact that the RTO value remains close to the RTT, as can be seen from the large set of markers around 200 seconds. Consequently, the RTO converges even more slowly because RTT samples have lower weights in the CoCoA+ RTO update algorithm when retransmissions occur.

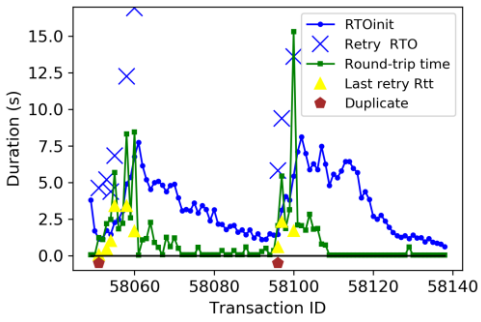
This is highlighted by Fig. 17, where CoCoA+ exhibits a first huge group of retransmissions around 200, and another one around 500 seconds. The latter is smaller compared to the previous one because the estimated RTO is growing, even if slowly. In our experiments, in particular, the CoCoA+ RTO converges to the correct value after 600 seconds of simulation, as can be seen by the absence of groups of retransmissions. The 4-state-strong algorithm in Fig. 17 (b) exhibits a similar behavior. Even if the 4-state-strong algorithm updates RTO based on the number of consecutive losses, at the beginning of the simulation it behaves like CoCoA+. The effects of the different estimators can be seen considering how retransmissions disperse over time. Each group of retransmissions is longer than the retransmission groups resulting from CoCoA+, this is due to the fact that the 4-state-strong algorithm is less aggressive than CoCoA+. As also reported in [7], the reason behind this behavior is that the 4-state-strong algorithm is designed mainly to handle scenarios with a high packet loss due to wireless links, thus in case of losses caused by congestions situations it requires more



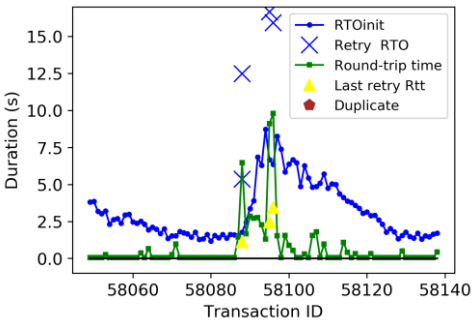
(a) CoCoA+ (b) 4-state-strong

(c) CoCoA-E (d) pCoCoA

Fig. 17, Retransmission dispersion.



(a) CoCoA+



(b) pCoCoA

Fig. 18, Exploitation of the correct *k* value.

time to converge.

Also CoCoA-E, as shown in Fig. 17 (c), presents a first retransmission group around 100 seconds, however, afterwards the RTO estimation converges to the correct value around 300 seconds, thanks to its RTO estimation strategy. This confirms that CoCoA-E converges to the actual RTT quickly under steady conditions, in particular the RTO value is always sufficient to avoid spurious timeout as also outlined in [15].

pCoCoA, instead, is reported in Fig. 17 (d) and, it exploits a more conservative variable initialization that mitigates this issue and converges to the correct RTO value approximately at 400 seconds exhibiting only a reduced set of retransmissions around 350 seconds. Specifically, pCoCoA initializes the RTO based on the measured RTT plus the $mdev_{max}$ contribution, this mitigates the effects of a small RTT at the beginning which, in the case of CoCoA+, initialize wrongly the RTO estimator.

In addition, since the pCoCoA algorithm precisely matches each CON request to the corresponding ACK, it can detect spurious retransmissions and it can compute a more accurate RTT value. Consequently, a more proper value of *k* is selected to estimate RTO. In Fig. 18 we show that CoCoA+ does not detect spurious retransmissions and results in the adoption of the k_{weak} value by relying on the weak estimator. The latter leads to a smaller or similar RTO value that is not sufficient for the subsequent transactions. Specifically, CoCoA+ generates a spurious retransmission for both transactions 58060 and 58100, as highlighted with brown markers. Such spurious retransmissions are due to the fact that the RTO estimation is lower than the actual RTT, when the retransmission timer is fired a retransmission occurs even if the first one is not lost. Consequently, because of retransmissions, CoCoA+ adopts the weak estimator to evaluate the RTO, which reduces the growing rate of RTO generating additional spurious retransmissions. pCoCoA instead converges rapidly to the actual RTT value by detecting spurious retransmissions, this is highlighted in Fig. 18 (b) by a steep increment in the RTO value around transaction 58100.

