

Packet Loss Ratio and Round-Trip Time based Adaptive CoAP Congestion Control for the Internet of Things

OUAKASSE Fathia^{*†} and RAKRAK Said^{*},

^{*}Applied Mathematics and Computer Science Laboratory (LAMAI)
Cadi Ayyad University, Marrakesh, Morocco.

[†]corresponding author:

Summary

Patterns traffic in the Internet of Things IoT is different from the ones in conventional networks due to the use of lightweight devices and constrained resources like Wireless Sensors Network WSN. While everyday objects are generating a large amount of data via the IoT protocols, data is communicated between different corresponding nodes. So, one of the most emerging messaging protocols used to address the needs of these lightweight IoT nodes is Constrained Application Protocol (CoAP). Nevertheless, these constrained devices communicate via a large amount of messages and notifications which cause the network congestion. The challenge is to design a congestion control mechanism for CoAP suitable to ensure a safe network operation while using network resources efficiently. In this context, this paper presents an improved congestion control algorithm for the estimation of 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. Simulated results show that the proposed mechanism can appropriately achieve higher performance compared to the basic CoAP congestion control and alternative algorithms based on TCP.

Keywords:

IoT, CoAP, congestion control, RTO, packet loss, RTT.

1. Introduction

Recently, WSNs have been widely deployed in many IoT applications in order to measure, control or detect physical and environmental events like pressure, humidity, temperature and pollution levels, as well as other critical parameters. Applications usually used to send queries to concerned sensors to retrieve values periodically from the measurements or detections. Moreover, it is estimated that by the year of 2020 more than 26 billion devices will be connected to satisfy a wide range of IoT applications [1].

However, in recent critical applications of WSN which require intervention, such as home automation, industry process control, healthcare, environment monitoring, smart grid, and ambient assisted living, the challenge is getting information when an event of interest

occurs in order to intervene in real-time. In this context, the publish/subscribe model [2] is the most appropriate model covering these requirements. Furthermore, one of the most important protocols based on this model is CoAP [3]. Indeed, 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 such as smart grid [4], building and home automation [5] smart cities [6] and in the healthcare industry, CoAP presents many applications, for example, a mechanism for health Monitoring using a wearable Sensor to provide real-time updates of the patient's status via CoAP protocol is presented in [7].

Although, CoAP has become increasingly proposed and proved its effectiveness in gathering data from smart sensors and controlling constrained devices, the problem of network congestion [8] in CoAP represents the great limitation that hinders the proper functioning of this protocol and causes the loss of packets. Network congestion can also significantly damage the performance of a network, manifesting in increased packet latencies, while a network may even become useless if the congestion collapse occurs [9].

Indeed, the core CoAP specification defines a basic congestion control mechanism which consists of the use of a backoff 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 and this goes back to the fact that it doesn't take into consideration the RTT of a packet since the networks conditions may change frequently because of the dynamic topology and the density of WSN nodes [10].

For all these reasons, in this paper, we propose an improved adaptive congestion control to overcome the

limitation persisting under the use of traditional CoAP congestion. The principle of our improvement consists of the use of both of the loss packets' ratio and the RTT 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 and a variety of alternative algorithms developed for TCP including the Basic RTO and Linux RTO.

The remainder of this paper is organized as follows: a brief presentation of the main aspects of CoAP protocol including reliability, message format, and a detailed basic CoAP congestion control are presented as background in the second section. Then, in the third section, the classical TCP congestion control algorithm and TCP Linux are described. Afterward, in the fourth section, the proposed improved congestion control algorithm is detailed and in the fifth 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 sixth section.

