Avrupa Bilim ve Teknoloji Dergisi
Sayı 20, S. 171-180, December 2020

Araştırma Makalesi

www.ejosat.com ISSN:2148-2683

European Journal of Science and Technology
No 20, pp. 171-180, December 2020

Research Article

AQM-of-Things: Special Queue Management Approach for Internet of Things

Kerem Aytaç[1*], Ömer Korçak[2]

[1] Marmara University Computer Engineering Dept., Koç Digital R&D Center, İstanbul, Türkiye (ORCID: 0000-0003-4794-4036)
[2] Marmara University Computer Engineering Dept., İstanbul, Türkiye (ORCID: 0000-0003-4419-556X)

(First received 27 June 2020 and in final form 11 October 2020)

(**DOI:** 10.31590/ejosat.759077)

ATIF/REFERENCE: Aytaç, K. & Korçak, Ö. (2020). AQM-of-Things: Special Queue Management Approach for Internet of Things. *European Journal of Science and Technology*, (20), 171-180.

Abstract

Although Internet of Things (IoT) networks are massively deployed in many different areas, there are significant problems regarding the network topology and capacity, due to low smartness level of IoT devices and cost matters. Congestion and queue management especially in an IoT network buffer is one of the most important subjects that need to be considered. In this paper, we propose a novel game-theoretical approach to manage the queue and avoid possible congestion, by adding a little intelligence to dumb nodes with a lightweight method called AQM-of-Things (AQMoT). Extensive-form game formulation is used for defining decision making criteria of both IoT nodes (when to send) and gateways (when to drop). We describe a game model according to the queue level as well as other network conditions. Thus, a novel congestion avoidance method is proposed, where senders care about the gateway's current situtation with a very lightweight game theoretical algorithm. We also demonstrate a conceptual comparison with alternative queue management approaches, and conclude that the proposed AQMoT approach has important advantages especially in IoT domain.

Keywords: Internet Of Things, Active Queue Management, Congestion Avoidance, Game Theory

Nesnelerin Kuyruk Yönetimi: Nesnelerin İnternetine Özel Kuyruk Yönetim Yaklaşımı

Öz

Birçok farklı alanda Nesnelerin İnterneti (IoT) kullanımının giderek yaygınlaşmasına rağmen, günümüzde hala cihazların düşük işlem yetenekleri ve maliyet endişeleri sebebiyle ağ topolojisi ve kapasitesiyle ilgili büyük problemler bulunmaktadır. Farklı öncelik seviyelerine sahip çok fazla veri trafiği oluşturan düşük işlem yetenekli sensörlerin ağ topolojisi içerisinde sayısı artırılırken genelde ağın kısıtları IoT'de göz ardı edilmektedir. Bu makalede, kuyrukları yönetebilmek ve ağdaki olası tıkanmaları engellemek için bu ağ noktalarına oyun teorisi yaklaşımlı bir aktif kuyruk yönetimi yaklaşımı önerilmiştir. AQM-of-Things (AQMoT) adı verilen bu yaklaşımda, ağ düğümlerine az miktarda ama etkili bir zeka kazandırılması önerilmektedir. Genişletilmiş bir oyun modeli formülize edilerek hem IoT cihazlarının ne zaman veri göndereceklerine dair, hem de haberleşme ünitelerinin ne zaman verileri düşüreceklerine dair karar verme mekanizmaları belirlenmiştir. Oyun modelinde kuyruk uzunluğu ve diğer ağ durumları dikkate alan bir yöntem geliştirilmiştir. Bu şekilde, diğer algoritmaların aksine haberleşme ünitelerinin durumunu hafif ve oyun teorisini baz alarak gözeten yenilikçi ve tıkanıklıktan kaçınan bir yaklaşımı öne sürüyoruz. Ayrıca AQMoT yaklaşımı alternatif kuyruk yönetim yaklaşımları ile kavramsal olarak karşılaştırılmış ve özellikle IoT alanında önemli avantajları olduğu sonucuna varılmıştır.

