On the design of efficient congestion control for the Constrained Application Protocol in IoT

Nabil Makarem, Wafaa Bou Diab, Imad Mougharbel, Naceur Malouch

Version of Record: https://www.sciencedirect.com/science/article/pii/S1389128622000457
Manuscript_d4d5d940c491e2cdc57a63f4ee7d8923

On the Design of Efficient Congestion Control for the Constrained Application Protocol in IoT

Nabil Makarem[a,b], Wafaa Bou Diab[b], Imad Mougharbel[c], Naceur Malouch[a]

[a] *LIP6, Sorbonne Université, France*
[b] *Lebanese University, Lebanon*
[c] *École de Technologie Supérieure - GREPCI Lab, Canada*

Abstract

The Constrained Application Protocol (CoAP) is one of the main candidates for a lightweight communication protocol for the Internet-of-Things. CoAP provides a simple congestion control mechanism based on successive retransmissions and binary exponential timeouts. This simple mechanism can significantly reduce CoAP performance especially in networks with high packet loss, and thus preventing an efficient deployment of the protocol. Enhanced mechanisms for CoAP were proposed in the literature. Some considered improving retransmission timeout estimation whereas others focused on augmenting the retransmission procedure. In this work, we analyze deeply main and recent proposals to highlight their shortcomings. Then, we propose and implement two congestion control algorithms IDC-CoAP and MBC-CoAP which improve retransmission timeout estimation for congestion detection, and adopt adequately a rate-based approach for congestion counteraction, while maintaining simplicity required by constrained devices. The two proposed algorithms are evaluated by means of pure simulations considering several network scenarios, and also using the realistic environment Cooja/Contiki. All results show that our algorithms achieve a much better tradeoff between goodput, reliability and overhead.

Keywords: Internet of Things, CoAP, Congestion Control, Rate Performance, Reliability, Contiki, Simulation, Cooja.

1. Introduction

The internet of Things (IoT) devices have raised new challenges in protocol design and standards. The Internet Engineering Task Force (IETF) has been developing new specifications that are featured for constrained devices in the IoT field. In particular, the Constrained Application Protocol (CoAP) [1] was designed for data transmissions between applications of IoT devices. Other protocols such as Message Queuing Telemetry Transport (MQTT) and Advanced Message Queuing Protocol (AMQP) have been designed also for communications, however, CoAP is becoming more and more used for constrained IoT devices and is suitable for different IoT fields such as healthcare, smart cities, agriculture and transportation [2, 3, 4, 5]. CoAP is supposed to be more efficient in energy consumption and bandwidth utilization, and it should be appropriate for high packet loss networks [6] due to its reduced overhead. Nevertheless, recent studies demonstrate that it is still necessary to improve the congestion control mechanism of CoAP to improve further its performance in terms of reliability and efficiency [7, 8]. Since constrained devices suffer from limited resources and processing capacities, congestion occurs when the node's traffic load exceeds its available capacity, and/or when high traffic is generated during the communication between large number of nodes. Other network related reasons are involved when some mobile networking technologies embrace sudden transmission delay spikes in addition to classic failures or disasters. CoAP standard provides a simple congestion control mechanism to handle network losses which is based on a simple send-and-wait protocol with a binary exponential backoff procedure in case of packet loss, i.e. no acknowledgement (ACK) is received [1]. Other related works have analyzed congestion control of CoAP and proposed improvements in calculating the retransmission timeout and/or the backoff procedure [9, 10, 11]. In this paper, CoAP and previous enhanced congestion control algorithms are analyzed to show their properties and shortcomings. Then, we propose further substantial improvements to congestion control algorithms for CoAP based on the concept of rate control in order to overcome previous limitations and enhance the tradeoff between reliability and rate performance. The resulting protocols, named IDC-CoAP and MBC-CoAP, are evaluated against protocols from the literature to show concretely their efficiency. The contributions of this work are summarized as follows:

- Profound analysis and performance evaluation of

*Corresponding author
Email addresses: nabil.makarem@lip6.fr (Nabil Makarem), wafaa.boudiab@ul.edu.lb (Wafaa Bou Diab), imad.mougharbel@etsmtl.ca (Imad Mougharbel), naceur.malouch@lip6.fr (Naceur Malouch)

several previous congestion control algorithms, supported by the development of an original simulator dedicated to CoAP testing.

- A new algorithm for retransmission timeout reduction to enhance congestion detection.
- Integration of two new congestion control rate-based algorithms in CoAP instead of the less efficient backoff procedure while keeping the complexity reasonable for constrained devices.
- Performance evaluation results using both our *ad hoc* simulator and the well-known Cooja/Contiki IoT environment show that our rate-based congestion control for CoAP achieves a better tradeoff between reliability, transmission overhead and bandwidth utilization.

The remainder of the paper is organized as follows. Literature review is summarized in Section 2. Section 3 describes CoAP and several previous congestion control algorithms. Then, Section 4 presents our new simulator environment and exposes the main issues in previous backoff-based and rate-based algorithms. In Section 5, the design of our proposed algorithms IDC-CoAP and MBC-CoAP for congestion control are presented. Performance evaluation results are presented in Section 6 where we compare our algorithms with previous works. Section 7 concludes the paper and presents some directions for future work.

2. Related work

Recently, there have been works conducted to evaluate and improve CoAP performance and particularly its congestion control. In the following, a summary review of research literature is presented which is relevant to this paper.

In [12], the authors suggested adjusting CoAP parameters to handle high traffic and high loss probability. They experimentally adjusted the constant value of the retransmission timeout (RTO) and the randomization factor of the initial timeout (ACK_RANDOM_FACTOR). They adopted new CoAP congestion control parameters based on experiments which is not sufficient since experiments are limited and usually do not cover different network conditions. Also, using smaller static backoff factors leads to packet losses because the sender will transmit faster during congestion periods.

The work in [13] proposed a congestion control scheme based on round trip time (RTT) measurements through a counter using the option field of the CoAP message. The sender recognizes the origin of an ACK and calculates the correct RTT to update the retransmission timeout (RTO) without smoothing. Therefore, instead of using the default RTO CoAP parameter (2 sec) for the retransmissions when ACK is not received, the authors use the new calculated RTO for the retransmissions.

CoAP Simple Congestion Control Advanced CoCoA [14] performs RTO calculations based on TCP RTO computation algorithm where smoothed RTT is used to update the RTO value automatically using two estimators weak and strong. They showed that the performance of CoCoA is better than CoAP in congested networks but different studies [9, 10] show that it has some limitations with bursty traffic. Recent works [9] have suggested different modifications in CoCoA RTO estimation in order to reduce spurious transmissions which refers to packets that are retransmitted because of incorrect estimation of RTO.

The authors in [15] evaluated another version of CoCoA named CoCoA-S using only the strong estimator. They employed alternative algorithms developed for TCP which are shown to be inadequate for CoAP. CoCoA-S is conservative in lossy networks which results in low throughput. The main algorithm CoCoA delivers better performance in comparison to CoCoA-S.

CoCoA+ [10] have adopted the same RTO estimation mechanism presented in [14] with minor updates on the weak estimator weights. The authors also introduced a backoff policy to set the timeout for the retransmissions named Variable Backoff Factor (VBF) to replace the binary exponential mechanism used by default in CoAP. In addition, they added an ageing mechanism to set RTO values if RTT is not updated for a certain time.

Other research works suggested improvements to CoCoA. The work named 4-state [11] differentiates between four states and each state was given a weight to be used when a loss is detected. Using small different weights for the backoff factors will improve the goodput because transmitting will be faster but losses will increase. Besides, the number of the algorithm parameters increased significantly and no study was conducted to tune them carefully.

An adaptive mechanism for handling congestion in CoAP called Fast-Slow RTO (FASOR) is proposed in [16]. RTO computation is composed of fast and slow RTO calculation. Fast RTO is based on TCP retransmission timer and updated with unambiguous RTT samples (where ACK messages matches CON messages) while slow RTO is measured from the original transmission of a packet till the arrival of its ACK regardless of the required number of re-transmissions. Fast RTO is used as a sign of link error (interference) whereas slow RTO is a sign of heavy congestion. The authors evaluated their algorithm among CoAP and CoCoA+ using Netem [17] as network emulator and libcoap library [18]. The authors tried to deduce whether a packet loss is due to link disruption or due to congestion by examining RTT samples (ambiguous or not), however, RTT analysis can hardly be used to detect the reasons behind a packet loss. Also when the network is lossy, the backoff mechanism will lead to a behavior similar to CoAP due to the exponential backoff mechanism. They also set the retransmission counter to 20 instead of 4 which is too excessive and will cause long packet delays and a very low throughput which might be useful only when packet delivery must be guaranteed on the behalf of all other perfor-

mance metrics.

The latest version of CoCoA, named pCoCoA, is proposed in [9]. The retransmission timeout calculation is based on TCP implementation in Linux kernel. The backoff mechanism is the same as in CoCoA+. Evaluation showed some improvement of the RTO calculation using several weights to estimate the round-trip time variance. However, many instructions are used which increases the processing overhead in constrained devices. pCoCoA will be analyzed in detail in Section 3.1.

The previous works mentioned above follow what we call the backoff-based approach since they act on the retransmission timeout and its evolution during the backoff period. The authors of BDP-CoAP [19] implemented in CoCoA+ the congestion control of TCP BBR protocol [20] which follows a rate-based measurement-based approach. BDP-CoAP will be analyzed in detail in Section 3.2. In summary, in addition to BBR components, BDP-CoAP incorporates some modifications in an attempt to enhance fairness and to adapt to CoAP and CoCoA+ constraints. Not only the resulting protocol is very complex, but we also show that BDP can outperform previous backoff-based protocols in few cases where the network conditions are prosperous.

3. CoAP and Previous Congestion Control Algorithms

CoAP [1] is a lightweight protocol designed for constrained devices in IoT networks which is based on Representational State Transfer (REST) that supports basic operations like GET, POST, PUT and DELETE. Its similarity with HTTP, the protocol of the World Wide Web of the Internet, eases interoperability and integration of different objects and systems in the Internet. CoAP is also suitable for IoT devices with limited hardware capacities. Compared to other protocols [21] used for constrained devices such as Message Queue Telemetry Transport (MQTT) and Advanced Message Queuing Protocol (AMQP), one major difference is that CoAP relies on UDP for communication and hence it is immune to all overheads and problems related to connection management especially in case of packet loss in wireless networks [6]. Thus, the complexity, imposed by other transport protocols, is eliminated in CoAP. As UDP is inherently not reliable, CoAP should provide its own reliability mechanism which can be designed to be very simple. CoAP communications can be selected to use either confirmable (CON) or non-confirmable (NON) messages. The standard CoAP [1] uses a simple mechanism for congestion control based on retransmissions with a binary exponential backoff (Fig. 1).

When a packet containing a CON message is lost, the client re-sends the message at doubled increasing intervals, until an acknowledgement (ACK) is received or the allowed number of attempts is reached. Two parameters control the retransmission process: An initial timeout value RTO_{init} and a retransmission counter r. For each

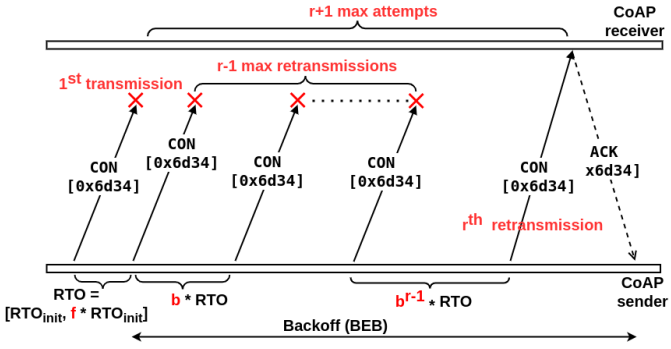


Figure 1: CoAP Congestion Control. Retransmission counter: r=4, retransmission timeout: RTO_{init}=2s, Random factor: f=1.5, Backoff factor: b=2

new CON message, the first timeout is set to a value between RTO_{init} and RTO_{init} times a randomization factor f, and the retransmission counter is initialized to 0. If the sender did not receive an ACK for the CON message, CON is re-transmitted and the re-transmission counter is incremented and the timeout value is doubled. The whole process for each CON is repeated until the counter reaches its limit or an ACK is received by the sender. In the following, we present the main previous proposals to enhance congestion control for CoAP. We classify them into two categories: backoff-based and rate-based.

3.1. Backoff-based Congestion Control

CoCoA (Congestion Control/Advanced) is an enhanced backoff mechanism for CoAP primitively proposed in [14] based mainly on changing RTO_{init} dynamically rather than the default constant value of 2 seconds. Further various modifications have been implemented over the past years to improve CoCoA performance. The improvements in pCoCoA [9] consist of the following main aspects:

- A mechanism to calculate the initial retransmission timeout based on the TCP retransimission timeout algorithm version implemented in Linux kernel [22].

- A variable backoff mechanism to set the RTO for retransmissions adopted from CoCoA+ [10]

The RTO_{init} calculation in pCoCoA follows almost the same algorithm as the one implemented for TCP in the Linux Kernel with additional instructions. RTO_{init} value is updated according to RTT measurements. RTO_{init} calculation in pCoCoA tries to handle two problems: First, when RTT increases suddenly and causes RTO_{init} overestimation. Second, when RTT variance (RTTVAR) drops to a small value leading to spurious retransmissions. pCoCoA algorithm introduces the maximum mean deviation ($mdev_{max}$). The parameters are initialized when the first corresponding RTT value R is received. When a new value R is computed, RTO_{init} is updated for the future transmissions. SRTT and RTTVAR are updated using different

Algorithm 1 pCoCoA RTO calculation: block 1

$SRTT = (1-\alpha)SRTT + \alpha R$
if $R < (SRTT - \text{RTTVAR})$ **then**
  $\text{RTTVAR} = (1-\gamma)\text{RTTVAR} + \gamma|SRTT - R|$
else
  if $|SRTT - R| > \text{RTTVAR}$ **then**
    $\text{RTTVAR} = (1-\beta)\text{RTTVAR} + \beta|SRTT - R|$
  else
    $\text{RTTVAR} = (1-\alpha)\text{RTTVAR} + \alpha|SRTT - R|$
  end if
end if

weights to slow down the sudden decrease when fluctuations happen (block 1 in Algorithm 1 pCoCoA).

When there is a sudden increase in the network delay, $mdev_{max}$ with aging mechanism is introduced to increase RTO value accordingly (block 2).