2. Background

2.1 CoAP overview

CoAP has been designed by the Internet Engineering Task Force (IETF) to support IoT with lightweight messaging for devices operating in a constrained environment. CoAP is an application layer protocol based on a REST architecture. It defines two kinds of interactions between end-points: (i) the client/server model which provides as well two interaction types: (1) a one-to-one interaction which means request/reply and (2) a multi-cast interaction; when a Client wants to interrogate servers it makes requests to servers, servers send back responses. Like HTTP, Clients have the ability to manage resources using requests: GET, PUT, POST and DELETE to perform Create, Retrieve, Update, and Delete operations, and (ii) a publish/subscribe model called the observer model [11], where a server, playing the role of the publisher, sends messages of notifications as publications to an observer, playing the role of subscriber, about a resource (event) that the subscriber is interested in receiving.

Unlike HTTP, CoAP doesn't run over TCP, it runs over UDP. Communication between clients and servers is afforded through connectionless datagrams. Retries and reordering are implemented in the application stack. UDP broadcasts and multicasts are also allowed by CoAP for addressing [12]. Otherwise, CoAP is considered more suitable for the IoT domain, this is going back to the fact

that it is possible to build sufficiently basic error checking and verification for UDP to make sure that messages arrived without the significant communication overhead as in the case of TCP [13]. An overview architecture of CoAP protocol is drawn in Fig.1.

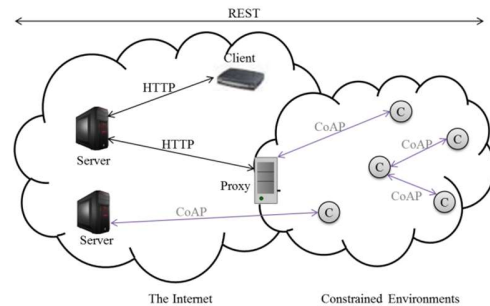


Figure. 1. An overview architecture of CoAP protocol

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 [10]. In the case of a confirmable message an acknowledgment message (ACK) is sent to the sender from the intended recipient as shown in Fig.2a, 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 as shown in Figure.2b [14].

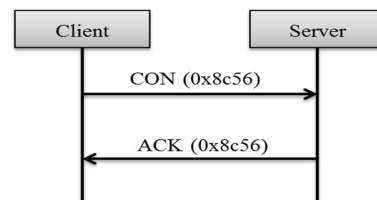


Figure. 2a. Reliable message transport

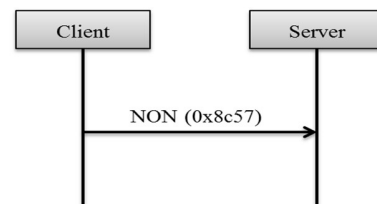


Figure. 2b. Unreliable message transport

Since SSL/TLS are not available to provide security in UDP, CoAP uses Datagram Transport Layer Security (DTLS) on top of its UDP transport protocol for transfers of data [12].

Now, the message format of CoAP is depicted in Fig.3.

31	-	-	16	15	-	8	7	6	5	4	3	2	1	0
Message ID				Code			TKL				T		Ver	
Token (optional)														
Options (optional)														
Payload (optional)							1 1 1 1 1 1 1 1							

Figure. 3. CoAP message format [15]

The header of the message contains:

- ✓ Version (Ver): indicates the CoAP version number.
- ✓ Type (T): indicates the type of the message Confirmable, Non-confirmable, Acknowledgment or Reset.
- ✓ Token Length (TKL): indicates the length of the variable-length Token field (0-8 bytes).
- ✓ Code: can indicate a request, a success response, a client error response, or a server error response. In the case of a request, the Code field indicates the Request Method; in the case of response, a Response Code.
- ✓ Message ID: used to detect message duplication and to match messages of type Acknowledgment/Reset to messages of type Confirmable/Non-confirmable.

The header is followed by Token, Options, and Payload which are optional fields; if there is any.

2.2 CoAP default congestion control

The problem of congestion happens when the traffic load offered to a network approaches the network capacity [16]. 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 [17]. 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 [18]. 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 [15]. Moreover, in Fig.4, the CoAP default congestion control is drawn.

