# An Improved Adaptive CoAP Congestion Control Algorithm

https://doi.org/10.3991/ijoe.v15i03.9122

Fathia Ouakasse(✉), Said Rakrak
Cadi Ayyad University, Marrakesh, Morocco
fathia.ouakasse@gmail.com

**Abstract**—The Constrained Application Protocol (CoAP) is one of the most emerging messaging protocols that have successfully fulfilled the need of the lightweight features required to handle communication between constrained devices in IoT environment. However, these devices are generating a huge amount of messages and notifications which cause the network congestion. Then, the challenge addressed in this paper; consists of designing a suitable congestion control mechanism for CoAP that ensures a safe network operation while keeping the use of network resources efficiently. To do so, this paper presents an improved congestion control algorithm for the estimation of a Retransmission Time Out (RTO) value to use in each transaction based on the packet loss ratio and the Round-Trip Time RTT of the previous transmission. A comprehensive analysis and evaluation of simulated results show that the proposed mechanism can appropriately achieve higher performance compared to the basic CoAP congestion control, TCP and TCP/ Linux.

**Keywords**—IoT, CoAP, congestion control, RTO, packet loss, RTT.

## 1 Introduction

Nowadays, Wireless Sensor Networks (WSN) have been widely deployed in several IoT application fields in order to measure, control or detect physical and environmental events such as pressure, humidity, temperature and pollution levels. Furthermore, in recent critical applications, that may require an urgent intervention, like healthcare, smart grid, and ambient assisted living, the challenge consists of getting information when an event of interest occurs in order to intervene in real-time.

In order to cover these requirements, the publish/subscribe model [1] is designed as the most appropriate model. Furthermore, several protocols based on this model were designed to support IoT applications. One of the most important protocols is CoAP [2]. This is going back to the fact that CoAP is the most appropriate protocol for lightweight devices and constrained resources in terms of memory, energy, and computing. Thus, CoAP has been widely used in different application fields for resource-constrained networks and M2M applications ranging from smart grid [3], building and home automation [4] smart cities [5] to healthcare industry [6], where real-time updates of the patient's status were provided via CoAP.

Although, CoAP has increasingly been used and proved its effectiveness in gathering data from smart sensors and controlling constrained devices, the problem of network congestion [7] still represents the great limitation that hinders the proper functioning of this protocol and causes the loss of packets. Network congestion can also significantly cause a performance deterioration of the network, manifesting in increased packet latencies, while a network may even become useless if the congestion collapse occurs [8].

Indeed, the core CoAP specification defines a basic congestion control mechanism which consists of the use of a back off mechanism to compute the Retransmission Time Out (RTO) for the next transmission. It is based on the use of a fixed RTO value which is doubled in each retransmission. Nevertheless, even if the core CoAP specification defines a basic congestion control mechanism to make it able to handle congestion control by itself, researches have proved that CoAP is not capable of being adaptive to network conditions. Actually, and as demonstrated in [9], this incapability is due to the fact that CoAP doesn't take into consideration the RTT of a packet since the network conditions may change frequently because of the dynamic topology and the density of WSN nodes.

To address all those aforementioned problems, we propose, in this paper, an improved adaptive congestion control algorithm that overcomes the limitation persisting under the use of traditional CoAP congestion algorithm. The principle of our improvement consists of the use of both of the loss packets' ratio and the RTT value in order to estimate an appropriate RTO for the next retransmission. Simulation results show that our proposition outperforms compared to the basic CoAP congestion control, TCP and Linux RTO.

The remainder of this paper is organized as follows: a brief description of the CoAP application layer protocol including reliability and the basic CoAP congestion control algorithm is presented as background in the second section. Then, in the third section, the classical TCP congestion control algorithm and TCP Linux are described. Some related works are discussed in the fourth section. Afterward, in the fifth section, the proposed improved congestion control algorithm is detailed and in the sixth section a comparative simulation of the proposed algorithm, Default CoAP congestion control, Basic RTO and Linux RTO is drawn using NS2 network simulator. Finally, a conclusion and some future directions are closing up our paper in the seventh section.

## 2 Background

In this section a description of CoAP protocol and the default congestion control are presented.