Algorithm 1 pCoCoA RTO calculation: block 2

if $R > SRTT$ **then**
  if $\text{RTTVAR} > mdev_{max}$ *for* 3 *consecutive times* **then**
    $mdev_{max} = average\ of\ the\ last$ 3 RTTVAR
  else if $\text{RTTVAR} > mdev_{max}$ *for* 8 *consecutive times* **then**
    $mdev_{max} = (1-\beta)mdev_{max} + \beta\text{RTTVAR}$
  end if
end if

If spurious transmission is detected, k is set to 6, otherwise, it is set to 4 (block 3).

Algorithm 1 pCoCoA RTO calculation: block 3

if (*spurious*) **then**
  $k = 6$
else
  $k = 4$
end if

Lastly, RTO_{init} is updated using another smoothed variable SRTO. SRTO calculation is based on the spurious flag k in order to limit the minimum SRTO values. Then, RTO_{init} is computed using another weighted sum that combines SRTO and previous RTO_{init} as per block 4.

The successive retransmission timeouts in the backoff mechanism of pCoCoA are based on the maintained RTO_{init} and also on several values of backoff factors used to multiply the timeouts instead of 2 (doubling) in case of successive losses. pCoCoA adopts the same CoCoA+ backoff mechanism, called Variable Backoff Factor (VBF), for setting retransmission timeout values. The value of VBF is chosen from a list [1.5, 2, 2.5] according to RTO_{init} value. The RTO for the backoff period is defined by:

$$RTO_{backoff} = RTO_{init} \times VBF, \text{ where} \quad (1)$$

Algorithm 1 pCoCoA RTO calculation: block 4

$SRTO = SRTT + max(k * \text{RTTVAR}, mdev_{max})$
$RTO_{init} = (1-\delta)SRTO + \delta RTO_{init}$

$$VBF = \begin{cases} 2.5, & RTO_{init} < 1 \text{ sec} \\ 2, & 1 \leq RTO_{init} < 3 \text{ sec} \\ 1.5, & RTO_{init} > 3 \text{ sec} \end{cases} \quad (2)$$

In 4-state [11], the RTO calculation is very similar to the one in CoCoA, however a more complex procedure is used to determine backoff factors. Each transaction is assumed to be in one of four states depending on the number of times a packet was retransmitted. Four different variable backoff factors are used - VBF1, VBF2, VBF3 and VBF4 where each variable factor corresponds to a different state.

3.2. Rate-based Congestion Control

The congestion control mechanism in most algorithms is backoff-based. However, few works tried to propose an alternative to backoff-based congestion control. The authors of BDP-CoAP [19] adopted the idea of TCP BBR protocol [20] in order to control the rate of CoAP transmissions. Instead of using mainly packet loss (three duplicate ACK reception) to infer the congestion, TCP BBR estimates the Bandwidth-Delay Product and determines the maximum number of packets in flight to not exceed in order to prevent losses. The Bandwidth-Delay Product is computed by estimating the round trip propagation delay and the available bandwidth through several measurements. In particular, an available bandwidth measurement is obtained at the reception of every ACK. A max filter is used to stabilize the estimated available bandwidth over a sliding time window.

$$\widehat{AvaiBw} = max\,(MeasBW_t) \quad \forall t \in [T - W_B, T]$$
where W_B is a time window
and T is the current time

TCP BBR stops sending packets when the number of packets in flight, i.e. packets that have not received yet their acknowledgements, is larger than the Bandwidth-Delay Product so that the bottleneck queue does not grow up more and thus buffer overflow is prevented.

The estimated max-filtered bandwidth is also used to control the sending rate through an eight-phase cycle with the use of pacing gains. Each phase corresponds to a packet transmission as shown in Fig. 2. At each phase, the sending rate is set to the estimated bandwidth multiplied by the pacing gain. In the first six phases of the cycle, the pacing gain is equal to 1. Then in the seventh phase it is set to 5/4 to increase the sending rate and probe for the available bandwidth. However, if in this phase, some losses had occurred, then the pacing gain is set to 3/4 to reduce the sending rate. In the eighth phase, the pacing gain is set to 3/4 in a preventive approach in case the probing of the bandwidth is not successful and also to empty

any resulting queue. The values 5/4 and 3/4 are chosen so that the average sending rate during the two probing and preventive phases does not change from other phases: $\left(\frac{5}{4}+\frac{3}{4}\right)/2 = \frac{8}{4}/2 = 1$. According to the authors of TCP BBR [20], this cycling scheme, allows BBR flows to achieve high throughput, low queuing delay, and convergence to a fair share of bandwidth.

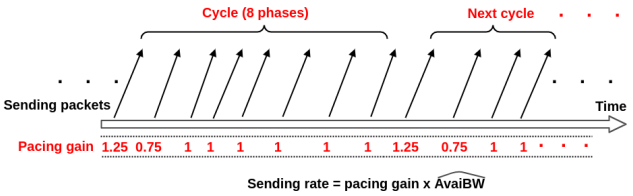


Figure 2: The eight-phase cycle scheme in TCP BBR

BDP-CoAP implements all the components proposed by BBR for TCP congestion avoidance with additional differences. First, BDP-CoAP computes the pacing gain factor through a ten-phase cycle instead of eight. Second, the pacing gain values of the probing phase and the preventive phase are 1.2 and 0.8 instead of 1.25 and 0.75 respectively. Other pacing gains are equal to 1 as in TCP BBR. Third, neither the pacing gain nor the estimated bandwidth are updated in case of retransmission. Indeed, when an ACK of a retranssmitted CON is received, the bandwidth measurement that can be done using this ACK is canceled and the estimation function is not called. Fourth, the filter used to compute the estimated bandwidth uses, in addition to the maximum of previous measurements, the minimum of these previous measurements. The min and the max are combined together with a weighted sum. Fifth, the time window used to compute the filtered estimated bandwidth is removed. Instead, the filter considers the last 10 measurements done at the last 10 packet sending instants. Naturally, losses are still detected via RTO_{init} expiration but its value is not multiplied by a backoff factor in case of successive losses. RTO_{init} is estimated exactly as in CoCoA+.

4. Analysis and Shortcomings of Previous Congestion Control Algorithms

4.1. Ad hoc Simulation Environment

We developed in Python language a dedicated simulator to analyze deeply CoAP algorithms in a controlled environment. Indeed, the simulator can generate different patterns of Round Trip Times (RTT) or reuse real RTT traces. It is able to emulate the available bandwidth observed by a CoAP sender and also emulate loss of packets. Hence, congestion and its strength and duration are emulated in a repeatable and supervised manner. The evolution of network conditions over time can be kept exactly the same from one simulation to another so that comparison between different algorithms is fair. Besides, compared to other simulation environments, our Python simulator produces results much faster. Hence, it is possible to test quickly many network scenarios and CoAP protocol variants.

4.2. Analysis and Shortcomings of pCoCoA and Backoff-based Congestion Control

There are two components in backoff-based Congestion Control algorithms: RTO_{init} calculation for the first retransmission and a backoff mechanism for the remaining re-transmissions. For RTO calculation, if RTO is less than the Round Trip Time RTT, then the packet is falsely retransmitted due to incorrect RTO estimation causing a spurious transmission. If RTO is much larger than RTT, then the sender will wait unnecessarily causing a degradation in terms of goodput and delays of packets delivery.

4.2.1. Spurious Transmissions

In pCoCoA, when RTT increases suddenly to a higher value, RTO increases because of the smoothed RTT (SRTT) value which is based on the average of the measured RTTs. However, after few transactions with lower RTTs, RTO decreases very closely to RTT which leads to spurious transmissions. In Fig. 3, we simulated such scenario where we plot the evolution of RTT values and RTO values over time. When RTT decreases suddenly, pCoCoA RTO converges quickly and this is risky because the sudden decrease might be followed by a sudden increase which leads to spurious transmissions as shown in the figure (bold green x-points at sequence numbers 451 and 452). A better design should handle RTO convergence gradually when RTT decreases as per the blue plot in the figure.

4.2.2. Large RTO Estimations

It could be simple to estimate RTO much greater than RTT to avoid spurious transmissions. However, the larger RTO, the lower the goodput and the longer the transmission delay. Also, the reaction to congestion may take more unnecessary time. Ideally, RTO should be as close as possible to RTT values without triggering spurious transmissions. Particularly, when RTT increases suddenly as simulated in Fig. 3, RTO is calculated by pCoCoA after the sudden increase of RTT but not fastly enough to avoid spurious transmissions. In such case, RTO must converge quickly as per the blue plot to avoid spurious transmissions but without exceeding that much RTT values as pCoCoA is behaving. Note that RTT values can increase suddenly due to severe congestion or other factors such as burst connection arrivals or handoffs in wireless networks.

4.2.3. Inaccurate Variable Backoff

pCoCoA adopts the same variable backoff mechanism used in CoCoA+. In this mechanism, the backoff factor changes depending on the previous value of RTO_{init}. However, it should be rather set depending on the congestion

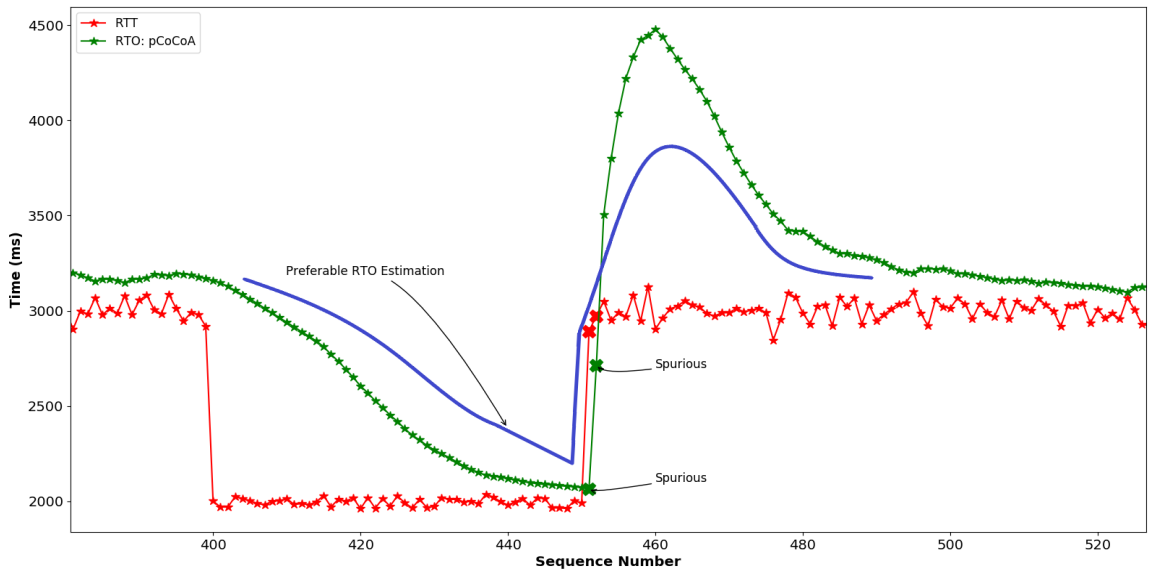


Figure 3: pCoCoA RTO Estimation vs Preferable RTO Estimation

state of the network. The retransmission timeout is useful to detect congestion but it is not sufficient to choose a backoff factor that relates to the congestion level in the network. For example, if RTO_{init} value is 1.5 seconds which might be close to RTT, and the ACK message was not received, in this case the backoff factor will be 2 according to pCoCoA backoff algorithm, then the sender will unnecessarily wait for 3 seconds. A large RTT, and thus a large RTO does not signify a congestion. Besides, when RTT values are less than 1 second in the network, then if a packet is lost, the backoff factor that will be used is 3! The waiting time will be a lot increased which affects the performance in terms of goodput and delay. The negative impact is even worse if the loss is due to interference or short congestion state. Actually, all the efforts done to reduce and optimize RTO_{init} are vanished when we use large backoff factors.

4.2.4. Complexity

pCoCoA adopts a congestion control mechanism similar to the one implemented in Linux TCP [22]. Although Linux TCP is used by many network applications in the internet, its complexity makes it not efficient for constrained devices that are limited in storage and processing capabilities. Especially, our simulations show that the block for calculating $mdev_{max}$ in Algorithm 1 pCoCoA block 3 is being executed up to 70% in each simulation but not being used in RTO_{init} final calculation except in few cases as shown in Fig. 4. We tested 29 different RTT network scenarios (x-axis) which are detailed later in Table 1 Section 6.1. The blue bars show the calculation for $mdev_{max}$ if RTTVAR is greater than mdev for 3 consecutive times, while the red bars present the calculation $mdev_{max}$ when RTTVAR is less than mdev for 8 consecutive times. The yellow bars present the usage of $mdev_{max}$ when calculating SRTO. Although it is calculated in all the transactions, $mdev_{max}$ is rarely used. This overhead might seem negligible with normal machines but it increases energy consumption and overhead computation in a constrained IoT environment for a negligible benefit.

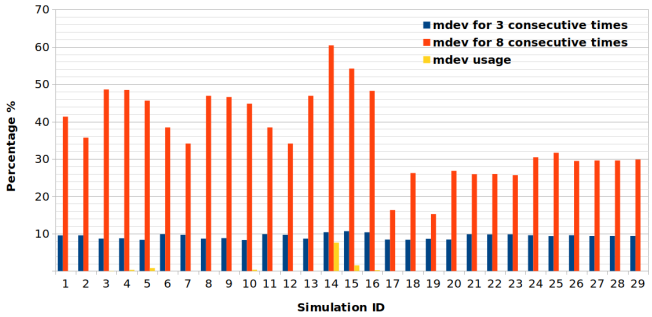


Figure 4: $mdev_{max}$ *usage by pCoCoA*

4.3. Shortcomings of Previous Rate-based Algorithms

The congestion avoidance algorithm proposed in TCP BBR for congestion control follows a measurement-based strategy to detect congestion and to set the sending rate adequately, instead of a more classic loss-based strategy. In this regard, the BBR congestion control can be very efficient since it aims at equating the sending rate to the available bandwidth which is somewhat the ultimate objective of any congestion control. However, it must be

judicially adapted to be incorporated in the CoAP protocol which has specific properties and is destined to specific devices and network environments. It turns out that the adaptations proposed by BDP-CoAP have several shortcomings presented in the following.

4.3.1. Bandwidth Sampling Inaccuracy

BBR is designed for TCP Congestion Avoidance periods where usually the bandwidth is very high and the number of packets sent and ACK received, is very high as well, resulting in a lot of measurement samples to estimate the available bandwidth quickly and precisely. In CoAP, the sending rate is 1 message per RTT or lower and hence the number of bandwidth measurement samples is very low. As a consequence, in contrast to what is proposed by BDP-CoAP, each sample must be considered in the estimation especially those obtained at the reception of an ACK of a retransmitted CON message. These ACKs reflect also successful transmissions and bandwidth availability and must be considered. Besides, after several successive retransmissions, which means losses, the first ACK received will provide an actual measurement on the new decreased available bandwidth that causes the losses. Ignoring samples from retransmitted packets will lead to inaccurate or nonexistent bandwidth estimation if there are losses in the network. Fig. 5 simulates a case of bandwidth sudden decrease showing the inability of BDP-CoAP to decrease its sending rate due to successive losses despite that many ACKs are received.