C. *Interfering Bursty Traffic*

In this section, we report the results obtained with the interfering bursty traffic pattern as adopted also in Section IV.B. In Fig. 19 we report the aggregate carried load obtained with CoAP, CoCoA+, 4-state-strong, CoCoA-E and pCoCoA for the CoAP clients that generate constant periodic traffic. The value is reported as a function of the duration of the ON period. As can be seen, the pCoCoA algorithm performs similarly to CoAP when the burst period is short (less than 60s), while it performs similarly to CoCoA+ when the ON period increases. This can be explained considering that pCoCoA always updates the RTO exploiting a precise mapping between requests and responses, regardless of the number of retransmissions. CoCoA+, instead, exploits the weak estimator that is not updated if more than two retransmissions are detected, thus resulting in periods in which the RTO_{init} is not updated. The 4-state-strong algorithm, instead, updates its estimators also when multiple retransmissions occur. However, by exploiting different weights that take into account the number of retransmissions across transactions, when the ON period increases, the algorithm behaves more aggressively than CoCoA+, thus decreasing the carried load significantly. Finally, CoCoA-E results in poor performance in this scenario. This is mainly because when RTT changes rapidly, for instance at the beginning of the ON period, the algorithm adopts the value $1 - \gamma_t$ of Eq. 13, which limits the contribution of the measured RTT in the RTO computation. In these cases, sporadic packet losses are filtered by the RTO computation in

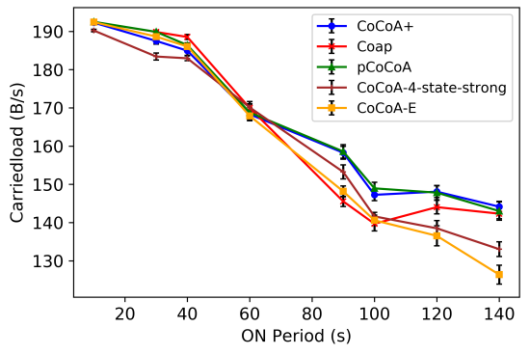


Fig. 19, Carried load – interfering bursty traffic.

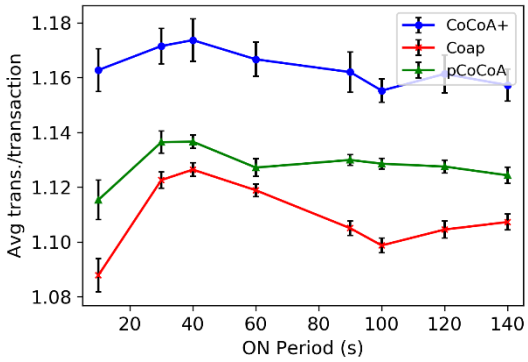
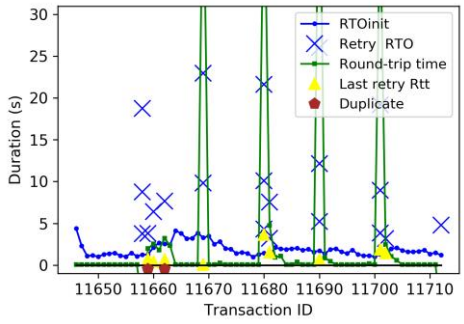
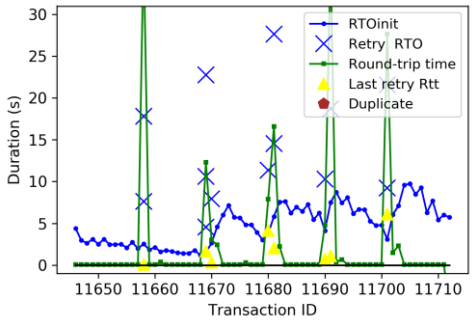


Fig. 20, Transmissions per transaction – interfering bursty traffic.



(a) CoCoA+



(b) pCoCoA

Fig. 21, Weak estimator failing to update RTO_{init} .

CoCoA-E, however, when the ON period is greater than 20 seconds, the value $1 - \gamma_t$ is used frequently, resulting in a slow increase of the RTO that causes frequent retransmissions. Moreover, when a transaction is completed without retransmissions, the overall RTO is immediately updated to the recent RTO value using Eq. 17. However, this results in even lower RTO values during the ON period, further exacerbating the issue.

Even if CoCoA+ and pCoCoA achieve similar performances in terms of carried load, when long bursty periods are considered, the number of transmissions per transaction differs significantly between the two algorithms. This is reported in Fig. 20 that shows the average number of transmissions per transaction. As can be seen, CoCoA+ requires a significantly higher number of transmissions to achieve the same carried load of pCoCoA. As mentioned this can be explained with the fact that CoCoA+ exploits a weak estimator instead of a precise mapping between requests and responses.