Anahtar Kelimeler: Nesnelerin İnterneti, Aktif Kuyruk Yönetimi, Tıkanıklıktan kaçınma, Oyun Teorisi

[*] Corresponding Author: Marmara Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü, İstanbul, Türkiye, ORCID: 0000-0003-4794-4036, kerem.aytac@marun.edu.tr

1. Introduction

In IoT world, there are many nodes with different sizes including sensors, actuators, gateways or some middle-nodes such as routers, access points, extenders, relays which create a crowded network area as a result. According to a survey, there are 50 billions of devices are connected to the Internet by 2020 where it will exponentially increase to 500 billions of devices as of 2030 (Ataç et al. 2019). Within these nodes, there are lots of data traversing, where important portion of them are possibly redundant or have low-priority. An IoT network is typically considered as constrained node networks (which is an IETF definition) as it includes many constrained dumb endpoints. These endpoints produce lots of data which they sense and send them to related destinations. Many of them have no intelligence or any idea about when or how frequently to send their sensed data. They do not care if a network node is congested or not.

There are many protocols suitable for IoT such as MQTT (Message Queuing Telemetry Transport) or AMQP (Active Message Queueing Protocol) which are lightweight but reliable protocols. These protocols stand on TCP/IP. Both of them uses queueing methodology to satisfy reliability and to avoid data loss during offline or communication failure states. On application layer, MQTT works with subscription/publish methodology which queues the messages before leaving the system and publishes to a specified topic which is considered also as application layer (Naik, 2017). MQTT does not implement a specific congestion control per se, but underlying TCP can handle this congestion control. Moreover, HTTP is still an irreplaceable protocol which is also used in IoT world even it is not lightweight like MQTT. Regardless of what is used, all of them relies on TCP/IP protocol and all of them needs a TCP congestion control. Recently new TCP congestion control algorithms are proposed for IoT networks (Verma et al. 2020). In addition to TCP-congestion control, Active Queue Management methodologies also used to maintain end-to-end non-congested network. One difference is that they reside at different layers where TCP-Congestion is handled in transport layer and Active Queue Management (AQM) is handled in network layer (Baker et al. 2015). Security is also a critical point over MQTT as there is always an option to use the non-secure MQTT which works on 1883 port as default. Applying a security over MQTT level, will also affect to lower levels, such as transport layer or network layer. There will be some impacts while implementing an algorithm at network layer. For example, a related work proposed an algorithm to authenticate and encrypt communication between the gateway and MQTT broker by also approaching from a new aspect to provide simultaneous encryption and MQTT-based communication by utilizing physical I2C property of the ARM Cortex M3 (Toğay et al. 2019). As this approach encrypts the message by starting from authentication phase, it should have some impacts while implementing some AQM algorithms which will be discussed later in this paper.

Another vital part of IoT are the gateways which act as bridges between sensors and the server/cloud layer. Lots of nodes send data to same gateway, and gateway should handle this load, otherwise a congestion may occur and all the communication between nodes will be down. Most sensor nodes are so dumb that they cannot be configured generically, flexibly. They are aimed to send what they sense within pre-configured time intervals. One can't tell them to stop while there is a huge load on a gateway or a main node. Even if we can tell them to stop, One need to speak in their different and separate languages. Thus, implementing a common interface would not be easy and efficient. So, instead of implementing on application layer, implementing on network layer would be easier and extendable.

Another method to handle TCP congestion on IoT network is to use Explicit Congestion Notification (ECN) (Gomez, 2019). In this approach, they define dumb devices such as sensors, actuators as constrained nodes, and clouds as unconstrained nodes. There might be direct connection between constrained and unconstrained nodes where there might be a middleware between them as well. ECN allows a node to signal via the IP header of a packet to notify whether congestion is about to happen, for instance, when a queue size reaches a certain threshold.