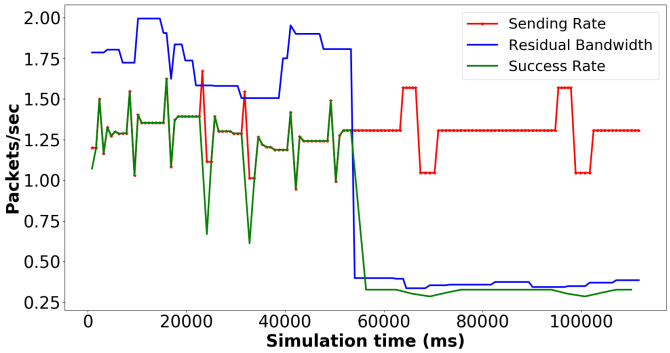


Figure 5: BDP-CoAP inefficiency in case of sudden bandwidth decrease.

4.3.2. Inadequacy of the Bandwidth Filter Time Window

For the same reason, sliding the bandwidth measurement window over *time* is not adequate to filter the measured samples because after several losses and/or sending rate reduction, the time window will not be able to cover enough number of samples and this number can even be drawn to zero which blocks totally the protocol. BDP-CoAP uses a window that slides on the instants of sending attempts instead of time. However, this procedure does not solve the problem because in case of losses, the attempts continues and makes the window sliding further, which yet removes past measurements from the filter but without adding new ones.

4.3.3. Bandwidth Delay Product Inapplicability

Similar to TCP BBR, BDP-CoAP computes the Bandwidth Delay Product and uses it to control the number of CoAP CON messages to send without waiting for their acknowledgements during an RTT. However, the CoAP concept is based on sending only one packet per RTT (NSTART=1 [1]) to keep its operation simple and avoid using a sending window and all algorithms for its management as TCP. With this constraint, packets inflight is either 0 or 1, and the Bandwidth Delay Product is always between 0 and 1. Even, if we allow NSTART to be more than one, the Bandwidth Delay Product might still be small in IoT environments due to low link data rates and small buffers.

4.3.4. Bandwidth Estimation filter Degradation

Including the minimum of the bandwidth measurement samples in the estimation filter is not adequate in terms of goodput maximization especially if the bandwidth is variable. The minimum was introduced as an attempt to improve fairness, however, the impact on the goodput is very harmful. Indeed, the minimum will slow the convergence to the maximum available bandwidth. Furthermore, in the filter, the minimum is associated with a weight that is related to the number of retransmissions. The more the retransmissions, the higher the weight, the slower the convergence. Fig. 6 shows indeed the inability of BDP-COAP to converge reasonably when the available bandwidth increases suddenly. The wastage of the bandwidth is huge.

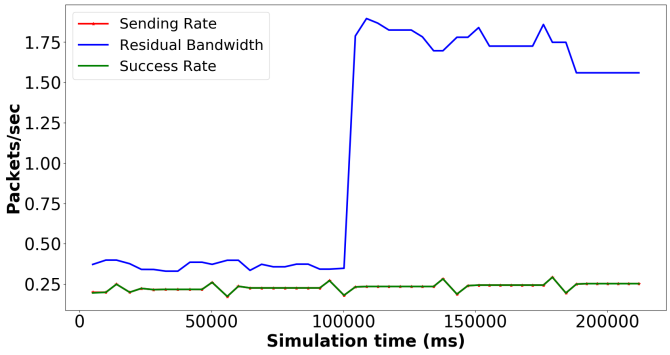


Figure 6: BDP-CoAP inefficiency in case of sudden bandwidth increase.

4.3.5. Complexity

Even more than TCP BBR, BDP-CoAP uses many variables, instructions and function calls in order to perform bandwidth measurements and processing them. Hence, the algorithm becomes too complex. As a matter of fact,

all simulations done in the BDP-CoAP work [19] have used a non-constrained type of devices while the employed simulator was designed especially for constrained devices. Again, the objective of having a good compromise between efficiency and complexity can not be ignored in IoT environments. This complexity can be reduced by removing unnecessary components that were designed for TCP and not really useful for CoAP. Also, one can use a different approach of congestion control other than a measurement-based.

5. Proposed algorithms: IDC-CoAP and MBC-CoAP

In this section, we propose three new mechanisms in order to overcome the previous shortcomings that we have discussed above and improve the overall performance of CoAP. First, we design a new algorithm to calculate the initial retransmission timeout RTO_{init} more efficiently to improve congestion detection. Then, we design two algorithms for congestion counteraction that both follow the rate-based approach. The outcome is two new proposed protocols called IDC-CoAP and MBC-CoAP. The former is based on Increasing/Decreasing the sending rate. The latter uses a Measurement-Based approach to control the sending rate and it is inspired from the recent BBR congestion control [20]. In both cases, RTO_{init} is used only to detect losses and it is never doubled or modified in case of loss, thus we call it also simply RTO.

5.1. Congestion Detection: RTO Calculation

The challenge in RTO calculation is the fast convergence during high fluctuations in the network such as sudden increase and sudden decrease of RTT. Precisely, if the CoAP sender uses a smaller RTO than RTT, it will generate a spurious transmission and the sending rate will be greater than 1 packet per RTT which may worsen the situation in case of congestion. Also, RTO should not be much higher than RTT because the sender will wait long before detecting the loss and before sending the lost and the next packets. The sender might also skip some good time intervals where packets can be successfully delivered. Another challenge is the limited capacities of some IoT constrained devices where the algorithm must be developed with minimal instructions so it can reduce the processing overhead and increase the lifetime of the constrained devices. That is why also spurious transmissions must be reduced as much as possible to reduce energy consumption.

Our new algorithm is still based on the concept of exponentially weighted moving average (EWMA) with several modifications from previous works to minimize further RTO while reducing the number of spurious transmissions, and in the same time reduce the number of instructions. The pseudo code is presented in Algorithm 2. SRTT is the maintained average round trip time and RTTVAR is the computed variation of RTT. SRTT is updated using the weight α (line 1). RTTVAR is updated using the weights α and γ (lines 2 to 6).

Algorithm 2 RTO calculation algorithm

```
1:  SRTT = (1 − α)SRTT + αR
2:  if R > SRTT then
3:     RTTVAR = (1 − α)RTTVAR + α|SRTT − R|
4:  else
5:     RTTVAR = (1 − γ)RTTVAR + γ|SRTT − R|
6:  end if
7:  if spurious is true then
8:     K = 7
9:  else
10:    K = 4
11: end if
12: RTO = SRTT + K ∗ RTTVAR
```

Once a first RTT value is measured, it is stored in the variable R. Then, $SRTT$ is initialized to R, and $RTTVAR$ to $R/2$. Afterwards, RTO is estimated for each new R as shown in Algorithm 2. In this algorithm, we update RTTVAR using a different weight depending on the value of the measured RTT R compared to the maintained average $SRTT$. This condition ($R > SRTT$) is necessary to adapt RTO estimation adequately to RTT fluctuations, on the behalf of some additional complexity. However, this condition was sufficient to estimate RTO correctly and the algorithm is still simpler than pCoCoA. The weights are fixed so that convergence is faster ($\alpha > \gamma$) when R increases ($R > SRTT$) because if convergence is not fast enough, there is a high risk to have $RTO < R$ causing spurious transmissions which is important to avoid utmost as we said before. In contrast, convergence is slower when R decreases ($R < SRTT$) in a preventive approach in order to observe first if this reduction is permanent or transient. Otherwise, if we converge fast, the estimated RTO can flip down below next R values causing again spurious transmissions. The weights used to compute RTTVAR should be tuned to find a good compromise between RTO reduction and spurious reduction as well.

Finally, in the algorithm, RTO for the next transmission is calculated using the smoothed value of RTT $SRTT$ to which we add the RTT variance $RTTVAR$ multiplied by margin factor K (lines 7 to 12). The concept of using $K \times RTTVAR$ is based on Jacobson/Karels algorithm (Timeout = Estimated_RTT_Average + 4 × Estimated_RTT_Deviation) [23]. Another challenge though is choosing the right value for K. K plays an important role in estimating RTO value. When it is set to 4 for all transmissions, performance can be reduced. Indeed, K should be preferably chosen dynamically according to the spurious status. If the previous transmission is spurious, then RTO value must be increased by increasing K to force the sender to wait for a longer period (lines 7-8). A lower K value should be used when spurious is not detected (lines 9-11).

In order to choose a good combination of values for the parameters α, γ and K that support our design objectives, different values in different network scenarios were tested and analyzed. We show here only results with one of the most challenging scenarios where RTT is varied continuously according to a uniform distribution between 500ms and 600ms during 100 times, then a sudden change to a normal distribution with mean 100ms and a standard deviation of 20ms during 100 times, then a sudden return to the uniform distribution and so on and so forth. Besides, a high increase of RTT to 1 second is generated 5 times every 1000 transmissions. Then, while varying the weights, we compute both the total number of spurious transmissions and the Root Mean Square Error (RMSE) which measures the difference between estimated RTO and measured R values.

According to the results from the figures 7 and 8, when we fix the value of α, then a low value of γ reduces both spurious transmissions and RMSE. In Figure 9, when γ is fixed to 1/32, a value of 1/8 for α is a good compromise between spurious and RMSE. Thus we fix $\alpha = \frac{1}{8}$ and $\gamma = \frac{1}{32}$. This result confirm our design rule $\alpha > \gamma$.

Regarding the K parameter, in Fig. 10, we fix K to 7 if there is a spurious transmission and we vary the value of K in the case when there is no spurious from 2 to 7. We observe that the higher these K values, the lower RMSE and number of spurious. However, the value of 4 is a good compromise since after this value, the improvement is minor compared to the improvement from the values of 2 to 4. In Fig. 11, we fix K to 4 if there is no spurious transmission and we vary the value of K in the case when there is a spurious from 2 to 7. We observe that the value of 7 reduces both spurious and RMSE. Thus the portion of the algorithm that computes K is useful to reduce further RMSE and the number of spurious transmissions. All other simulation results, not shown here, has confirmed this setting.

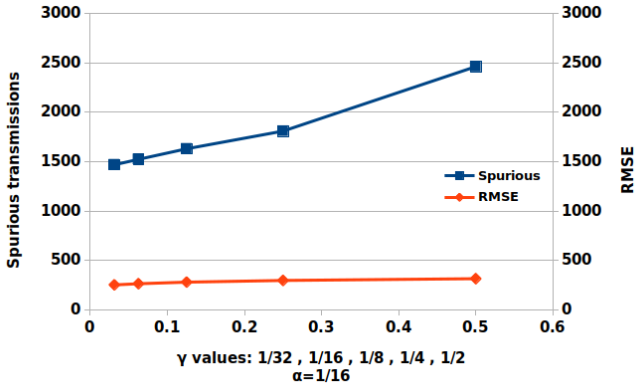


Figure 7: $\alpha = \frac{1}{16}$

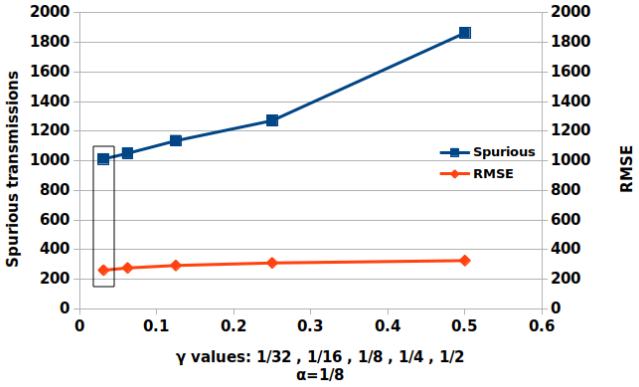


Figure 8: $\alpha = \frac{1}{8}$

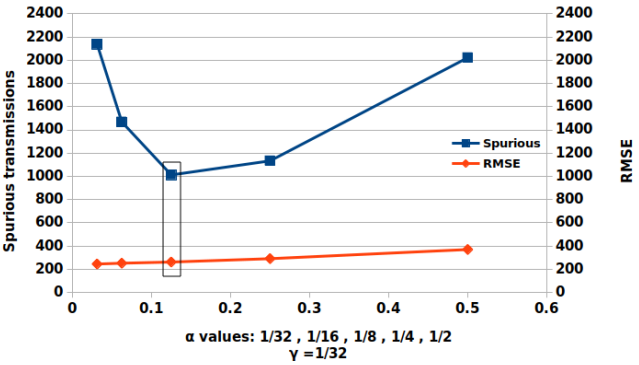


Figure 9: $\gamma = \frac{1}{32}$

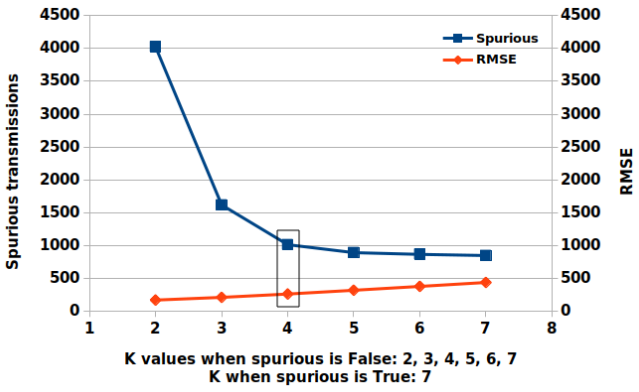


Figure 10: K = 7 when there is a spurious

5.2. *Congestion Counteraction: Rate-Based Control Algorithms*

Most of congestion control algorithms for CoAP follow the backoff based approach where they try to use a different static backoff factor or use a variable backoff factor according to some conditions. We claim that the backoff mechanism is not sufficient to leverage the bandwidth available to the CoAP sender. It is better to deploy a "real" congestion control mechanism in order to decide correctly how long to wait before sending the next packet

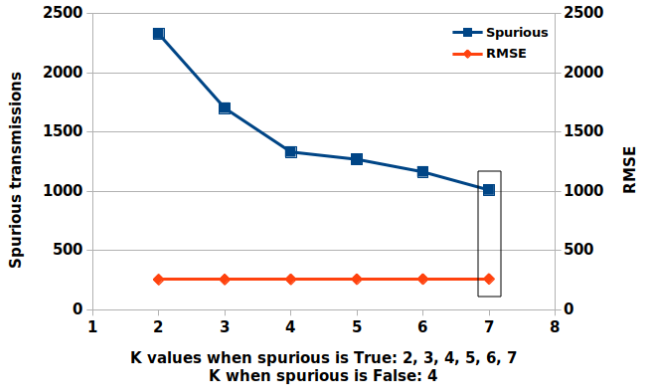


Figure 11: K = 4 when there is no spurious

and avoid losses. The inter-sending delay should be inversely proportional to the available bandwidth, and this is hard to achieve through the backoff mechanism even with variable backoff factors.