### 2.1 CoAP overview

CoAP is an application layer protocol based on Representational State Transfer (REST) architecture; it has been designed by the Internet Engineering Task Force (IETF) to support IoT with lightweight messaging for devices operating in a

constrained environment. CoAP defines two kinds of interactions between end-points; the client/server and the publish/subscribe interactions. The client/server model supports two interaction types: (i) a one-to-one interaction i.e. request/reply and (ii) a multi-cast interaction i.e. a client interrogating several servers using requests. And the publish/subscribe model called the observer model [10]; here the server plays the role of a publisher and the observer plays the role of a subscriber. A server can send messages of notifications called publications about an event interesting the observer. Otherwise, communication between clients and servers is afforded through connectionless datagrams because CoAP runs over UDP. Retries and reordering are implemented in the application stack. UDP broadcasts and multicasts are also allowed by CoAP for addressing [11]. In addition, as in the case of TCP, to make sure that messages arrived without a significant communication overhead, a basic error checking and verification for UDP can be built [12], which makes CoAP more suitable for the IoT domain. An over-view architecture of the CoAP protocol is drawn in Figure 1.

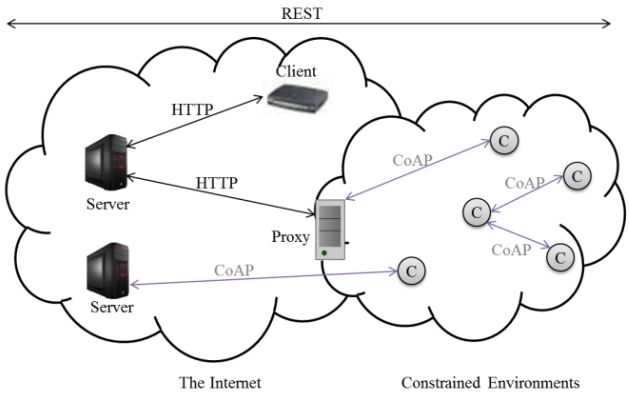


**Fig. 1.** An overview architecture of CoAP protocol

Furthermore, CoAP utilizes four message types; confirmable, non-confirmable, reset, and acknowledgment, where two among them concern reliability messages. The reliability of CoAP consists of a confirmable message and a non-confirmable message [9]. In the case of a confirmable message, an acknowledgment message (ACK) is sent to the sender from the intended, else the message is retransmitted. This is just a confirmation that the message is received, but it doesn't confirm that its contents were decoded correctly. However, a non-confirmable message is fire and forget, i.e. no reception confirmation [13].

## 2.2 CoAP default congestion control

The problem of congestion happens when the traffic load offered to a network approaches the network capacity [14]. This phenomenon is one of the main obstacles that still hinder the well-functioning of many protocols and thus impacts directly the

efficiency of the communication. In this context, CoAP must handle the congestion control by itself because it is based on UDP. Unlike HTTP which is based on TCP where a proper end-to-end congestion control is provided. CoAP offers a basic congestion control in the case of Confirmable messages, which consists of the use of a fixed RTO [15]. Initially, the RTO value is set to a random number between a constant ACK TIMEOUT and a constant ACK TIMEOUT multiplied by a constant ACK_RANDOM_FACTOR [16]. By using an exponential back-off mechanism, messages that haven't received an acknowledgment within the fixed RTO duration are retransmitted and subsequently, this RTO value is doubled. As a result, CoAP defines a constant MAX_RETRANSMIT, which specifies the maximum number of message retransmissions before the sender stops sending and the transmission is considered to have failed. In Table 1, we present the default values of parameters used in the basic CoAP congestion control mechanism as specified by the base CoAP specification [14]. Moreover, in Figure 2, the CoAP default congestion control is drawn.

**Table 1.** Default parameter values as specified in CoAP specification

| Parameter | Value |
|---|---|
| ACK TIMEOUT | 2 s |
| ACK RANDOM FACTOR | 1.5 |
| MAX RETRANSMIT | 4 |

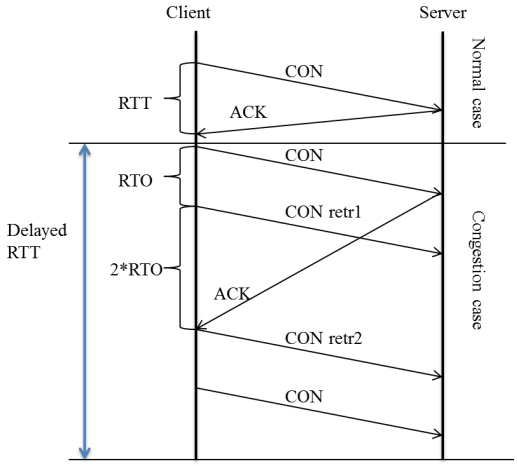


**Fig. 2.** The CoAP default congestion control

Although the core CoAP specification defines a basic congestion control mechanism to make it able to handle congestion control by itself, default CoAP congestion control is significantly considered too aggressive because it is still not capable of adapting its behavior to network conditions [17]. So, the fact that it doesn't take into account the value of previous RTT makes it difficult to determine an optimal RTO. Furthermore, if the value of RTO used by the basic CoAP congestion control is less than the actual RTT, it can lead to inaccurate retransmissions;