In aspect of AQM, the well known approach is Random Early Detection (RED) (Floyd et al. 1993) which calculates the average size of queue and drops the packets with defined possibilities between specific minimum and maximum queue thresholds. This satisfies proactive queue congestion control. This method is too simple but also generic. This approach inspires the researchers to create more RED-based AQM approaches to fit specific domains.

In the literature, there are several extensions of RED. Weighted RED (WRED) defines different queue thresholds for different traffic classes and it is likely better at applying QoS-Sensitive congestion avoidance. It has reasonable approach to drops and marks packets. Although WRED is limited in terms of number of traffic flow types, it is still more viable for some routers (Freed et al. 2006). Another extension of RED is Adaptive RED (ARED) (Feng et al. 1999) which tries to adapt itself to be more or less aggressive according to the observation of the average queue length. Caring with the content, another extension of RED is XRED which is a content aware approach (Hassan et al. 2003). This approach is similar to our proposed approach which contributes to IoT networking. The similarity and differences will be defined in Section III in detail.

GREEN (Generalized Random Early Evasion Network) (Feng et al. 2002) is another congestion avoidance solution. A proactive queue management algorithm is proposed to ensure higher degree of fairness between flows. This algorithm uses more intelligent drop possibility calculation. It considers some network conditions like MSS, RTT or outgoing link capacity. So that, it calculates more adaptive drop possibilities rather than random calculated possibilities and thus, can create fairness between flows. PIE (Proportional Integral controller Enhanced) (Pan et al. 2013) is robust and optimized for various network scenarios which does not require per-packet extra processing, so that causes very small overhead and ease of implementation and deployment at both device and software side. This algorithm uses different approach of drop probability calculation by not only using the current estimation of the queuing delay, but also sensing the direction of where the delay is shifting, which is not a simple random drop possibility.

Grazia et al (Grazia et al. 2017) provides a simulation-based comparison of the selected ones of these algorithms in terms of goodput, throughput, RTT variation, fairness, etc. in order to study suitability of these algorithms for an IoT environment. They conclude that although some algorithms have no advantages over IoT network, ARED has a good performance over specifically high congested IoT networks. PIE seems to be the winner in any aspect of stress levels.

Another related approach is to provide fairness by penalizing unresponsive (or aggressive) traffic in case of network congestion. CHOKe algorithm is widely researched fairness-oriented AQM algorithm which effectively punishes unresponsive traffic, although it is stateless and easy to implement. In order to improve the performance of CHOKe (Pan et al. 2000) (Eshete et al. 2013), numerous extensions are proposed, such as CHOKeR (Lu et al. 2013), CHOKeH (Abbas et al. 2018), D-PAC (Hu et al. 2018), etc. These approaches further reduce the ratio of unresponsive traffic by making more match-and-drop comparisons. There is also another recent study which provides a fairness-oriented AQM algorithm called Hash Table and Circular Buffer (HTCB) that is inspired from CHOKe. This approach utilizes a hash table to identify aggressive traffic and aims to improve performance in network nodes with low buffer space. (Hu et al. 2020)

Although all the mentioned approaches are valuable, they have still some limitations and deficiencies for IoT domain as we will discuss in Section III. In this paper, we will adopt a very simple but powerful intelligence on network layer of IoT nodes to use network fairly and feasible, moreover to handle a congestion and queue management. The main contribution of this paper can be summarized as follows. We propose an approach where any part of network (including source nodes) will give a hand to handle the congestion and queue management. We propose a novel decision-making mechanism for both the sender (whether to send a packet or not) and the receiver (whether to drop a packet or not) based on an extensive-form game model. In the game model, we take account of priority and size of packets, network conditions in the network side and also network conditions between the sender and the destination. On the contrary of most AQM algorithms, we avoid most of the packet drops in gateway by a light-weight decision making mechanism and we avoid dumb sensors to exploit network nodes with sending too much redundant data.

Rest of the paper is organized as follows. Next section describes a general problem definition and detailed explanation of proposed novel AQM implementation. Section III decribes the efficiency of the proposed implementation by comparing with existing methods in various performance metrics. Finally, Section IV concludes the paper.