Evidently, if one wants to maximize the throughput achieved by the CoAP connection, the sender can send aggressively at its maximum rate to ensure a maximum throughput. However this will engender a lot of packet losses, retransmissions and possibly losses of CoAP messages at the application layer if many successive packets are lost. More substantially, it will also waste a lot of energy due to wasted transmissions. This is a different constraint from classic congestion control where losses are not harmful if they are recovered quickly. Here the damage of a packet loss is irreversible and should be avoided. The challenge is that in order to check if the bandwidth is available or not, the only way for the CoAP sender to do is to send a packet, but if this packet is sent while the bandwidth is not available then it will be lost. Retransmission of the lost packet does not reduce the incurred cost related to energy consumption. As a consequence, when a timeout expires indicating that the bandwidth is not available, determining the right time to wait before sending the retransmission is crucial. A short time may cause additional losses and a long time may reduce dramatically the goodput. It is clear that this time should relate to the available bandwidth.

Hence, the main idea is to remove entirely the backoff mechanism from CoAP and integrate a new mechanism that determines the adequate *spacing* between successive transmissions including retransmissions regardless of the RTO value or the retransmission counter. It is essential though to keep this algorithm as simple as possible. That is why our first proposal is based on the simple Additive Increase Multiplicative Decrease of the transmission rate even though we apply it to the time spacing between packets. We call the resulted protocol IDC-CoAP. The second proposal follows the measurement-based approach developed in BBR where the transmission rate is determined based on available bandwidth measurements and periodic probing of the bandwidth. We call this protocol version MBC-CoAP. Different from BDP-CoAP, MBC-CoAP adapts more adequately BBR to the existing properties of CoAP and avoids all BDP-CoAP design inaccuracies.

5.2.1. IDC-COAP

In this version of the protocol we aim at keeping the control algorithm as simple as possible by simply following the Additive Increase Multiplicative Decrease principle to control the rate with two differences. The first consists on working on the spacing between successive packets instead of the rate. The second consists on adding a phase of fast rate increase to benefit more from the available bandwidth in case it opens up. When the CoAP sender transmits a packet, there are two main network events from its point of view:

- An ACK is received. It means that the sending rate $\leq$ residual bandwidth. The current time *spacing* between packets can be decreased to increase the rate.
- The RTO expires. It means in case of congestion that the sending rate $>$ residual bandwidth. The current time *spacing* should be increased to decrease the rate.

Algorithm 3 IDC-CoAP pseudo-code

```
1:  Wait for CoAP ACK or RTO expiration
2:  if ack is true then
3:      if spacing ≥ loss_spacing then
4:          spacing = spacing − dw ∗ spacing
5:      else
6:          spacing = spacing − fw ∗ spacing
7:      end if
8:  else
9:      loss_spacing = spacing /* Save congestion level */
10:     spacing = iw ∗ spacing
11: end if
12: spacing =
    max(spacing, current_time − last_send_time)
13: Send next packet (transmission or retransmission) at:
    last_send_time + spacing
```

The pseudo code of the control algorithm is presented in Algorithm 3. When a loss is detected, the sending rate should be decreased. Therefore, the spacing is increased by multiplying it by the incremental weight iw (line 10). When ACK is received, the spacing is reduced gradually using the decremental weight dw (lines 2 and 4). When spacing becomes lower than the spacing saved at the loss event (line 5) which corresponds to the last known available bandwidth, then spacing is reduced with a higher decremental factor fw in order to find quickly the new possible expanded available bandwidth (line 6). The maximum function is invoked to make sure that the sender can not send before the reception of the next ACK or the expiration of RTO (line 12). In the performance evaluation

section, we show that a good combination of the algorithm parameters is: incremental weight $iw = 1.5$, decremental weight $dw = 0.01$ and fast decremental weight $fw = 0.5$. These parameters were chosen to achieve a good tradeoff between goodput and loss ratio. However, they can be easily tuned when the application requires better goodput on behalf of losses and energy consumption, or vice versa.

5.2.2. MBC-COAP

In this version of the protocol, we follow the measurement-based approach implemented in BBR to compute the *spacing* between packet sending instants. We adopt the same concept of the max-filtered estimation of the available bandwidth and the same values for the probing and preventive pacing gains, i.e. 1.25 and 0.75 respectively. We also use the same length of the pacing cycle and the same update procedure. However, in order to overcome the shortcomings mentioned earlier (Section 4.3), we do not estimate neither the bandwidth delay product nor the minimum round trip propagation delay. We also do not maintain the packets in flight. These components are unnecessary for CoAP so removing them simplifies the protocol. Besides, the window used in the bandwidth estimation filter slides each time the CoAP sender receives an ACK. Thus, we compute the sending rate based on the maximum of the last m measurements, with m being the size of the sliding window. Which means, instead of using a time window, we use a space window. This modification is especially useful in high lossy environments. Importantly, in contrast to BDP-CoAP, we include each received ACK in the estimation of the bandwidth including those corresponding to retransmissions so that the number of measurement samples is sufficient to estimate more precisely the available bandwidth and converge to it rapidly. We simplified further the algorithm by removing all function calls as presented in the pseudo-code of Algorithm 4.

In the algorithm, we measure the spacing between successive received ACKs as an estimation of the "available spacing" which is inversely proportional to the available bandwidth. The last m spacing measurement samples are maintained to be used for computing the sending spacing (lines 2-7). After an ACK reception or an expiration of a timeout, MBC-CoAP sends the next packet according to the previously computed spacing (line 8). The next spacing for the next sending is computed by calculating first the next pacing gain (lines 9-20), then calculating the minimum of the last m spacing measurements corresponding to the maximum of bandwidth samples (line 21). Even though this algorithm is much simpler than the one in TCP BBR and BDP-CoAP, it is still more complex than IDC-CoAP due to the need of maintaining measurements.

Algorithm 4 MBC-CoAP pseudo-code

```
1:  Wait for CoAP ACK or RTO expiration
2:  if ack is true then
3:      measurement_sample =
        current_time − last_ACK_time
4:      last_ACK_time = current_time
5:      Add measurement_sample to Spacings[m]
6:      Remove oldest measurement sample from
        Spacings[m]
7:  end if
8:  Send next packet (transmission or retransmission) at:
    max(last_send_time + spacing, current_time)
9:  cycle_index = (cycle_index + 1) % 8
10: if cycle_index = 0 then
11:     if retransmission then
12:         pg = 0.75
13:     else
14:         pg = 1.25 /* Probing phase */
15:     end if
16: else if cycle_index = 1 then
17:     pg = 0.75 /* Preventive phase */
18: else
19:     pg = 1
20: end if
21: spacing = min(Spacings[m])/pg
```

6. Performance evaluation

In this section, we use our python simulator presented in section 4.1, to first evaluate the efficiency of our algorithm of RTO calculation, then, to study and compare the performance of the full congestion control algorithms of IDC-CoAP and MBC-CoAP against other previous congestion control algorithms. Second, we use Cooja/Contiki [24] to complement the comparison between rate-based and backoff-based approaches in a more realistic IoT environment.

6.1. Congestion Detection: RTO Calculation

Our new algorithm of RTO estimation used by both IDC-CoAP and MBC-CoAP is compared with the one included in pCoCoA since it was proven to be better than other previous RTO calculations [9]. Table 1 shows RTT network scenarios that will be used in this evaluation.

In most of the scenarios, RTT is varied continuously according to a Uniform distribution between two values during 100 times (Period 1), then a sudden change to a Normal distribution with a given mean and a given standard deviation during 100 times (Period 2), then a sudden return to the Uniform distribution and so on and so forth. The total number of transmissions is 100000 in all scenarios. Fig. 12 shows scenario 6 where we switch from a high RTT average with a large deviation to a low average with a low deviation. Besides, to challenge more the algorithms, in some simulations, we can have additional events corresponding to a high increase of RTT to 10 seconds generated 5 times every 1000 transmissions. Scenarios 2 and 3 use RTTs from real measurements between two sites. Scenario 2 is from Paris in France to Auckland in New Zealand, and scenario 3 is from Paris to Rennes in France.

ID	Network scenario				
	Period 1		Period 2		Additional events
	Number of Transmissions	RTT Distribution	Number of Transmissions	RTT Distribution	
1	100000	Pareto (4, 1000)	N/A	N/A	No
2	100000	Real trace	N/A	N/A	No
3	100000	Real trace	N/A	N/A	No
4	100	U(100,300)	100	N(3000,100)	No
5	100	U(100,1000)	100	N(3000,100)	No
6	100	U(100,300)	100	N(3000,1000)	No
7	100	U(100,2000)	100	N(3000,100)	No
8	100	U(300,3000)	100	N(6000,200)	Yes
9	100	U(300,3000)	100	N(4000,200)	Yes
10	100	U(200,1000)	100	N(6000,200)	Yes
11	100	U(100,2000)	100	N(6000,200)	Yes
12	100	U(100,500)	100	N(4000,200)	Yes
13	100	U(100,500)	100	N(4000,1000)	Yes
14	100	U(100,500)	100	D(1000)	Yes
15	100	U(100,500)	100	D(6000)	Yes
16	100	N(6000,200)	100	U(200,1000)	Yes
17	100	N(6000,200)	100	U(300,3000)	Yes
18	100	U(300,3000)	100	N(4000,2000)	Yes
19	100	U(100,2000)	100	N(6000,2000)	Yes
20	100	U(300,3000)	100	N(6000,200)	No
21	100	U(300,3000)	100	N(4000,200)	No
22	100	U(200,1000)	100	N(6000,200)	No
23	100	U(100,2000)	100	N(6000,200)	No
24	100	U(100,500)	100	N(4000,200)	No
25	100	U(100,500)	100	D(6000)	No
26	100	N(6000,200)	100	U(200,1000)	No
27	100	N(6000,200)	100	U(300,3000)	No
28	100	U(100,300)	100	N(3000,300)	No
29	100	U(100,1000)	100	N(3000,300)	No

Table 1: Different simulation scenarios to challenge RTO calculation algorithms

Fig. 13 shows RTTs over time of the real trace of scenario 2.

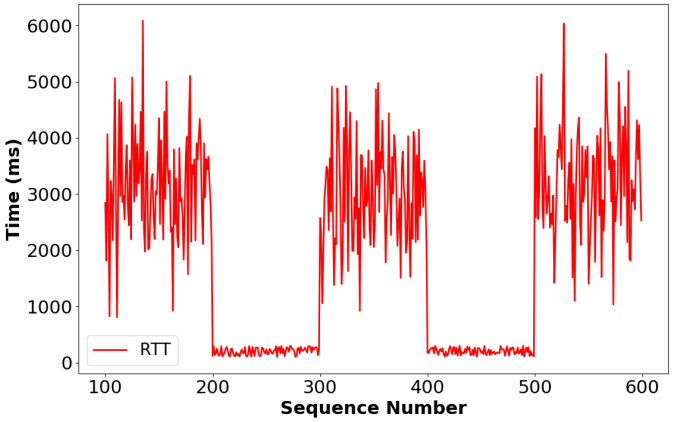


Figure 12: Scenario 6: High average/high deviation to low average/low deviation and vice versa

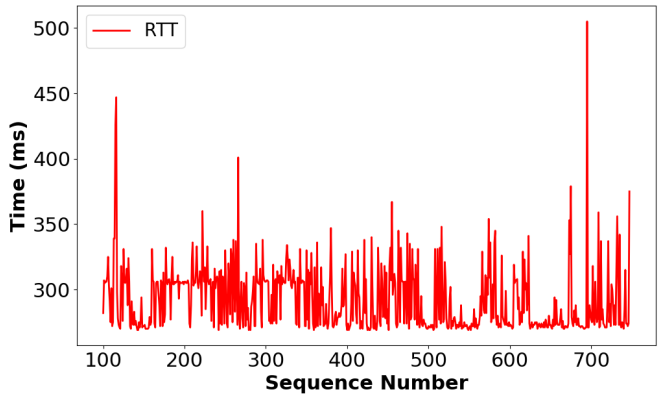


Figure 13: Network scenario 2: Example of real RTT data set

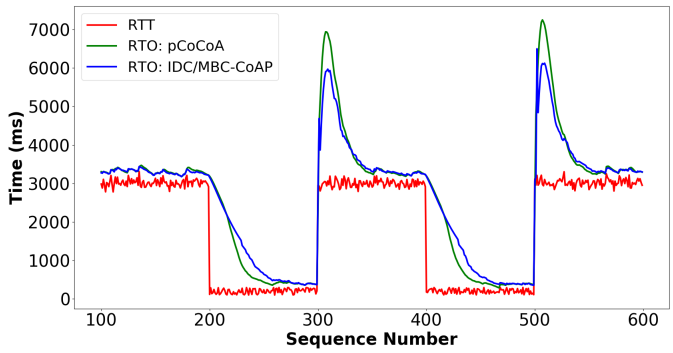


Figure 14: RTO calculation - Scenario 4

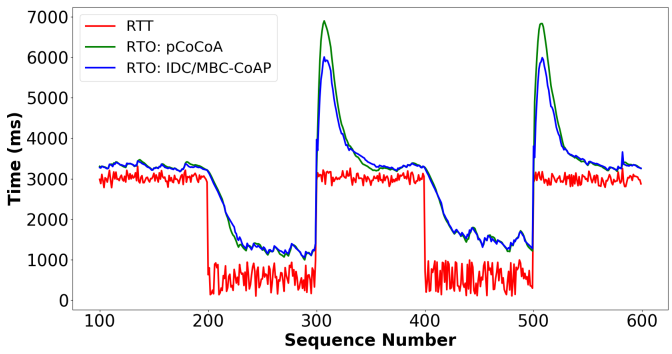


Figure 15: RTO calculation - Scenario 5

First, we study the instantaneous behavior of both algorithms using scenarios 4 to 7 mentioned in Table 1. These scenarios cover all possibilities regarding fluctuations of RTT average and variance. Figures 14 and 15 show that when observed RTT increases suddenly in the network, then RTO in IDC/MBC-CoAP also increases quickly to avoid spurious transmissions due to underestimations of RTO. However, pCoCoA RTO increases more than required which leads to more delay in packet retransmissions. Besides, this increase is not quick enough to avoid spurious transmissions as shown more clearly in Fig. 22 that corresponds to the same network scenario as Fig. 14.