inaccurate retransmissions occur in the case of delays in the confirmation of the request. On the other hand, if the RTO value exceeds the RTT, it can lead to unnecessarily long idle times before retransmitting the packet in case the request or confirmation packets were lost [18].

So, the calculation of an appropriate RTO is essential to overcome the problem of congestion.

# 3 Congestion Control Algorithms

In this section we describe two of the main congestion control mechanisms TCP based; the Classical TCP congestion control algorithm and TCP Linux.

## 3.1 Classical TCP congestion control algorithm

Unlike the basic CoAP congestion control which uses a fixed RTO, the computation of RTO in the classical TCP congestion control is based on the history variation of RTT. The specification of this algorithm is proposed by RFC 6298 [19]. According to this specification, the calculation of the actual RTO to use in the next transmission is based on two variables; smoothed average of RTT (SRTT) and RTT variation (RTTVAR); where SRTT is used to preserve the history of RTT and RTTVAR keeps the history of RTT variation. Both of these parameters are constant and their impact factors respectively are 7/8 and 3/4.

Initially, the RTO value is set to 1 second. After the first transmission, the first RTT value is received. To compute the following RTO value, SRTT is set as RTT received and RTTVAR is set as $RTT_{received/2}$, the following formulas are used [16]:

$$SRTT = RTT$$

$$RTTVAR = RTT/2$$

After subsequent RTT measurements are received, the following formulas are applied:

$$SRTT = (1 - \alpha) * SRTT + \alpha * RTT \quad (1)$$

$$RTTVAR = (1 - \beta) * RTTVAR + \beta * |SRTT - RTT| \quad (2)$$

$$RTO = SRTT + \max (G, K*RTTVAR) \quad (3)$$

The formula (3) is used to estimate the RTO value to be used in the following transmission. When the RTO timer expires, the RTO value is doubled [20].

According to RFC 2988 [8], the value of the constant K in (3) is 4. Furthermore in (1) and (2) alpha and beta are also constants and their values respectively are 1/8, 1/4.

The mechanism of the calculation of RTTVAR and SRTT based on the RTT is described in Figure 3.

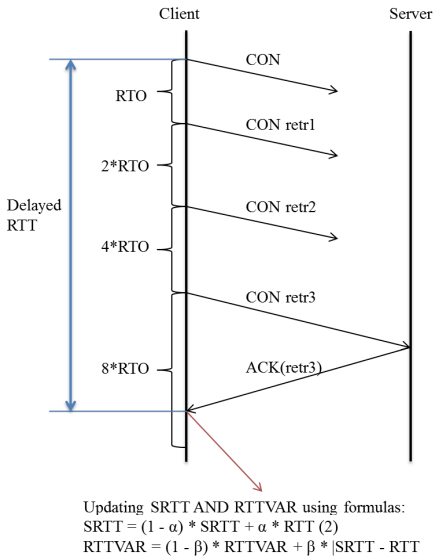


**Fig. 3.** The calculation of RTT in the classical TCP congestion control

Moreover, the G value defines the clock granularity in seconds and according to experiences, finer clock granularities inferior or equal to 100 ms perform somewhat better than other granularity values [8]. Thus, it is recommended to choose the G value not greater than 100 ms [19]. At the same time, G should be at least one order of magnitude smaller than the RTT [21].

### 3.2 Linux RTO

Linux RTO is based on the classical TCP congestion control algorithm recommended in RFC 6298. However, Linux RTO adds some modifications. Indeed, in Linux RTO the estimation of the following RTO value is done via two mechanisms: (i) if the current measured RTT value is smaller than the previous RTT value, and in order to avoid the peaks in RTO value when the channel seems to improve and (ii) if constant RTT values are given after subsequent measurement, and in order to avoid spurious retransmissions, Linux-RTO algorithm avoids the RTO estimator to converge into an RTT value [22]. Since classical RTO and Linux RTO use the RTTs measured values to update the RTO value; we refer to them in general as RTT-based algorithms.