Table1. Default parameter values as specified in CoAP specification

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

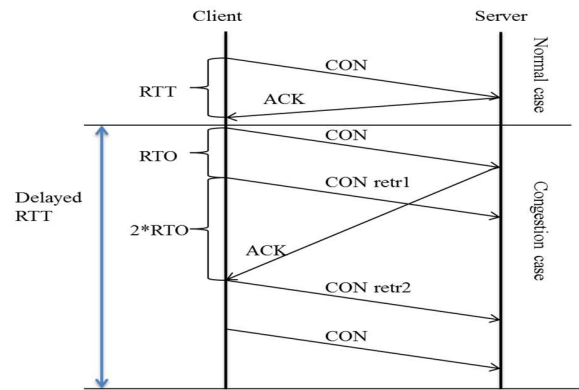


Figure. 4. The CoAP default congestion control

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 [19], thus, it doesn't take into account the value of RTT obtained from previous transactions in order to compute the next RTO. Therefore, it is 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 [20].

So, the calculation of an appropriate RTO is essential to overcome the problem of congestion. In the following, we describe two alternative congestion control algorithms developed for TCP which use formulas based

on the RTT in order to compute the RTO value, including two mechanisms; Basic RTO and Linux RTO.

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 and the history variation of RTT. The specification of this algorithm is proposed by RFC 6298 [21]. 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 [18]:

$$\begin{aligned} SRTT &= RTT \\ RTTVAR &= RTT/2 \\ RTO &= SRTT + \max(G, K \cdot RTTVAR) \end{aligned} \quad (1)$$

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

$$\begin{aligned} SRTT &= (1 - \alpha) \cdot SRTT + \alpha \cdot RTT & (2) \\ RTTVAR &= (1 - \beta) \cdot RTTVAR + \beta \cdot |SRTT - RTT| & (3) \\ RTO &= SRTT + \max(G, K \cdot RTTVAR) & (1) \end{aligned}$$

Thus the formula (1) is used to estimate the RTO value to be used in the following transmission. When the RTO timer expires, the RTO value is doubled [22]. According to RFC 2988 [9], the value of the constant K in (1) is 4. Furthermore in (2) and (3) 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 Fig.5.

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 [9].

Thus, it is recommended to choose G value not greater than 100 ms [21]. At the same time, G should be at least one order of magnitude smaller than the RTT [23].

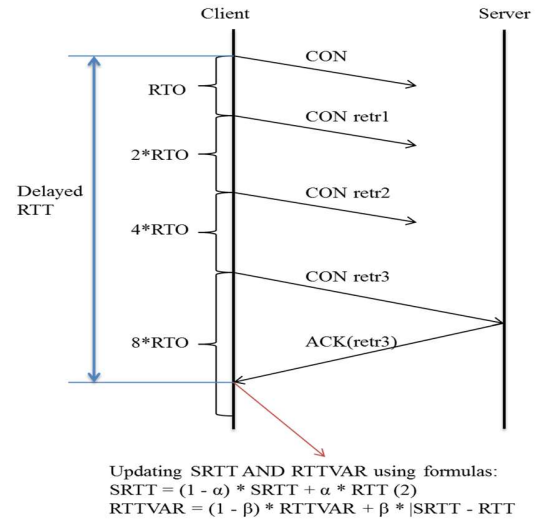


Figure. 5. The calculation of RTT in the classical TCP congestion control

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 [24]. 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 [25].

4. The proposed adaptive CoAP congestion control algorithm

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.

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 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; thus, the server notifies the client with the packet loss ratio based on sequence numbers of received messages. Two scenarios are proposed; (i) if the packet loss ratio is lower than 50%, the RTO value will be updated according to formula (4) 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 (5).