Figures 14 and 16 show a better convergence behavior of IDC/MBC-CoAP RTO than pCoCoA when RTO decreases to smaller values with low variations. pCoCoA decreases faster which is risky because it can cause spurious transmissions as shown above. Our RTO calculation converges in a slower manner to be cautious and prevent spurious transmissions. Figures 15 and 17 show that when RTO decreases to smaller values but with high variations, then both RTO calculation algorithms are similar but still our algorithm reacts better to sudden increase by provid-

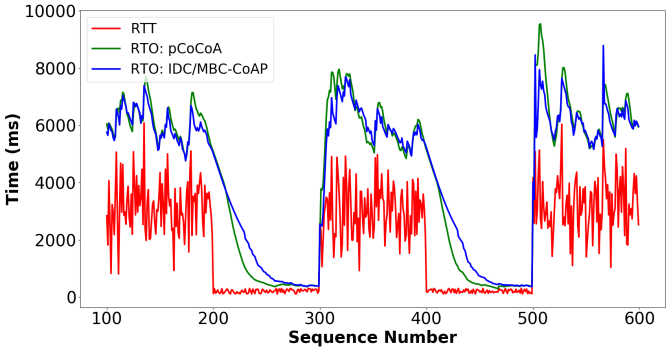


Figure 16: RTO calculation - Scenario 6

ing less spurious and low delay.

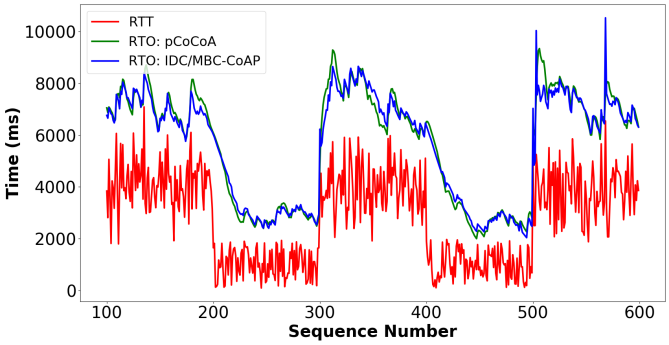


Figure 17: RTO calculation - Scenario 7

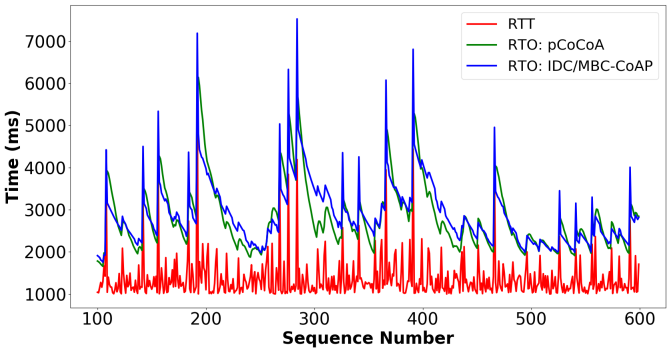


Figure 18: RTO calculation - Pareto distribution

We observe the similar better behavior in Fig. 19 where RTT values in the python network simulator correspond to real RTT measurements (Scenario 2 - Table 1). Fig. 18 shows the same results with the *Pareto* distribution for RTT values. By examining closely these figures, we can see that our design principle is still applied by converging fastly but not too high when RTT increases fastly, and by converging relatively slowly when RTT decreases fastly to avoid as much as possible spurious transmissions while trying to minimize RTO values. Indeed, using the python simulator, we were able to calibrate the different parameters of our RTO calculation to reach this design principle as mentioned in Section 5.1.

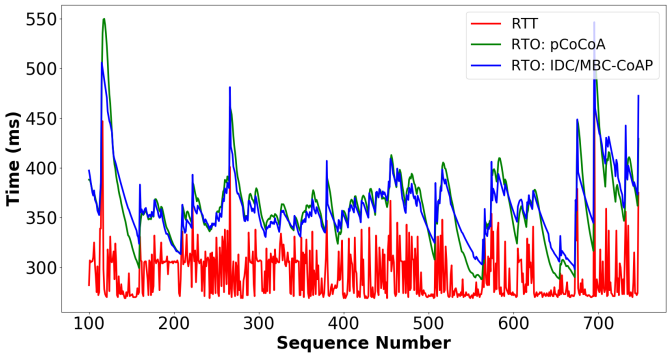


Figure 19: RTO calculation - Real RTTs

Next, the global behavior of both algorithms are evaluated using two performance metrics: The Root Mean Square Error (RMSE) which measures the difference between RTO and RTT values, and the total number of spurious transmissions observed during the network simulation.

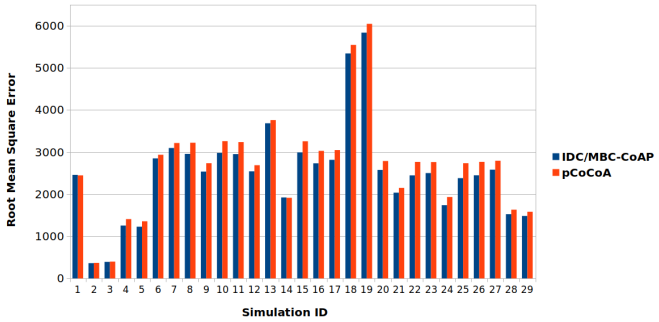


Figure 20: RMSE

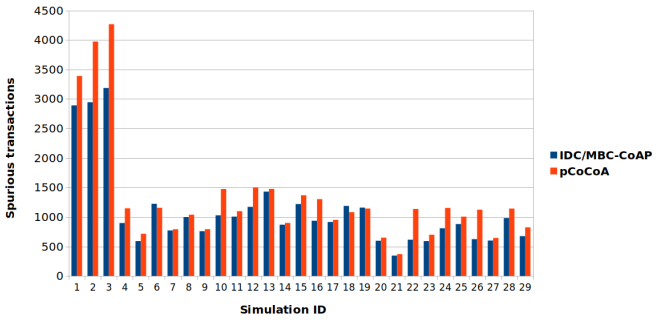


Figure 21: Spurious transmissions

Fig. 20 and Fig. 21 show the average of spurious transmissions and RMSE values calculated in different scenarios of Table 1. For instance, simulation ID 1 refers to Pareto

distribution, simulation IDs 2-3 refer to real RTT scenarios, simulation IDs 4-7 refer to RTT scenarios analyzed instantaneously above.

According to Fig. 20 and Fig. 21, our proposed RTO calculation algorithm provides lower RMSE and lower number of spurious transmissions in almost all the network scenarios. In the case RMSE of pCoCoA is very close to IDC/MBC-CoAP (IDs 1,2,3,14), pCoCoA generates more spurious transmissions. In the case pCoCoA generates few spurious transmissions less than IDC/MBC-COAP (IDs 6,18), RMSE of pCoCoA is worse. Thus, our RTO calculation achieves a better tradeoff between the two performance metrics.

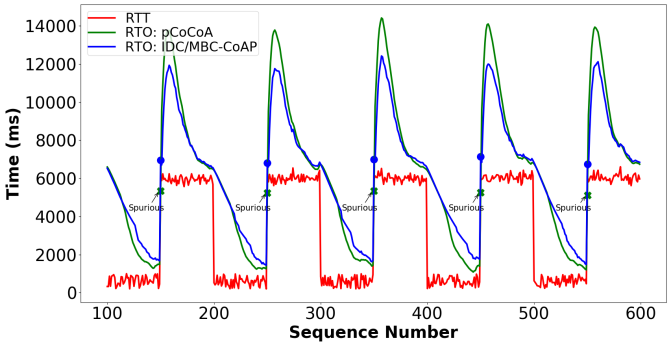


Figure 22: Spurious transmissions occurrence graph

6.2. Congestion Counteraction: Backoff-based vs. Rate-based

In this section, the performance of the full congestion control of IDC-CoAP and MBC-CoAP are analyzed and compared with several previous algorithms using primarily the following performance metrics:

- Goodput or success rate: Total amount of successfully received data in a given time interval
- Loss ratio: Observed losses at the application level
- Overhead: Total amount of lost packets in the network over total amount of packets sent successfully

In particular, the Overhead performance metric measures the ability of the congestion control to send packets only when there is no congestion in the network to avoid losses, and also avoid wasting energy. In other words, Overhead computes how much effort or energy is spent to send one packet successfully. This performance metric is very important especially for IoT devices supplied by batteries.

6.2.1. Simulation Results

Here, the performances of our IDC-CoAP and MBC-CoAP are analyzed and compared against pCoCoA, CoCoA+, 4-state, BDP-CoAP and BEB of the standard CoAP, using again our original python simulator presented in Section 4.1. Particularly, the objective is to explore the ability of the congestion control algorithms to adjust their sending rate to utilize the available network bandwidth.

Fig. 23 shows the behavior of the instantaneous sending rate achieved by the different congestion control algorithms in presence of a variable residual bandwidth. The red and green plots show respectively the sending and success rates of the algorithms, while the blue plot corresponds to the residual bandwidth available in the network during the simulation period. Simulation parameters for these figures and the next ones are all summarized in Table 2. Regarding the algorithms of previous work, we have used their default or advised parameters [1, 10, 11, 9, 19].

Parameter	Value	Description
r	4	Retransmission counter
α	1/8	First weight for RTO calculation
γ	1/32	Second weight for RTO calculation
K	4 or 7	Spurious weight for RTO calculation
Residual Bandwidth (packets/sec)	U(0.6, 1)	Instantaneous behaviour simulations
	U(0, 0.2) ... U(0.9, 1.1) ... U(2.1, 2.3)	Low variability simulations
	U(0.9, 1.1) ... U(0.5, 1.5) ... U(0, 2)	High variability simulations
RTT (ms)	N(500, 10)	Round Trip Time
MBC-CoAP specific parameters		
m	3 or 10	Number of measurements
pg	1.25, 0.75, 1, 1, 1, 1, 1, 1	Cycle pacing gains
IDC-CoAP specific parameters		
iw	1.1, 1.5	Incremental weight
dw	0.01	Decremental weight
fw	0.5	Fast decremental weight

Table 2: Simulation parameters used in evaluating congestion control algorithms for CoAP

According to Figures 23a, 23c and 23e, pCoCoA, 4-state and CoCoA+ sending rates are very high compared to the residual bandwidth because they do not adjust the sending rate according to the available bandwidth but try to send at the maximum allowable rate as soon as it seems to be possible. In fact, when a packet is lost, the initial value of the retransmission timeout RTO_{init} is multiplied by the backoff factor and hence the sending rate is decreased but without a direct relationship with the residual bandwidth. Still, after one or several losses and timeout multiplications, one retransmission goes through the network successfully. When the ACK of this packet is received, a new CoAP packet is immediately sent resulting in a sending rate that moves back again to the maximum allowable rate of $1/RTT$, which will cause again another loss. The transmission rate after this loss is $1/RTO_{init}$ which will very likely cause also another loss since RTO_{init} is optimized and its value is close to RTT. Even when the retransmission counter is reached and the CoAP packet is dropped definitively, the algorithms do not change their congestion counteraction and start sending the next packet using RTO_{init}.

In Figures 23a and 23c, the success rates of pCoCoA and 4-state are able to approach sometimes the residual

bandwidth when it increases but the sending rate continues to be much higher causing unnecessary retransmissions. In contrast, in Fig. 23e, the success rate of CoCoA+ is always far from the residual bandwidth despite the fact that its backoff mechanism is the same as pCoCoA. This is because CoCoA+ does not minimize the computation of RTO_{init} as pCoCoA, and hence the time required to retransmit lost packets is larger and the convergence to the residual bandwidth is slower. Once again, this shows the importance of minimizing RTO_{init}.

We notice also that in Fig. 23c, the success rate of 4-state is better than pCoCoA because 4-state uses more tuned backoff factors that allow to retransmit more quickly which can be seen from the oscillations of the sending rate plot. Unfortunately, this goodput gain comes with the cost of increasing retransmissions.

As a first conclusion, these algorithms can be efficient if the bandwidth is available most of the time and/or losses occurs sparsely due to other reasons such as interference. However, if losses occur because many connections are using the same bottleneck link in the network, i.e. congestion, then the three backoff-based algorithms fail to adjust the sending rate adequately, justifying the need for a "real" congestion control for CoAP.

In IDC-CoAP, the available bandwidth is respected in the calculation of the sending rate to minimize losses during congestion periods (Fig. 23b). When the packets are lost, IDC-CoAP tends to increase the spacing between successive transmissions which will reduce the sending rate to converge back to the available bandwidth and that is why the ratio of packet losses over successful packets is reduced as per Fig. 28. When the available bandwidth expands, IDC-CoAP ends up increasing its sending rate after a reasonable amount of time which can be reduced further by tuning the spacing decremental factor. As a result, the sending rate is the same as the success rate most of the time leading to a small loss ratio. Besides, the success rate tends to be very close to the available bandwidth when the latter is somewhat stable.

According to Fig. 23d, MBC-CoAP is also avoiding losses by trying to equate the sending rate with the success rate in order to stay below the residual bandwidth limit. We can see clearly the eight-phase cycle including the probing phase using the pacing gain of 1.25 that allows the sending rate to increase and thus converge slowly but surely to the available bandwidth offered to the CoAP sender. When the rate decreases suddenly, then MBC-CoAP takes some time to reduce its sending rate due to the cycle. However, this is compensated by a closer sending rate to the available bandwidth when the latter is somewhat stable.

The previous work rate-based BDP shown in Fig. 23f seems to perform similarly as backoff-based algorithms. This is because when the available bandwidth is lower than the initial estimated bandwidth which is the starting point of all simulated algorithms, successive losses prevent BDP from converging as we have explained in Section 4.3. Fig. 24 shows the same simulation for BDP when including retransmissions in bandwidth measurements. The modified BDP behaves now similarly to MBC-CoAP but in reality the convergence to the available bandwidth is still much slower resulting to less losses but to a much lower goodput. This is because the bandwidth estimation of BDP includes in addition to the maximum of previous measurements, the minimum of these measurements. Fig. 25 shows the sending rate of BDP when we replace the min-max filter by a max-filter. This second modification approaches now the behavior and the performance of MBC-CoAP.

From these instantaneous figures, one can conclude that the rate-based approach if well designed is more appropriate than the backoff-based approach. This will be confirmed further with next results where averages of Goodput, Loss ratio and Overhead are computed and compared in several network scenarios.

Fig. 26 - Fig. 28 represent the average of the 3 performance metrics: Goodput, Application Loss ratio and Overhead per simulation. The simulation was run up to 5 hours and each simulation was repeated 3 times. The residual bandwidth has been varied between 0.1 and 2 packets per second.