## 4 Related Works

Since the basic CoAP congestion control mechanism can hardly meet the requirements of many IoT applications, several approaches were proposed to improve

the aforementioned CoAP shortcomings. The CoAP Simple Congestion Control/Advanced CoCoA [23] is the most important extension of CoAP that has been standardized by IETF. Indeed, basic CoAP doesn't care about the network characteristics; it behaves the same way in any type of network. So, in order to optimize the CoAP congestion control abilities, CoCoA, based on TCP RTO estimation algorithm, uses RTT measurements to add state information about individual RTOs for different destination endpoints based on two mechanisms: (i) a strong estimator; when the packet is received on the initial transmission without any retransmissions and (ii) a weak estimator; when RTT value is measured after at most two retransmissions. Both of those mechanisms implement the same algorithm but have different sets of state variables. The overall RTO estimated in the formula (6) is made from the estimator that made the most recent contribution using either formula (4) or formula (5) [21].

$$RTOrecent = 0.25 * RTOweak + 0.75 * RTOrecent \quad (4)$$

$$RTOrecent = 0.5 * RTOstrong + 0.5 * RTOrecent \quad (5)$$

$$RTOoverall = 0.5 * RTOrecent + 0.5 * RTOoverall \quad (6)$$

The fact that CoCoA uses a constant backoff factor and RTO aging mechanism penalize its performances. The reason why authors in [20] propose a mechanism using a variable backoff factor depending on the estimated RTO called CoCoA+. This mechanism allows avoiding neither quick retransmissions for low RTO values, which can lead to congestion, nor slow retransmissions for large RTO values, which take a long time to retransmit and can lead to unnecessary delay increase. In addition, in the case when the RTO value hasn't been updated for a long time, CoCoA+ adds an incorporated RTO aging mechanism.

Furthermore, in [14], the authors design a 4-state estimator scheme for CoCoA depending on the number of times a packet has been retransmitted. The transaction starts in state 1, and each time a packet is retransmitted, its state increases by one. Each time a packet is successfully transmitted and acknowledged within its stipulated time, its state decreases by one. This allows setting the backoff parameters accordingly.

Nevertheless, in the presence of a high number of packet losses, subsequent updates of the weak RTO estimator can cause some unexpected or unpredictable problems [18]. Since CoAP limits the MAX_RETRANSMIT in four, a new $RTT_{weak}$ might be obtained after the second, third, fourth, or fifth transmission. Thus, the specification of correspondence between each transmission and its CoAP acknowledgment might be not possible. This mechanism may have a great impact on the calculation of the overall RTO. In addition, when the $RTT_{weak}$ is measured after multiple retransmissions, the new calculated RTO might increase in a considerable way compared to $RTO_{init}$.

So, in the aforementioned congestion control mechanisms, the issue of setting a right RTO value with burst traffic is still limited because setting a correct and an accurate RTT of retransmitted packet is hardly obtained. In the next section, we

propose an adaptive RTO based on packet loss and RTT to provide an improved congestion control mechanism that addresses these issues.

## 5 The Proposed Improved CoAP Congestion Control

In addition to the fact that the basic CoAP congestion control doesn't use the RTT of previous transactions to estimate the following RTO, it also doesn't take into consideration the utility of the packet loss ratio as well, to adapt its behavior to network conditions.

The packet loss is defined as the number of packets which failed to reach their destination across a computer network. Packet loss is either caused by link-layer interference or network congestion and it is measured as a percentage of packets lost according to packets sent.

Indeed, in IoT networks, the packet loss is considered one of the big consequences of the network congestion problem. Based on this fact, in this paper, we propose an improved congestion control algorithm based on the packet loss ratio and the RTT value considered in the previous transmission.

Furthermore, in order to provide an adaptive dynamic retransmission timeout which can be suitable for network conditions in the IoT applications, we propose to update the RTO value in each retransmission according to the packet loss ratio. The correlation between actual and previous RTO values seems primordial to adapt the recent RTO value to network conditions, basically the RTT and the packet loss ratio which is in a frequent change. So, there is no need to aging techniques because our RTO is in a frequent change according to the packet loss changes and it will never keep a fixed value for an extended period of time. Thus, the server notifies the client with the packet loss ratio based on sequence numbers of received messages i.e. when the server receives a message, it gets a set of sequence numbers and it recognizes the sequence numbers missed then it calculates a packet loss percentage according to packets sent. In our conception, two scenarios are proposed; (i) if the packet loss ratio is lower than 50%, the RTO value will be updated according to formula (7) in order to prevent unnecessarily long idle time, otherwise, (ii) if the packet loss ratio exceeds 50%, the RTO will be updated in order to correct the loss according to formula (8).