In particular, if we analyze a specific run as reported in Fig. 21, we can see that CoCoA+ – Fig. 21 (a) – does not update the RTO when the interfering traffic is ON, which starts in correspondence of the RTT peaks. On the other side, the pCoCoA algorithm – Fig. 21 (b) – can estimate the correct RTT value, thus triggering proper retransmissions, thanks to a continuous update of the RTO_{init} both during the ON and OFF periods of the interfering traffic.

VIII. Conclusions

In this paper, we presented an in-depth analysis of the CoCoA+ advanced congestion control algorithm. Specifically, we considered two different scenarios: one with only CoAP traffic and another one in which CoAP traffic competes with interfering bursty traffic that is transmitted without rate control. The simulation results allowed us to highlight some issues of the algorithm, mostly related to the relationship between RTT estimation and RTO calculation. In particular, we have shown that in certain scenarios CoCoA+ unnecessarily triggers more retransmissions than CoAP, and it does not responsively adapt to network load variations. In order to overcome such shortcomings, a set of modifications to CoCoA+ is proposed.

The resulting algorithm, named pCoCoA, is demonstrated to be effective in guaranteeing a carried load comparable with CoCoA+ in all the presented scenarios, requiring a lower number of retransmissions. Moreover, the comparison also show that, differently from other algorithms, the pCoCoA deals well even in scenarios with interfering bursty traffic.

References

[1] Shelby, Z., K. Hartke, and C. Bormann. "The Constrained Application Protocol (CoAP)." (2014).
[2] Betzler, August, et al. “Congestion control in reliable CoAP communication.” Proceedings of the 16th ACM international conference on Modeling, analysis & simulation of wireless and mobile systems. ACM, 2013.
[3] August Betzler, Carles Gomez, Ilker Demirkol, Josep Paradells, “CoCoA+: An advanced congestion control mechanism for CoAP,” Ad Hoc Networks, Volume 33, 2015, Pages 126-139.
[4] Betzler, August, et al. “CoAP congestion control for the internet of things.” IEEE Communications Magazine 54.7 (2016): 154-160.
[5] Järvinen, Ilpo, Laila Daniel, and Markku Kojo. “Experimental evaluation of alternative congestion control algorithms for Constrained Application Protocol (CoAP).” Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on. IEEE, 2015.
[6] E. Ancillotti and R. Bruno, "Comparison of CoAP and CoCoA+ congestion control mechanisms for different IoT application scenarios," 2017 IEEE Symposium on Computers and Communications (ISCC), Heraklion, 2017, pp. 1186-1192.
[7] Bhalerao, Rahul, Sridhar Srinivasa Subramanian, and Joseph Pasquale. “An analysis and improvement of congestion control in the CoAP Internet-of-Things protocol.” Consumer Communications & Networking Conference (CCNC), 13th IEEE Annual. IEEE, 2016.
[8] Bormann, C., Betzler, A., Gomez, C., & Demirkol, I. (2017). “CoAP simple congestion control/advanced.” Working Draft, IETF Secretariat, Internet-Draft draft-bormann-core-cocoa-01.
[9] Paxson, V., Allman, M., Chu, J., & Sargent, M. (2011). Computing TCP's retransmission timer (No. RFC 6298).
[10] Sarolahti, Pasi, and Alexey Kuznetsov. "Congestion Control in Linux TCP." USENIX Annual Technical Conference, FREENIX Track. 2002.
[11] Jacobson, Van. "Congestion avoidance and control." ACM SIGCOMM computer communication review. Vol. 18. No. 4. ACM, 1988.
[12] A. G. A. Elrahim, H. A. Elsayed, S. E. Ramly and M. M. Ibrahim, "Improving TCP congestion control for wireless sensor networks," 2011 4th Annual Caneus Fly by Wireless Workshop, Montreal, QC, 2011, pp. 1-6.
[13] S. Bolettieri, C. Vallati, G. Tanganelli, E. Mingozzi, Highlighting Some Shortcomings of the CoCoA+ Congestion Control Algorithm, Proceedings of the 16th International Conference on Ad Hoc Networks and Wireless (ADHOC-NOW 2017), Messina, Italy, September 20-22, 2017.
[14] Balandina E., Koucheryavy Y., Gurtov A. (2013) Computing the Retransmission Timeout in CoAP. In: Balandin S., Andreev S., Koucheryavy Y. (eds) Internet of Things, Smart Spaces, and Next Generation Networking. Lecture Notes in Computer Science, vol 8121. Springer, Berlin, Heidelberg.
[15] R. Ludwig and K. Sklower “The Eifel Retransmission Timer”, ACM SIGCOMM Computer Communication Review, Vol. 30, Issue 3, pp. 17-27, July 2000.
[16] A. Betzler, C. Gomez, I. Demirkol and J. Paradells, "CoAP congestion control for the internet of things," in IEEE Communications Magazine, vol. 54, no. 7, pp. 154-160, July 2016.