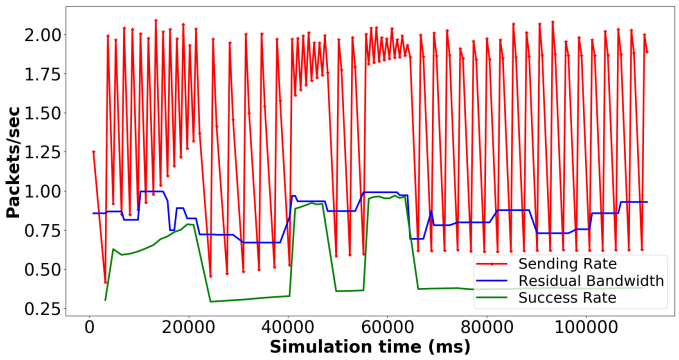
Fig. 26, shows the average goodput of all algorithms while varying the average residual bandwidth in each set of simulations. The variance around the average is fixed to the same value since the residual bandwidth is varied uniformly between the average - 0.1 and the average + 0.1. For small residual bandwidth values, IDC-COAP and MBC-CoAP are slightly better than backoff-based algorithms since there is no enough bandwidth to send packets. When the available bandwidth increases, our rate-based algorithms IDC-CoAP and MBC-CoAP show a linear behavior, however a step-wise behavior with the presence of a large plateau is shown by all backoff-based algorithms. This is caused by the non fine-grained control performed by the fixed backoff factors that impose few possible values of the retranssmion timeouts.

In the plateau, the goodput does not increase with the increase of the residual bandwidth. The value of the plateau corresponds approximately to

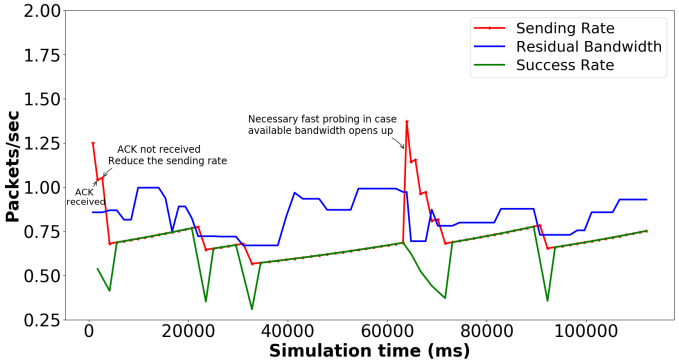
$$\frac{1}{RTO_{init}average + RTTaverage}$$

corresponding to a lost transmission followed by a successful retransmission. Thus for the default CoAP the plateau is at $\frac{1}{2\frac{1+1.5}{2}+0.5} = \frac{1}{2\times 1.25+0.5} = 0.33$. Recall that 1.5 is the randomization factor of the retransmission timeout. For other backoff-based algorithms that attempt to minimize RTO_{init}, the plateau is approximately at $\frac{1}{0.5\times 1.25+0.5} = 0.88$. Another smaller plateau appears for backoff-based algorithms around $\frac{1}{0.5+(0.5+2*0.5)\times 1.25} = 0.42$. In general, the plateau values correspond to

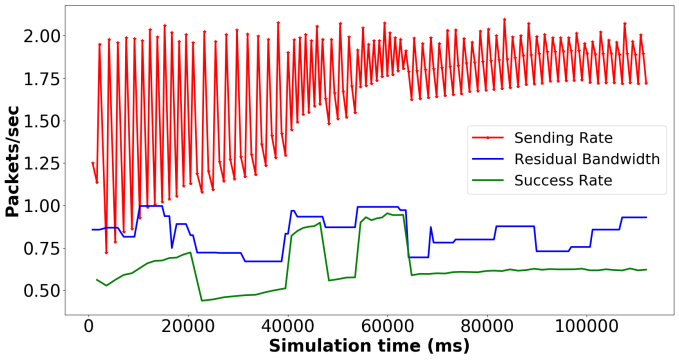
$$\frac{1}{RTTaverage + \sum_{i=0}^{j} b^i \times (RTO_{init}average)},$$



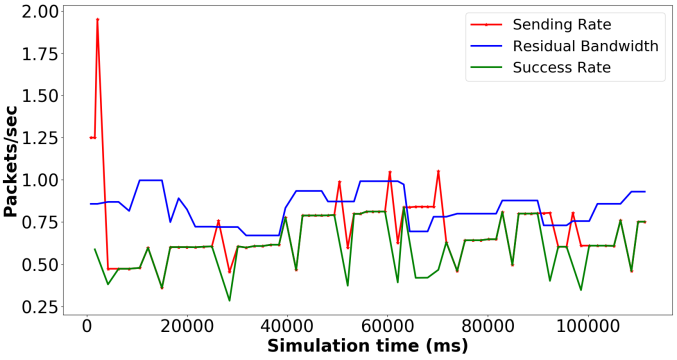
(a) pCoCoA



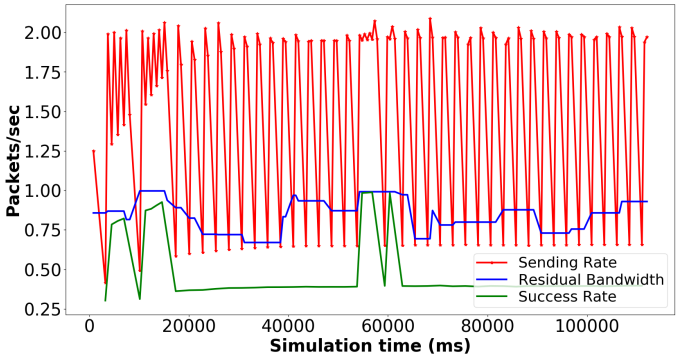
(b) IDC-CoAP



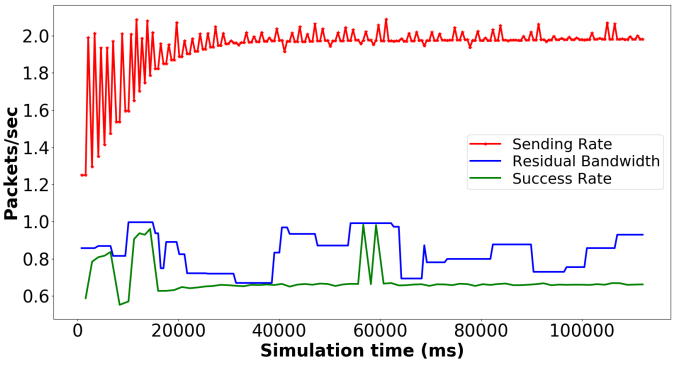
(c) 4-state



(d) MBC-CoAP



(e) CoCoA+



(f) BDP-CoAP

Figure 23: Instantaneous behaviour of backoff-based (left) and rate-based (right) congestion control algorithms

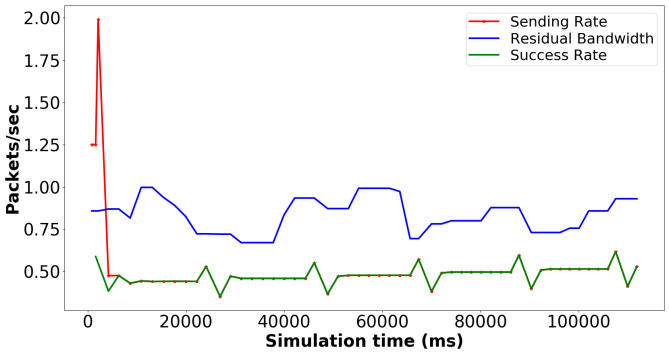


Figure 24: Correcting BDP-CoAP: Including bandwidth measurement samples from retransmissions

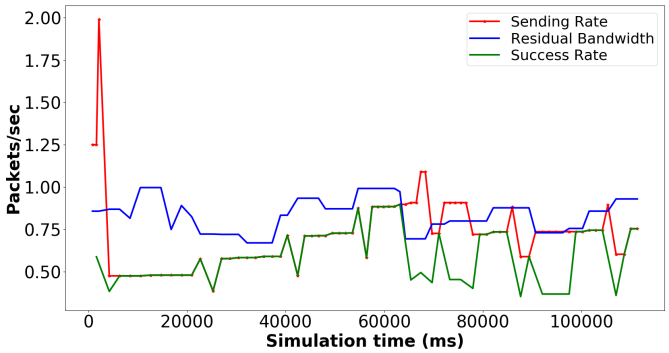


Figure 25: Correcting BDP-CoAP: Removing the samples minimum from the bandwidth estimation while including bandwidth measurement samples from retransmissions

$j = 0, 1, 2, \cdots$. The variable backoff factors used by the algorithms are not sufficient to perform a fine-grained control. Indeed, according to pCoCoA and CoCoA+, the chosen backoff factor for the given RTT is 2.5, thus the plateau is more precisely at value $\frac{1}{0.5+(0.5+\mathbf{2.5}*0.5)\times 1.25} = 0.37$. 4-state uses 1.7 half of the time and 2.5 most of the other half during the simulation which allow avoiding a clear first plateau but not the second large one.

On the contrary, the linear behavior of rate-based IDC and MBC algorithms engenders an additional gain of the goodput. Ideally, the goodput will be equal to the average available bandwidth which is not achievable because the rate control algorithm is operating blindly without prior knowledge of the network status. If the residual bandwidth offered to the CoAP sender is not very variable then the expression of the linear relationship between the goodput and the average residual bandwidth can be obtained through a steady state analysis. For MBC-CoAP, the goodput is computed by assuming that the gain cycling is operating close to the residual bandwidth and that bandwidth probing with the pacing gain 1.25 will bypass the available bandwidth. The goodput can be approximated by

$$\frac{7}{6 \times 1.25 + 1 + \frac{1.25}{0.75}} ResBW$$

For IDC-CoAP, we can compute the spacing values and the number of transmissions in a period comprised between two successive losses, which means when the spacing is equal to $1/ResBW$. Denote by S_i the current value of the spacing, then we will have successively $S_0 = iw/ResBW, S_1 = (1-dw)iw/ResBW, \cdots, S_n = (1-dw)^n iw/ResBW = 1/ResBW$. Hence, the total number of transmitted CoAP packets during this period is equal to

$$n = -\frac{\log(iw)}{\log(1-dw)}$$

The goodput is then approximated by

$$\frac{n}{\sum_{i=0}^{n}(1-dw)^i \frac{iw}{ResBW}} = \frac{dw}{iw}\frac{n}{1-(1-dw)^{n+1}} ResBW$$

This formula can be used to tune the incremental and decremental weights of IDC-CoAP for a given performance objective. Indeed, an incremental weight $iw = 1.1$ provides a better goodput than $iw = 1.5$ and than MBC-CoAP when the bandwidth variability is limited.

The last observation is for BDP-CoAP which behaves almost like backoff-based algorithms when the available bandwidth is small. Then, when the bandwidth increases approaching the maximum which is 1 packet/RTT it provides similar behavior as rate-based algorithms IDC-COAP and MBC-CoAP in terms of goodput.

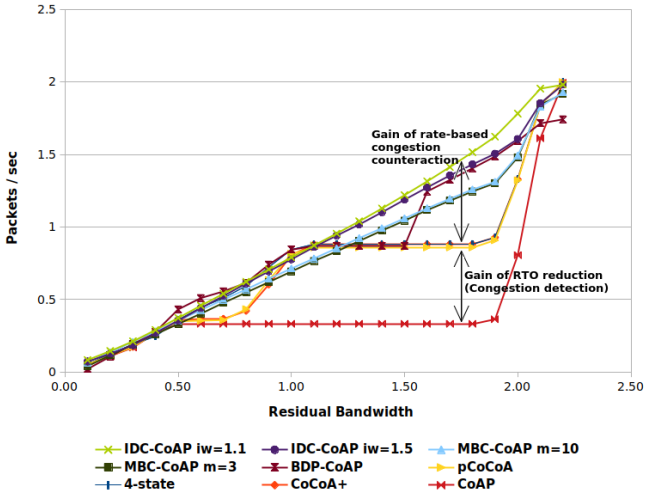


Figure 26: Goodput - Low variability residual bandwidth

Observed Loss ratio at the application level is presented in Fig. 27. When the residual bandwidth is increased, successive losses are reduced and hence application losses are reduced and even totally canceled because the retransmission counter r is 4. However, when the residual bandwidth is small, we see clearly that CoAP losses are

higher with all backoff-based algorithms including the default CoAP which means that these algorithms experience more successive network losses than rate-based algorithms. These results are confirmed in Fig. 28 where the Overhead metric is shown. Usually the algorithm that achieves a much higher goodput, experiences also a much higher Overhead. Nevertheless, all backoff-based algorithms show higher Overhead despite of having lower goodput as seen before. Thus they consume more energy and reduce battery life. It is difficult though to achieve a good tradeoff between Overhead and goodput. Tuning the parameters of IDC/MBC-CoAP helps improving this tradeoff. For IDC-CoAP, increasing the incremental weight iw and/or decreasing the decremental weight dw, decreases the goodput and reduces the Overhead. As a matter of fact, IDC-CoAP with $iw = 1.5$ achieves almost no overhead when the average residual bandwidth is greater than 0.3 packets/s. As for MBC-CoAP, decreasing the measurement window m, decreases the goodput and reduces the Overhead as we will see in next simulations.