Initially, like the classical TCP defined in RFC 6298 [21], we set the RTO value to 1s. Then after the reception of the first packet loss ratio and the RTT value, the formulas (4) and (5) are used in order to calculate the following RTO to be used in the next retransmission.

$$RTO_{recent} = RTT * packet_loss_ratio + (1 - packet_loss_ratio) * RTO_{previous} \quad (4)$$

$$RTO_{recent} = RTO_{previous} * packet_loss_ratio + (1 - packet_loss_ratio) * RTT \quad (5)$$

The detailed algorithm of our proposition is drawn in Fig.6.

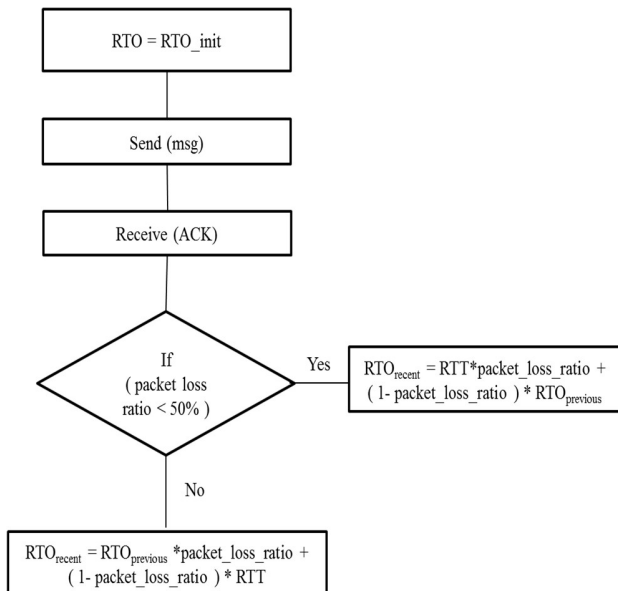


Figure. 6. A proposed algorithm for our improved CoAP congestion control

5. Simulated results

In order to evaluate the performance of our proposed mechanism, we perform, in this section, simulations. We carry our evaluations on NS2 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. Our comparison is performed in terms of the number of dropped packets, the maximum RTO value and the number of successful transactions. The parameters considered in this simulation are detailed in Table 2.

Table 2. Simulation parameters considered in our comparison

Parameter	Value
RTO_Init	1s
Nodes number	1 to 40 nodes
Packet size	1 Ko
Link speed	1.5 Mbps
Link delay	10s
Simulation duration	12s

Figure.7 shows the maximum RTO values presented by the proposed algorithm, basic CoAP, classical TCP and TCP Linux in our scenario. Both 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 going back 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 formula (4) and (5). 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.

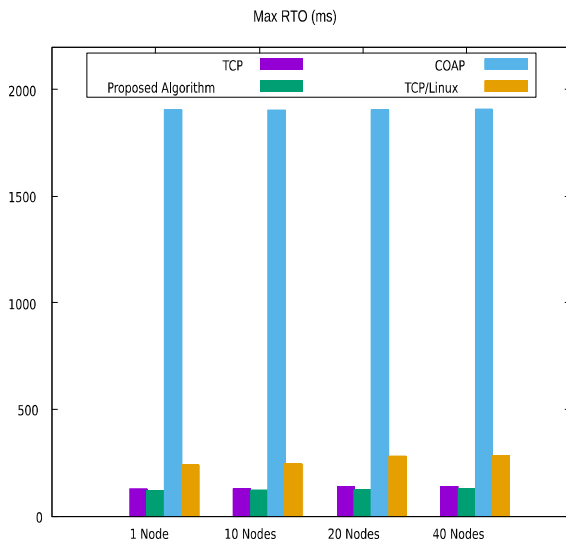


Figure. 7. Comparison of the maximum RTO values

Figure.8 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.