In other words, when the packet loss has a low value ($pl < 0.5$), we conserve nearly the same RTO value as the previous value ($RTO_{recent} \approx RTO_{previous}$), this is in order to reduce idle time (waiting time). On the other hand, when the pl increases ($pl > 0.5$) the RTO conserve as well nearly the same value as the previous value, this is in order to correct the loss of packets. These formulas aim to adapt the RTO calculation to network conditions (RTT and packet loss) by conserving nearly the same value of the retransmission timeout.

Initially, like the basic CoAP specification [15], we initiate the RTO to a random value between ACK_TIMEOUT and ACK_TIMEOUT *ACK_RANDOM_FACTOR. Once the RTO is initiated, a message is sent to the corresponding client. Then after the reception of the message, the receiver calculates the RTT and the packet loss values based on the received packets and the sequence

numbers. Afterward, the formulas (7) and (8) are used in order to calculate the following RTO to use in the next retransmission.

$$RTO_{recent} = RTT * packet\_loss\_ratio + (1- packet\_loss\_ratio) * RTO_{previous} \quad (7)$$

$$RTO_{recent} = RTO_{previous} * packet\_loss\_ratio + (1- packet\_loss\_ratio) * RTT \quad (8)$$

The detailed algorithm of our proposition is drawn in Figure 4.

```
RTO_init=random(ACK_TIMEOUT,ACK_TIMEOUT*ACK_RANDOM_FACTOR)
  RTO = RTO_init
  For (i=1; i<packet.packet_nbr ; i++)
              send_coap_packet ( )
              packet.pl=calculate_packet_loss ( )
              packet.rtt=calculate_rtt ( )
              receive_coap_ack ( )
              if (pl<0.5)
                      then
                      RTO=RTO*(1-packet.pl) + packet.rtt*packet.pl
                      else
                      RTO=RTO*packet.pl + (1-packet.pl)*packet.rtt
              endif
  endfor
```

**Fig. 4.** The algorithm of the proposed adaptive CoAP congestion control

## 6 Simulated Results

In order to evaluate the performance of our proposed mechanism, we perform, in this section, simulations. We carry our evaluations on NS simulator in order to compare the performance of our proposed algorithm with the basic CoAP congestion control and two alternative RTO calculations; classical TCP and TCP Linux. We have used NS-2.34 as simulation tool. Our platform is Ubuntu Linux 14.04 X86 and our computer configurations are Intel I3, 2.4 GHz for the CPU, 8 GB for the RAM and 500Gb for the Hard Drive.

The traffic is generated in a wireless channel. We initially estimate that 12s was sufficient to evaluate the performances due to the fact that the $RTO_{init}$ is equal to 1s. For the same reasons, we opt for the use of a constant bit rate (CBR) in our generated traffic in order to keep the same large number of bits per a short period of time and avoid returning to a short number. As for the data generation rate, we have used the CBR default value because it reflects the case of the generation of data in a real IoT network. We have adopted a basic topology (point to multipoint), thus a server communicating with multiple clients. On the other hand, in our conception, packet loss is due basically to network congestion seeing that we configure our simulation environment so that it doesn't contain any interference source.

Our comparison is performed in terms of the number of dropped packets, the maximum RTO value and the number of successful transactions. We have run the simulation in total 16 times; 4 times for each protocol and in each time we changed the node number. We estimate that these simulations are enough because only one

parameter (node number) changes. After running the simulation, we have extracted from the generated trace file the useful data via Linux line commands in order to plot graphs. The parameters considered in this simulation are detailed in Table 2.