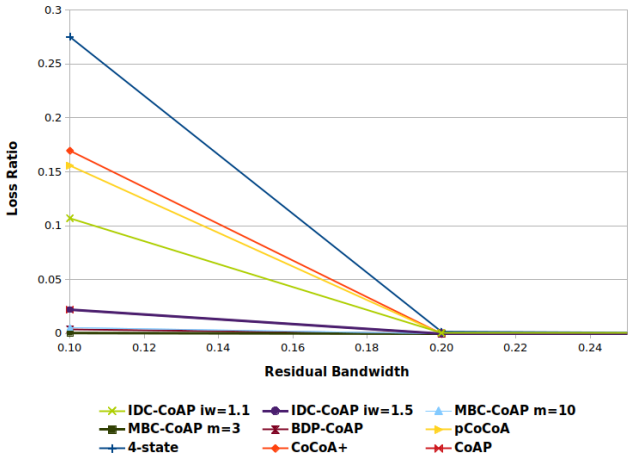


Figure 27: Application Loss ratio - Low variability residual bandwidth

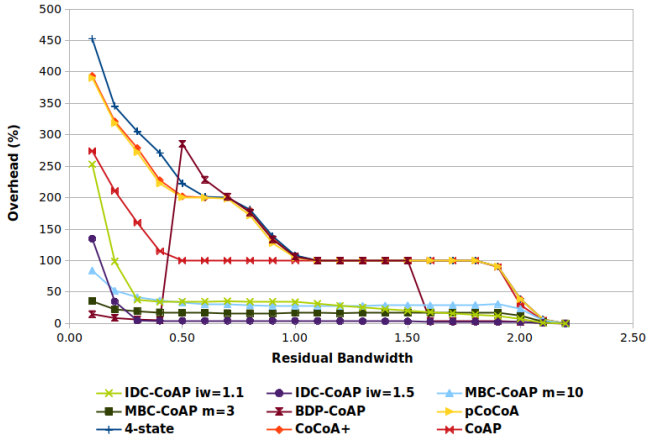


Figure 28: Overhead - Low variability residual bandwidth

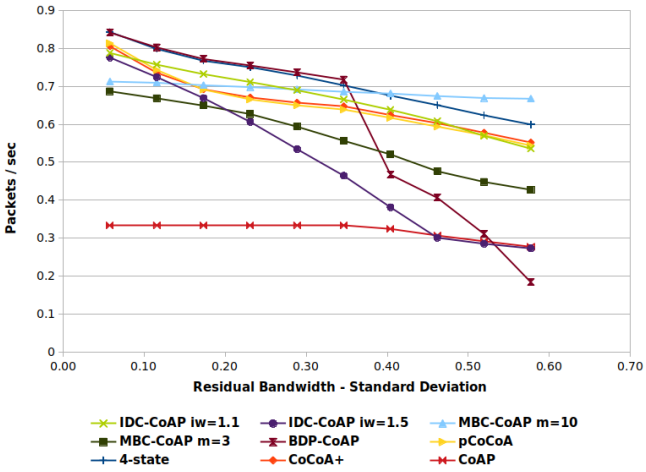


Figure 29: Goodput - High variability residual bandwidth

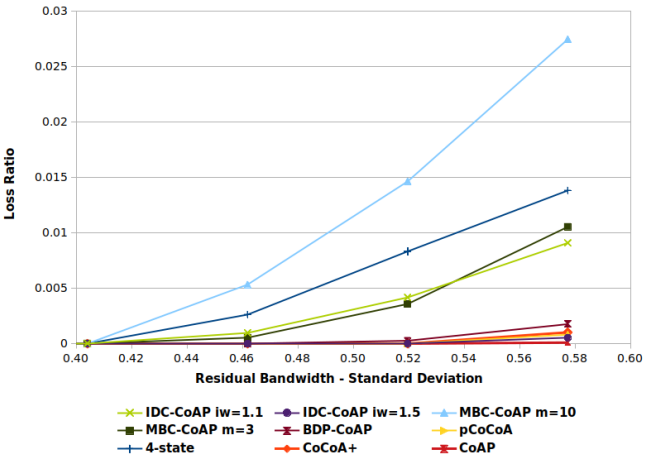


Figure 30: Application Loss ratio - High variability residual bandwidth

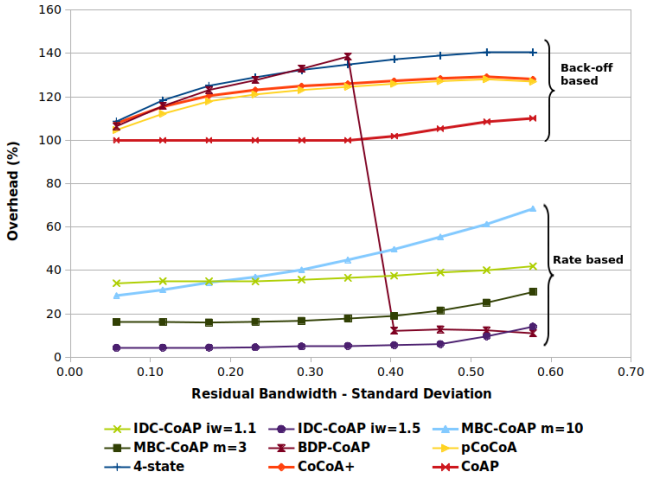


Figure 31: Overhead - High variability residual bandwidth

Finally, we test the robustness of the algorithms in front of a more dynamic network environment. In Fig. 29, 30 and 31, we fix the average residual bandwidth to 1 pack-

ets/s and we increase the standard deviation of the uniform distribution of the residual bandwidth by increasing the maximum and the minimum values from $[0.9, 1.1]$ to $[0, 2]$. Recall that 2 packets/s is the maximum possible rate which corresponds to $1/RTTaverage$. The residual bandwidth changes every 5 seconds. Here again, all backoff-based congestion control algorithms fail to reach an acceptable tradeoff between goodput and losses. Fig. 31 shows that the overhead of these algorithms are extremely high more than 100% and up to 140% indicating that any packet must be transmitted at least twice in order to be received successfully. The Overhead of IDC/MBC-CoAP is much lower while they still achieve a reasonable goodput in Fig. 29 even when the residual bandwidth is very variable. The overhead of IDC with $iw = 1.5$ is even around 10%. IDC with $iw = 1.5$ can be considered as a good tradeoff between goodput and Overhead especially when the residual bandwidth variability is medium. If the performance objective is a high goodput in a highly dynamic environment regardless of the Overhead and processing complexity, then MBC-CoAP with $m = 10$ is the choice because its goodput shows more stability thanks to the bandwidth estimation procedure and the gain cycling. In the contrary, the goodput of BDP-CoAP decreases dramatically with the increase of bandwidth variability. This is because the bandwidth is varying quickly between low and high values, thus the *min* operator used in the estimation of the residual bandwidth is not adequate at all. Besides, BDP uses the last ten attempts as measurement points instead of the last ten measurements, i.e. the measurement window slides at every transmission attempt. Unfortunately, this generates a similar behavior to a time window which is not adequate for CoAP because the number of sent packets per RTT is small especially when there are losses.

Finally, it is worth noticing that we have also performed same set of simulations with lower RTT values down to 50ms and higher values of the residual bandwidth up to 20 packets/s and we observed similar results including the step-wise behavior of backoff-based algorithms and the satisfactory performance of IDC-CoAP and MBC-CoAP in both low and high network bandwidth variability.

6.2.2. Implementation in Contiki OS and Cooja Simulations

In order to validate our study in a realistic environment, we have implemented IDC-CoAP in the Contiki Operating System [25]. Contiki OS is used for IoT devices and especially tiny ones such as the TelosB/SkyMote family and Zolertia Z1 mote [26]. Then, we use the real hardware emulator MSPSim [27, 28] to load Contiki OS on it, and we use Cooja simulator [24] to create several network scenarios composed of motes playing the role of a CoAP receiver and CoAP senders.

One of MSPSIM and Cooja features is the ability of emulating constrained real devices while reflecting their hardware specifications and processing capacities. The motes implement IEEE 802.15.4 at the physical and MAC layers. Distinguished from previous works, the radio duty cycle (RDC) feature of the MAC layer is kept enabled in all our experiments. Two types of motes are used for CoAP senders and receivers which are Z1 and wismote. Table 3 shows the hardware specifications of these motes and the settings in the Contiki OS loaded in the motes. The RPL router uses the more constrained sky mote since it implements neither transport nor application layers.

Zolertia Z1 mote specifications	
RAM	8KB
ROM	92KB
Micro-Controller	MSP430F2617
CPU Clock speed	16MHz
RF standard	CC2420 2.4GHz / 250Kbps data rate
Wismote mote specifications	
RAM	16KB
ROM	256KB
Micro-Controller	MSP430F5
CPU Clock speed	25MHz
RF standard	CC2520 2.4GHz / 250Kbps data rate
Simulation Parameters	
Physical protocol	IEEE 802.15.4
RDC	On (Contikimac driver)
MAC	CSMA driver
Transmission (TX) ratio	90% or 95% or 100%
Routing protocol	RPL
Network protocol	6LoWPAN/IPv6
UIP buffer size	256
CoAP frame size	80 bytes

Table 3: Cooja/Conitki parameters and hardware specifications of Z1 and Wismote motes

Six network topologies with different number of nodes are defined for the performance analysis. These topologies are: a grid of 30 nodes, a U-shape with 15 nodes, a Square-shape with 18 nodes, a ring and a chain with 12 nodes, and a dumbbell with 7 or 10 or 15 nodes. The Square-shape and the U-shape are obtained by shutting down some nodes in the grid topology. In the first one only border nodes communicate with the CoAP receiver and in the second one only the border nodes forming a U-shape communicate with the receiver. Fig. 32 illustrates these 6 topologies with 1 RPL border router (green color), CoAP receiver (yellow color) and CoAP senders (pink color). The distance between the unit squares is 10m. Choosing various network topologies determines how many direct neighbors each node has. Also, it determines how many nodes compete for the radio channel and the available bandwidth. These topologies create a diversity of available links, bandwidth and number of CoAP connections in the network.

Destination Oriented Directed Acyclic Graph (DODAG) is initiated by the RPL border router which stores the routing information for all the nodes. The RPL border router serves as a relay for CoAP messages, it does not send or receive any CoAP message. An initialization phase around 100 seconds for each simulation is allowed since the RPL border router needs an amount of time to build

the DODAG across the network. No results are collected during this phase. Once the network is initialized, CoAP senders generate messages which are directed towards the CoAP receiver. NSTART is set to 1 as per CoAP default specification. The simulations of the different scenarios have a 15 min duration. These simulations are repeated 5 times for each scenario. On one hand, as shown in the previous sections, IDC-CoAP with $iw = 1.5$ is a good tradeoff between goodput, Overhead and code simplicity then we can choose it as a representative of rate-based congestion control algorithms in this study with Contiki/Cooja. On the other hand, we have obtained the last version of the implementation of CoCoA+ from its authors, which will serve as a representative of backoff-based approach for congestion control. In the previous section we showed that the performance of CoCoA+ is indeed very close to pCoCoA and 4-state. Besides, we will compare with the existing CoAP implementation in Contiki that follows the current standard using the simple binary exponential backoff [1].

Fig. 33 shows the three measured performance metrics: Goodput, Loss ratio and Overhead for the three protocols: IDC-CoAP, CoCoA+ and CoAP for each of the four network topologies: Ring, Chain, Dumbbell and Grid. As a first observation, it is clear that IDC-CoAP is always better than CoCoA+, and that CoAP is evidently not efficient enough in all cases.

More closely, Fig. 33a shows that the Goodput of IDC-CoAP is higher than CoCoA+ even when the transmission ratio TX is 100%, i.e. no loss simulation by Cooja. In the ring topology, most of the motes send directly to the CoAP receiver and thus the network is somewhat stable with less congestion events. To challenge more the algorithms, we decrease the transmission ratio to 95% and 90%. Still CoCoA+ and CoAP achieve lower Goodput than IDC-CoAP. They also experience more application losses (Fig. 33b) and overhead (Fig. 33c). Even when transmission losses are high (TX=90%), IDC-CoAP is able to reduce the Overhead compared to other algorithms as shown in Fig. 33c. This is because transmission losses are perceived as a residual bandwidth reduction from the sender and thus IDC-CoAP which applies the rate-based congestion control reacts better to these losses.

In the chain topology, we varied the number of nodes in the chain to study the impact on the Goodput, Loss ratio and Overhead. Naturally, the overall performance degrades when the number of nodes in the chain increases because the nodes near to the receiver becomes more congested. In Fig. 33d, IDC-CoAP algorithm results in a better goodput than CoCoA+ and CoAP in all number of nodes. It also achieves zero loss ratio and lower overhead when the number of nodes varies as per Fig. 33e and Fig. 33f. Hence, IDC-CoAP preformance is still robust when we have more congested nodes in the network.

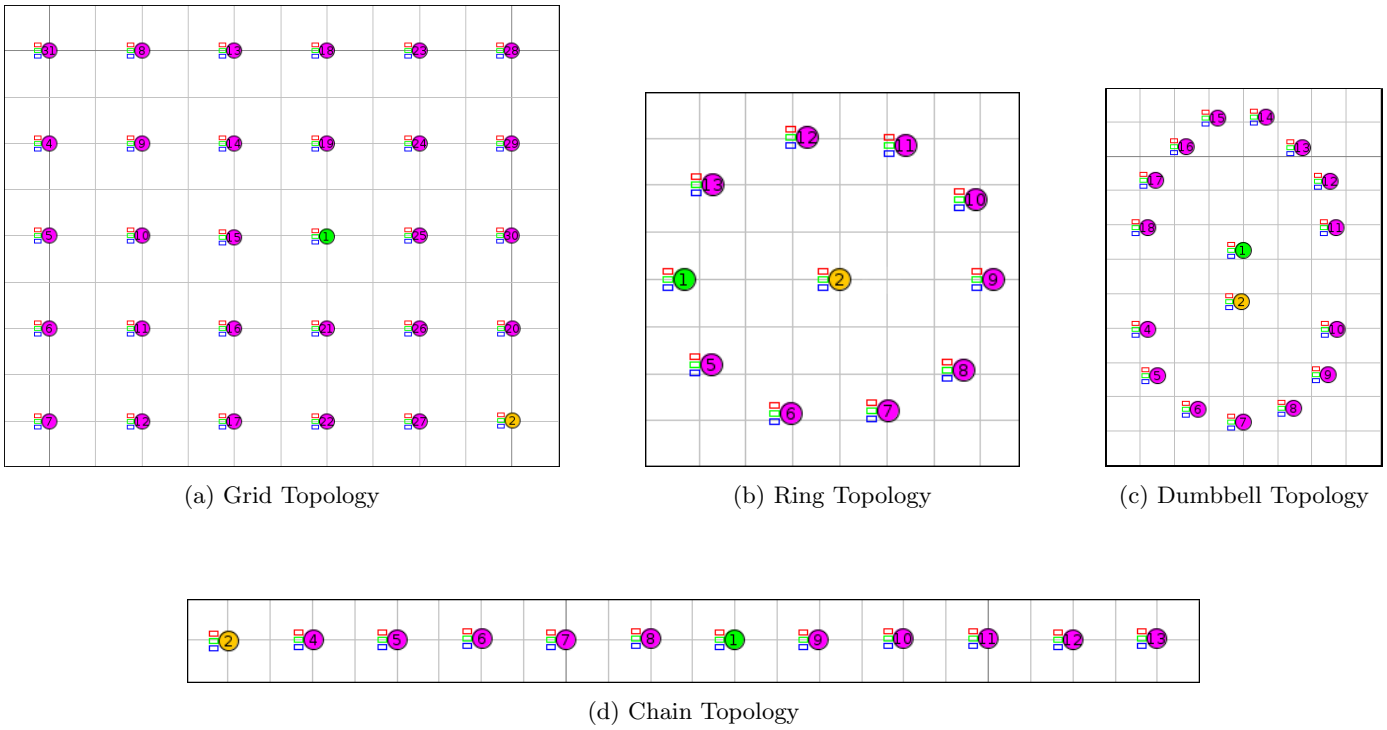
Similar to chain and ring topologies, IDC-CoAP attains better performance results in dumbbell topology and this is illustrated in Fig. 33g to Fig. 33i. The creation of a congested link between the RPL router and the CoAP receiver does not impact the relative performance of IDC-CoAP compared to the others. Here also we varied the number of nodes with similar results as in the chain topology.