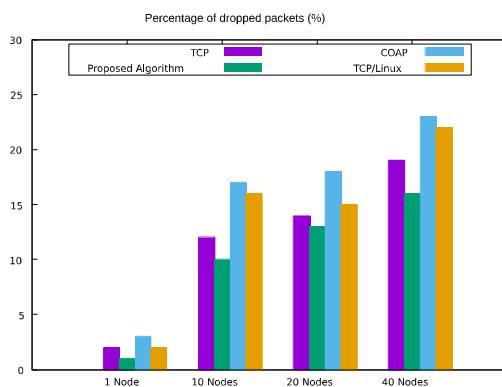


Figure. 8. Percentage of dropped packets presented by each congestion control algorithm considered in our comparison

Figure.9 present 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 each single experiment of 12 s penalizes our proposition. If a use case involves a greater duration, its performance will increase.

As expected, in Fig.10, 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 try 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.

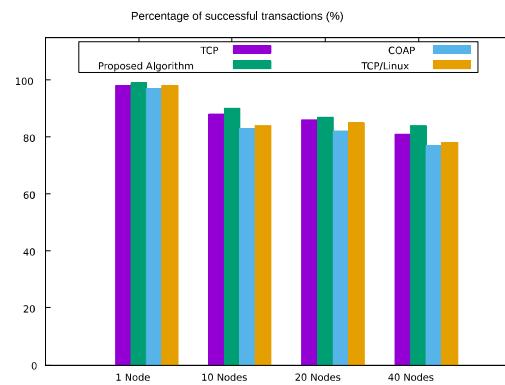


Figure. 9. The flow of successful transactions in congested network

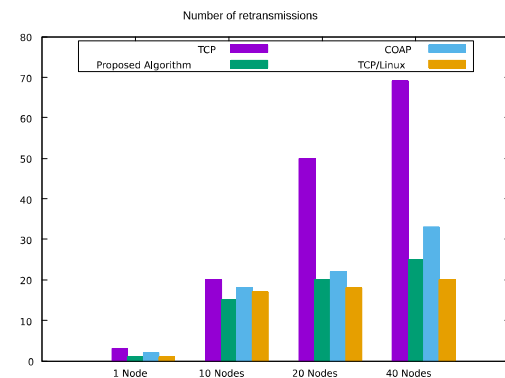


Figure. 10. Number of retransmissions of congestion control algorithms according to nodes number

Thanks to its flexibility and its ability to adapt its behavior to different network conditions, our proposition consistently outperforms 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.

6. Conclusion

The Internet of Things (IoT) is now offering the ability to transfer data over a network without requiring human-to-human or human-to-computer interaction. It is 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, thing which lowers its performances. The challenge is to design a congestion control mechanism for CoAP suitable to ensure safe network operation while using network resources efficiently. Thus, in this paper, we present an improved congestion control algorithm, adaptive to network condition, for the estimation of RTO value to be used for the next transaction based on the packet loss ratio and the Round Trip Time RTT of the previous transmission. 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 draw a simulation under NS2 simulator. Simulated results show that the proposed mechanism can appropriately achieve higher performance compared to the algorithms considered in this paper.