**Table 2.** Simulation parameters considered in our comparison

| Parameter | Value |
|---|---|
| RTO_Init | 1 s |
| Nodes number | 1 to 40 nodes |
| Packet size | 1 KB |
| Link speed | 1.5 Mb/s |
| Link delay | 10 ms |
| Data generation rate | 448 Kb/s |
| Simulation duration | 12 s |

Figure 5 shows the maximum RTO values presented by the proposed algorithm, basic CoAP, classical TCP and TCP Linux in our scenario. Both of the classical TCP and Linux TCP congestion control algorithms present a lower RTO and start to increase proportionally to the increase of nodes number. However, our proposed algorithm presents the lower maximum RTO and outperforms all the algorithms considered in this comparison, this is due to the fact that our algorithm presents a dynamic and controlled retransmission timeout adapted to be appropriate and suitable for the IoT communications particularities. The two mechanisms presented by our proposition effectively limit the growth of RTO values since it is the previous RTO that makes the higher weight in each of formula (7) and (8). On the other hand, the basic CoAP presents slightly high maximum RTO values. Indeed, the basic CoAP underperforms since it uses a fixed range of initial RTO values and does not adapt to the current RTT. Thus, if the real RTT is noticeably below the default RTO range, CoAP reacts slowly to losses, while, if the RTT lies in the RTO range or even exceeds it, spurious retransmissions are likely to happen.

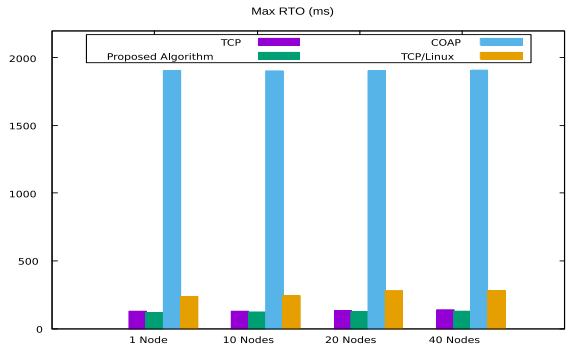


**Fig. 5.** Comparison of the maximum RTO values

Figure 6 shows the percentage of dropped packets presented by each congestion control algorithm considered in our comparison. The clear difference in the percentage of dropped packets between the proposed, CoAP, classical TCP and Linux

TCP algorithms is a result of the different RTO estimations they apply. Furthermore, in our proposed algorithm, the fundamental reason of dropped packets is that the internal buffers of the CoAP reach its limit after the RTO adaptation phase. Once the RTO estimation reaches a steady state, our algorithm optimizes the rate of dropped packets and achieves a good and steady performance.

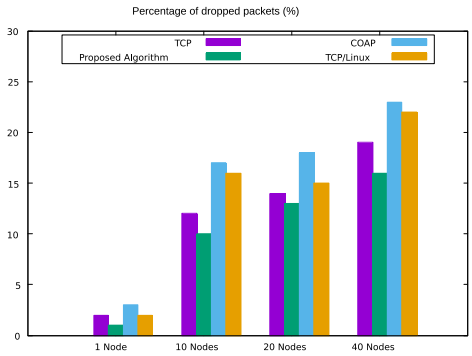


**Fig. 6.** Percentage of dropped packets presented by each congestion control algorithm considered in our comparison

Figure 7 presents the number of successful transactions achieved by different algorithms considered in this paper. Our proposed algorithm shows a slight increase in the number of successful transactions compared to classical TCP and Linux RTO, but a great increase compared to the basic CoAP congestion control according to nodes number used in this scenario. Otherwise, CoAP congestion control presents the lower successful transactions number which is the result of its behavior's insensitivity towards network conditions.

Note that the relatively short duration of every single experiment of 12 s penalizes our proposition. If a use case involves a greater duration, its performance will increase.

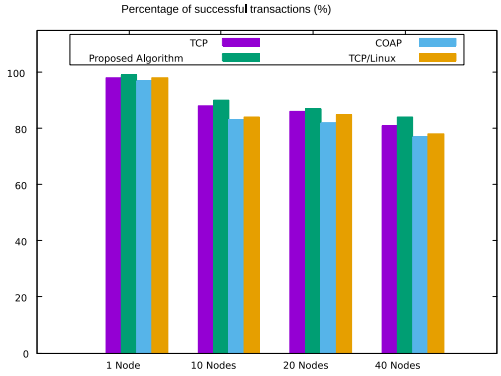


**Fig. 7.** The flow of successful transactions in congested network

As expected, in Figure 8, the number of retransmissions according to nodes number shows that the proposed algorithm presents the lower retransmissions number compared to algorithms considered in this paper but start to increase slightly according to nodes number. This increase is explained by the fact that the proposed algorithm tries to cover the number of dropped packets by retransmissions since the estimation of the RTO for transactions is based basically on the packet loss ratio.