The results of the grid topology and its sub-topologies Square-shape and U-shape are shown in Fig. 33j to Fig. 33l. Here, the position of congestion in the network topology becomes more variable and the residual bandwidth as perceived by senders can be more dynamic, especially in the full grid topology. Even in this case, IDC-CoAP is still showing a higher goodput, a lower application loss ratio and also a lower overhead. This reinforce the necessity of rate-based congestion control for CoAP rather than current backoff-based ones.

7. Conclusion and Future Work

Since the design of the Constrained Application Protocol (CoAP), several research works have tried to improve further its performance so that it becomes more promising and improve significantly its widespread usage in different IoT fields. Particularly, congestion control algorithms in this protocol play an important role in its efficiency in terms of reliability, energy consumption and rate performance. Most previous works have followed the backoff-based approach for congestion control while in this paper we proved that a rate-based approach is much better in most network scenarios. Indeed, the non fine-grained nature of the backoff procedure does not allow a precise control even when several backoff factors and weights are used. To be concretely efficient, the rate-based control must be at the same time simple and well adapted to IoT networks and devices that employ the CoAP protocol. Our two rate-based protocols IDC-CoAP and MBC-CoAP are able to leverage the available bandwidth in the network and thus reduce message losses and unnecessary retransmissions which are very harmful to IoT devices. Our results, obtained from pure simulation and from the Cooja/Contiki environment, show that the more the available bandwidth or the network dynamics, the more the gain in all performance metrics. Besides, we provided a simpler and better algorithm for retransmission timeout reduction which improves congestion detection, an essential component for congestion control.

Depending on the application, our congestion control protocols can be tuned to optimize further the performance. For instance, in IoT health monitoring applications, some information should be delivered with high reliability such as the body temperature or the heart rate. In this case, the CoAP sending rate can be reduced conservatively to avoid losses. In a weather sensing application for example, a packet lost is less damaging because the next packet has an updated data of the weather. Besides, if sensors are using wired power supplies, increasing the sending rate allows more data to be collected. Finally, since CoAP can be used also for file transfer, incorporating an effective rate-based control in it is beneficial.



(a) Grid Topology (b) Ring Topology (c) Dumbbell Topology

(d) Chain Topology

Figure 32: Network topologies for CoAP congestion control performance evaluation with Cooja/Contiki OS environment

In [29], we have developed a model that brings up the importance of reducing the retransmission timeout and the inadequacy of the backoff procedure in CoAP. It would be interesting to develop similar models for IDC-CoAP and MBC-CoAP to complete analytically their study. Moreover, since our results show that the measurement-based congestion control MBC-CoAP has robust stability and convergence properties, we aim to explore how to reduce further its complexity so that it can be incorporated in very tiny IoT devices. Naturally, we would also compare our new CoAP protocols with other protocols that use TCP as a transport protocol such as MQTT.

Acknowledgement

We would like to thank Prof. Neal Cardwell, BBR author, for the fruitful discussion and his prompt feedback during MBC-CoAP implementation in our simulator. We also thank Prof. August Betzler, CoCoA+ author, for providing the latest version of CoCoA+ and his precious guidance for setting CoCoA+ in Cooja/Contiki.

References

[1] Z. Shelby, K. Hartke, C. Bormann, The Constrained Application Protocol (CoAP), RFC 7252, https://rfc-editor.org/rfc/rfc7252.txt (Jun. 2014).

[2] J. Joshi, D. Kurian, S. Bhasin, S. Mukherjee, P. Awasthi, S. Sharma, S. Mittal, Health monitoring using wearable sensor and cloud computing, in: 2016 International Conference on Cybernetics, Robotics and Control (CRC), 2016, pp. 104–108.

[3] O. Bergmann, K. T. Hillmann, S. Gerdes, A CoAP-gateway for smart homes, 2012 International Conference on Computing, Networking and Communications (ICNC) (2012) 446–450.

[4] I. Shin, D. Eom, B. Song, The CoAP-based m2m gateway for distribution automation system using DNP3.0 in smart grid environment, in: 2015 IEEE International Conference on Smart Grid Communications (SmartGridComm), 2015, pp. 713–718.

[5] J. Krimmling, S. Peter, Integration and evaluation of intrusion detection for CoAP in smart city applications, 2014 IEEE Conference on Communications and Network Security (2014) 73–78.

[6] D. Thangavel, X. Ma, A. Valera, H. X. Tan, C. K. Y. Tan, Performance evaluation of MQTT and CoAP via a common middleware, in: 2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014, pp. 1–6.

[7] I. Järvinen, I. Raitahila, Z. Cao, M. Kojo, Is CoAP congestion safe?, in: Proceedings of the Applied Networking Research Workshop, ANRW '18, Association for Computing Machinery, New York, NY, USA, 2018, p. 4349.

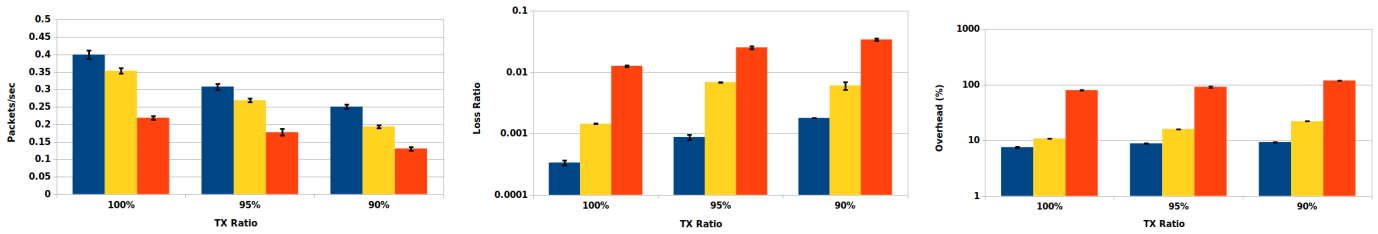
[8] A. Maheshwari, R. K. Yadav, Analysis of congestion control mechanism for IoT, in: 2020 10th International Conference on Cloud Computing, Data Science Engineering (Confluence), 2020, pp. 288–293.

[9] S. Bolettieri, G. Tanganelli, C. Vallati, E. Mingozzi, pCoCoA: A precise congestion control algorithm for coap, Ad Hoc Networks 80 (2018) 116 – 129.

[10] A. Betzler, C. Gomez, I. Demirkol, J. Paradells, CoCoA+: An advanced congestion control mechanism for CoAP, Ad Hoc Networks 33 (2015) 126 – 139.

[11] R. Bhalerao, S. S. Subramanian, J. Pasquale, An analysis and improvement of congestion control in the CoAP Internet-of-Things protocol, in: 2016 13th IEEE Annual Consumer Communications Networking Conference (CCNC), 2016, pp. 889–894.

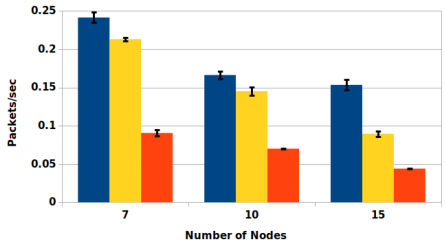
[12] M. Collina, M. Bartolucci, A. Vanelli-Coralli, G. E. Corazza, Internet of things application layer protocol analysis over error and delay prone links, in: 2014 7th Advanced Satellite Mul-



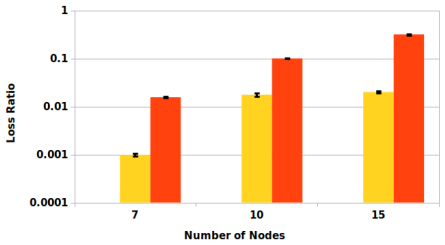
(a) Goodput - Ring Topology

(b) Application Loss Ratio - Ring

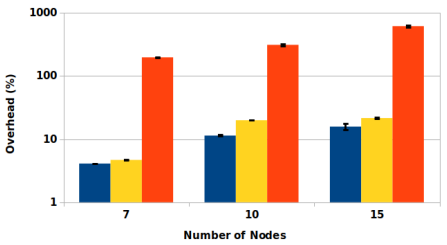
(c) Overhead - Ring



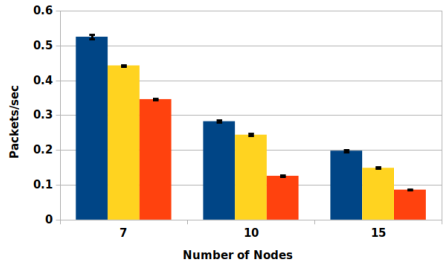
(d) Goodput - Chain



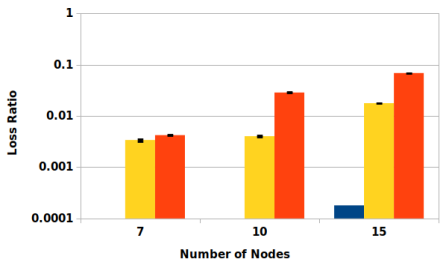
(e) Application Loss Ratio - Chain



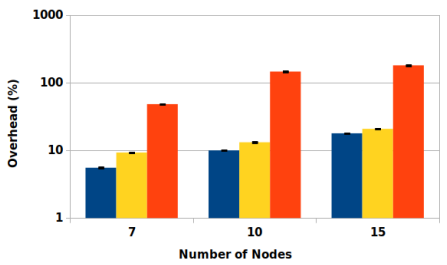
(f) Overhead - Chain



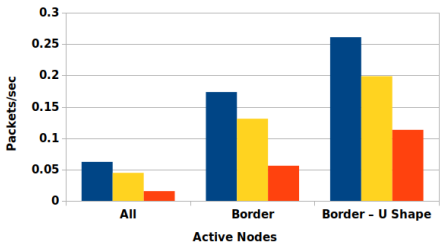
(g) Goodput - Dumbbell



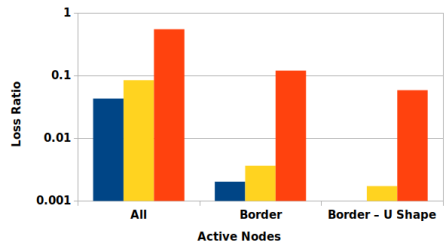
(h) Application Loss Ratio - Dumbbell



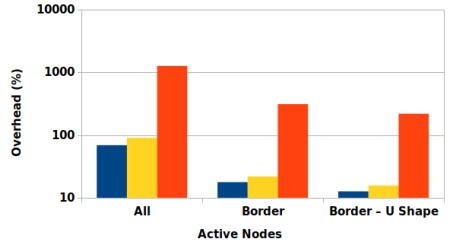
(i) Overhead - Dumbbell



(j) Goodput - Grid



(k) Application Loss Ratio - Grid



(l) Overhead - Grid

Figure 33: Performance evaluation results of IDC-CoAP (rate-based) vs. CoCoA+ and CoAP (backoff-based) using Cooja/Contiki

timedia Systems Conference and the 13th Signal Processing for Space Communications Workshop (ASMS/SPSC), 2014, pp. 398–404.
[13] J. J. Lee, S. M. Chung, B. Lee, K. T. Kim, H. Y. Youn, Round trip time based adaptive congestion control with CoAP for sensor network, in: 2016 International Conference on Distributed Computing in Sensor Systems (DCOSS), 2016, pp. 113–115.
[14] A. Betzler, C. Gomez, I. Demirkol, J. Paradells, CoAP congestion control for the internet of things, IEEE Communications Magazine 54 (7) (2016) 154–160.
[15] A. Betzler, C. Gomez, I. Demirkol, J. Paradells, CoAP congestion control for the internet of things, IEEE Communications Magazine 54 (7) (2016) 154–160.
[16] I. Jarvinen, I. Raitahila, Z. Cao, M. Kojo, FASOR Retransmission Timeout and Congestion Control Mechanism for CoAP, in: 2018 IEEE Global Communications Conference (GLOBECOM), 2018, pp. 1–7.
[17] S. Hemminger, NetEm - Network Emulator, http://manpages.ubuntu.com/manpages/cosmic/en/man8/tc-netem.8.html, last checked: 01.03.2020.
[18] O. Bergmann, libcoap: C-implementation of CoAP, https://libcoap.net/, last checked: 01.03.2020.
[19] E. Ancillotti, R. Bruno, BDP-CoAP: Leveraging Bandwidth-Delay Product for Congestion Control in CoAP, in: 2019 IEEE 5th World Forum on Internet of Things (WF-IoT), 2019, pp. 656–661.
[20] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, V. Jacobson, BBR: Congestion-Based Congestion Control, ACM Queue 14, September-October (2016) 20 – 53.
[21] V. Karagiannis, P. Chatzimisios, F. Vázquez-Gallego, J. Alonso-Zarate, A Survey on Application Layer Protocols for the Internet of Things, Transaction on IoT and Cloud Computing (TICC), 2015 1 (1).
[22] P. Sarolahti, A. Kuznetsov, Congestion Control in Linux TCP, in: Proceedings of the FREENIX Track: 2002 USENIX Annual Technical Conference, USENIX Association, Berkeley, CA, USA, 2002, pp. 49–62.
[23] V. Jacobson, Congestion Avoidance and Control, SIGCOMM Comput. Commun. Rev. 18 (4) (1988) 314–329.
[24] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, T. Voigt, Cross-Level Sensor Network Simulation with COOJA, in: Proceedings. 2006 31st IEEE Conference on Local Computer Networks, 2006, pp. 641–648.
[25] A. Dunkels, B. Gronvall, T. Voigt, Contiki - a lightweight and flexible operating system for tiny networked sensors, in: 29th Annual IEEE International Conference on Local Computer Networks, 2004, pp. 455–462.
[26] Zolertia Z1, Low-power Wireless Sensor Network Platform, https://github.com/Zolertia/Resources/wiki/The-Z1-mote, last checked: 01.10.2020.
[27] MSPSim, Emulator of the MSP430 series, https://github.com/contiki-ng/mspsim, last checked: 01.10.2020.
[28] M. Alonso-Arce, J. Aorga, S. Arrizabalaga, P. Bustamante, A wireless sensor network PBL lab for the master in telecommunications engineering, in: 2016 Technologies Applied to Electronics Teaching (TAEE), 2016, pp. 1–8.
[29] N. Makarem, W. Bou Diab, I. Mougharbel, N. Malouch, Performance study of the constrained application protocol in lossy networks, in: 2019 IFIP Networking Conference (IFIP Networking), 2019, pp. 1–2.