References

- [1] Middleton P, Kjeldsen P, Tully J. Forecast: The Internet of Things, Worldwide, Gartner, Inc., Tech. Rep. 2013.
- [2] Eugster PT, Felber PA, Guerraoui R, Kermarrec AM. The many faces of publish/subscribe. *ACM Computing Survey*. 35: 114–131, 2003.
- [3] Shelby Z, Hartke H, Bormann C. Constrained Application Protocol (CoAP) draftietf core coap 18. RFC 7252, Ver. 17, 18. 2013.
- [4] In-Jae S, Doo-Seop E, Byung-Kwen S. The CoAP-based M2M gateway for distribution automation system using DNP3.0 in smart grid environment. *IEEE International Conference on Smart Grid Communications (SmartGridComm)*, Miami, Florida, 2015.
- [5] Bergmann O, Hillmann KT, Gerdes SA. CoAP-gateway for smart homes. *IEEE International Conference on Computing, Networking and Communications (ICNC)*, Maui, Hawaii, pp. 446–450, 2012.
- [6] Krimmling J, Peter S. 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, 2014.
- [7] Joshi J, Kurian D, Bhasin S, Mukherjee S, Awasthi P, Sharma S, Mittal S. Health Monitoring Using Wearable Sensor and Cloud Computing. *International Conference on Cybernetics, Robotics and Control (CRC)*, Hong Kong, China, pp 104 – 108, 2016.
- [8] Yuan H, Yugang N, Fenghao G. Congestion Control for Wireless Sensor Networks: A survey. *Control and Decision Conference*, Changsha, China, pp. 4853–4858, 2014.
- [9] Paxson V, Allman M. Computing TCP's Retransmission Timer. RFC 2988, 2000.
- [10] Davis EG, Calveras A, Demirkol I. Improving Packet Delivery Performance of Publish/Subscribe Protocols in Wireless Sensor Networks. *Journal of sensors*, 13: 648–680, 2013.
- [11] Hartke K. Observing Resources in CoAP Draft-Ietf-Core-Observe-06. RFC 7641, Ver. 06, 2012.
- [12] Jaffey T. MQTT and CoAP, IoT Protocols. www.eclipse.org/community/eclipse_newsletter/2014/february/article2.php. (2014 accessed 20 Sep. 2017).
- [13] Masek P, Hosek J, Zeman K, Kröpl F. Implementation of True IoT Vision: Survey on Enabling Protocols and Hands-On Experience. *International Journal of Distributed Sensor Networks*, Article ID 8160282, 1–18, 2016.
- [14] Chen X. Constrained Application Protocol for Internet of Things. <http://www.cse.wustl.edu/~jain/cse574-14/ftp/coap/index.html>. (2014 accessed 03 Jul. 2017).
- [15] Shelby Z, Hartke K, Bormann C. The Constrained Application Protocol (CoAP). *Internet Requests for Comments, RFC Editor, RFC 7252*, 2014.
- [16] Bhalerao R, Subramanian SS, Pasquale J. 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, 2016.
- [17] Betzler A, Gomez C, Demirkol I, Paradells J. 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, 2013.
- [18] Balandina E, Koucheryavy Y, Gurtov A. Computing the Retransmission Timeout in CoAP. *The 13th International Conference on Next Generation Wired/Wireless Networking, NEW2AN*, St. Petersburg, Russia, pp. 352–362, 2013.
- [19] Betzler A, Gomez C, Demirkol I, Kovatsch M. Congestion Control for CoAP cloud services. *SOCNE conference*, Barcelona, Spain, DOI: 10.1109/ETFA.2014.7005340, 2014.

- [20] Betzler A, Gomez C, Demirkol I, Paradells J. CoCoA+: An advanced congestion control mechanism for CoAP. *Ad Hoc Networks*, Vol. 33, pp. 126 – 139, 2015.
- [21] Paxson V, Allman M, Chu J, Sargent M. Computing TCP's Retransmission Timer. RFC 6298, 2011.
- [22] Järvinen I, Daniel L, Kojo M. 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, 2015.
- [23] Bormann C, Betzler A, Gomez C, Demirkol I. CoAP Simple Congestion Control/Advanced, draft bormann-core-cocoa-00. Ver. draft-bormann-core-cocoa, 2016.
- [24] Sarolahti P, Kuznetsov A. Congestion Control in Linux TCP. *Proceedings FREENIX Track: 2002 USENIX Annual Technical Conference*, Berkeley, CA, pp. 49–62, 2002.
- [25] Betzler A, Gomez C, Demirkol I, Paradells J. CoAP Congestion Control for the Internet of Things. *IEEE Communications Magazine*, Vol. 54, Issue: 7, pp. 154 – 160, 2016.

Fathia OUAKASSE MS degree in Telecommunications and Computer Sciences from the National Institute of Posts and Telecommunications, Rabat (Morocco) in 2011. PhD student in the Laboratory of Applied Mathematics and Computer Science, Faculty of Science and Techniques, Cadi Ayyad University, Marrakesh, Morocco.

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