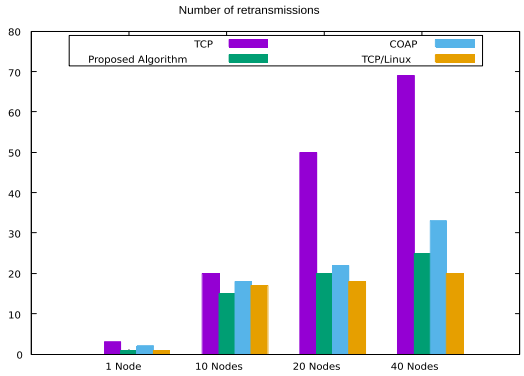


**Fig. 8.** Retransmissions number of congestion control algorithms according to nodes number

The flexibility and the ability to adapt its behavior to different network conditions make our approach apt to outperform the default CoAP congestion control mechanism and TCP calculation algorithms; it is able to increase the number of successful transactions and to reduce the network congestion. Consequently, the proposed congestion control algorithm can maintain high performance in almost all the considered scenarios. Nevertheless, basic CoAP fails to achieve good performance due to its lack of sensitivity to network conditions.

## 7 Conclusion and Future Work

In many critical application fields like industry process and health, the connection of devices must be managed to ensure the reliable data transmission. So, Internet of Things is now connecting different devices in our entourage through the use of WSN based on different protocols. One of the most appropriate protocols for lightweight devices and constrained resources in terms of memory, energy, and computing is CoAP. However, in such a network, the problem of congestion is very frequent. Nevertheless, CoAP offers a basic congestion control insensitive to network conditions using an exponential back-off mechanism that lowers its performances. The challenge is to design a suitable congestion control mechanism for CoAP to ensure a safe network operation while keeping the use of network resources efficient. In this direction, this paper presents an improved congestion control algorithm, adaptive to network condition, for the estimation of RTO value. In order to compare

the performance of our proposed algorithm to other existing algorithms like basic CoAP congestion control, classical TCP, and Linux RTO, we have drawn a simulation under NS2 simulator. The analysis and evaluation of our simulated results show that the proposed mechanism can appropriately achieve higher performance compared to the algorithms considered in this paper. Future directions will consist of applying the idea of the paper in a mobile environment; an environment where nodes are able to move and change their positions frequently, in order to evaluate its performance in such environment.

## 8 References

[1] Eugster, P.T., Felber, P.A., Guerraoui, R., Kermarrec, A.M. (2003). The many faces of publish/subscribe. ACM Computing Survey. 35: 114–131. https://doi.org/10.1145/857076.857078

[2] Shelby, Z., Hartke, H., Bormann, C. (2013). Constrained Application Protocol (CoAP) drafietf core coap 18. RFC 7252, Ver. 17, 18.

[3] In-Jae, S., Doo-Seop, E., Byung-Kwen, S. (2015). The CoAP-based M2M gateway fordistribution automation system using DNP3.0 in smart grid environment. IEEE International Conference on Smart Grid Communications (SmartGridComm), Miami, Florida.

[4] Bergmann, O., Hillmann, K.T., Gerdes, S.A. (2012). CoAP-gateway for smart homes. IEEE International Conference on Computing, Networking and Communications (ICNC), Maui, Hawaii, pp. 446-450. https://doi.org/10.1109/ICCNC.2012.6167461

[5] Krimmling, J., Peter, S. (2014). Integration and evaluation of intrusion detection for CoAP in smart city applications. IEEE Conference on Communications and Network Security (CNS), San Francisco, CA, USA, pp. 73-78. https://doi.org/10.1109/CNS.2014.6997468

[6] Joshi, J., Kurian, D., Bhasin, S., Mukherjee, S., Awasthi, P., Sharma, S., Mittal, S. (2016). Health Monitoring Using Wearable Sensor and Cloud Computing. International Conference on Cybernetics, Robotics and Control (CRC), Hong Kong, China, pp 104 – 108, 2016. https://doi.org/10.1109/CRC.2016.031

[7] Yuan, H., Yugang, N., Fenghao, G. (2014). Congestion Control for Wireless Sensor Networks: A survey. Control and Decision Conference, Changsha, China, pp. 4853-4858. https://doi.org/10.1109/CCDC.2014.6853042

[8] Paxson, V., Allman, M. (2000). Computing TCP's Retransmission Timer. RFC 2988. https://doi.org/10.17487/rfc2988

[9] Davis, E.G., Calveras, A., Demirkol, I. (2013). Improving Packet Delivery Performance of Publish/Subscribe Protocols in Wireless Sensor Networks. Journal of sensors, 13: 648-680. https://doi.org/10.3390/s130100648

[10] Hartke, K. (2012). Observing Resources in CoAP Draft-Ietf-Core-Observe-06. RFC 7641, Ver. 06.

[11] Jaffey, T. (2014). MQTT and CoAP, IoT Protocols. www.eclipse.org/community/eclipse_newsletter/2014/february/article2.php. (Accessed 20 Jan. 2018).

[12] Masek, P., Hosek, J., Zeman, K., Kröpfl, F. (2016). Implementation of True IoT Vision: Survey on Enabling Protocols and Hands-On Experience. International Journal of Distributed Sensor Networks, Article ID 8160282, 1-18. https://doi.org/10.1155/2016/8160282

[13] Chen, X. (2014). Constrained Application Protocol for Internet of Things. http://www.cse.wustl.edu/~jain/cse574-14/ftp/coap/index.html. (Accessed 03 June 2018).
[14] Bhalerao, R., Subramanian, S.S., Pasquale, J. (2016). An Analysis and Improvement of Congestion Control in the CoAP Internet-of-Things Protocol. Annual Consumer Communications & Networking Conference (CCNC), Las Vegas, USA, pp. 889 – 894. https://doi.org/10.1109/CCNC.2016.7444906
[15] Betzler, A., Gomez, C., Demirkol, I., Paradells, J. (2013). Congestion Control in Reliable CoAP Communication. 16th ACM international conference on Modeling, analysis & simulation of wireless and mobile systems MSWiM, Barcelona, Spain, pp. 365- 372. https://doi.org/10.1145/2507924.2507954
[16] Balandina, E., Koucheryavy, Y., Gurtov, A. (2013). Computing the Retransmission Timeout in CoAP. The 13th International Conference on Next Generation Wired/Wireless Networking, NEW2AN, St. Petersburg, Russia, pp. 352-362. https://doi.org/10.1007/978-3-642-40316-3_31
[17] Betzler, A., Gomez, C., Demirkol, I., Kovatsch, M. (2014). Congestion Control for CoAP cloud services. SOCNE conference, Barcelona, Spain, https://doi.org/10.1109/ETFA.2014.7005340
[18] Betzler, A., Gomez, C., Demirkol, I., Paradells, J. (2015). CoCoA+: An advanced congestion control mechanism for CoAP. Ad Hoc Networks, Vol. 33, pp. 126 – 139. https://doi.org/10.1016/j.adhoc.2015.04.007
[19] Paxson, V., Allman, M., Chu, J., Sargent, M. (2011). Computing TCP's Retransmission Timer. RFC 6298. https://doi.org/10.17487/rfc6298
[20] Järvinen, I., Daniel, L., Kojo, M. (2015). Experimental Evaluation of Alternative Congestion Control Algorithms for Constrained Application Protocol (CoAP). 2nd World Forum on Internet of Things (WF-IoT), Milan, Italy, pp. 453 – 458. https://doi.org/10.1109/WF-IoT.2015.7389097
[21] Bormann, C., Betzler, A., Gomez, C., Demirkol, I. (2016). CoAP Simple Congestion Control/Advanced, draft bormann-core-cocoa-00. Ver. draft-bormann-core-cocoa.
[22] Sarolahti, P., Kuznetsov, A. (2002). Congestion Control in Linux TCP. Proceedings FREENIX Track: 2002 USENIX Annual Technical Conference, Berkeley, CA, pp. 49–62.
[23] Betzler, A., Gomez, C., Demirkol, I., Paradells, J. (2016). CoAP Congestion Control for the Internet of Things. IEEE Communications Magazine, Vol. 54, Issue: 7, pp. 154 – 160. https://doi.org/10.1109/MCOM.2016.7509394

## 9 Authors

**Fathia Ouakasse** (corresponding author) MS degree in Telecommunications and Computer Sciences from the National Institute of Posts and Telecommunications, Rabat (Morocco). PhD student in the Laboratory of Applied Mathematics and Computer Science, Faculty of Science and Techniques, Cadi Ayyad University, Marrakesh, Morocco. (fathia.ouakasse@gmail.com).

**Said Rakrak** Professor and researcher in the Laboratory of Applied Mathematics and Computer Science, Faculty of Science and Techniques, Cadi Ayyad University, Marrakesh, Morocco.

Article submitted 30 June 2018. Resubmitted 14 September and 23 November 2018. Final acceptance 23 November 20187. Final version published as submitted by the authors.