2. Background and Algorithm Proposal

2.1. Background

In IoT, many kinds of topologies can be encountered. Within a simple network, there might be thousands of devices connected to each other. In some topologies, only one gateway takes care all the sensors. Where in some topologies there are some mid-gateways that are responsible from a portion of sensors and collect the data and process to another main gateway to push the data to Internet as shown in Figure 1.

MQTT protocol is the most common way to communicate. It uses the advantage of having a queue mechanism which keeps the messages in the queue when there is a connectivity problem, bottleneck, low-rates etc. When possible, it tries to consume the queue and send the messages to the destination. As IoT environments are prone to weak networks, there are many things to consider. Such as, if sender fills up its queue with message containing sensed values, and when a connection is up, it bursts all the messages in a dummy way. Sensors never consider if a gateway is tired or not. They always send over and over. They like exploiting the network, they do not recognize what a fairness is. So, they are such dumb that no gateway can tell them to stop. In an IoT environment, all the sensors vary, thus not identical. So, while a dumb sensor exploiting the network, some important sensors cannot send their important values which might cause an alert due to congestion. At these times, we wish to implement a fairness amongst sensors considering duplicate, unimportant or malfunctioned values. So, to satisfy all the requirements mentioned, we propose a novel AQM method which is called AQM of Things (AQMoT). Using this AQM approach, we try to handle the congestion within a network while satisfying some fairness and also a little intelligence for sensors or gateways in the network layer.

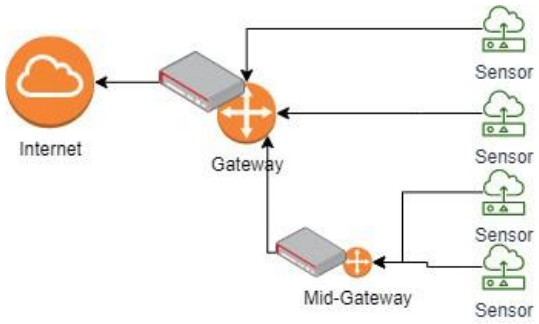

Figure 1. An IoT Sample Environment where sensors directly connected to one and main gateway which tunnels to internet

2.2. AQM-of-Things

2.2.1. Concept

The main concept is to think from the angle of sensors and gateways. A source (sensor, mid-gateway) will send message to a destination (gateway, communication node, mid-gateway). A sensor should consider the possibility of a congestion at the destination and also consider whether its message is important or not, whether it is big or small, and also consider the network status if it is weak or strong. So, if we impersonate as sensor, we'd not send our messages if they are not important and the network is weak and destination possibly in critical queue levels. We'd keep the message in the queue or drop it. Because our message will be dropped without enqueued or lost in network, why should we lose effort intentionally. Moreover, we'd try our chance sending the message if it is important even if destination queue is possibly congested. We can increase the number of examples. We'd also send our messages if it is big and unimportant, but destination has empty buffer, because the road is fully open and destination will accept, why not using it. It also keeps a good utilization over network. Many AQM algorithms have some important thresholds for deciding what to do with the packet. For example, accepting it or dropping it with a defined possibility according to the average queue. Calculation of average queue length also differs from method to another method. Such as, RED uses a simplistic average calculation like moving average where WRED depends on the previous average as well as the current size of the queue. Moreover, all of them define a different drop function beyond a defined threshold named as Q_{min}. A drop function defines the possibility to drop a packet when it tries to enqueue itself to the queue. It is not more than a spinning a roulette to decide whether

to drop or not. It is not actually intelligent way to drop these packets. Moreover, what if sender is aware of the occupancy at the destination; would it still send the packet if its packet was going to be dropped or possibly not accepted? Why should destination always try to orchestrate everything where sender can be pushed for a cooperation easily? Let us give an analogy. In an online meeting there is a teacher and some students isolated from each other around him/her. In a traditional way, when students are talking at the same time, teacher tries to understand the conversations one by one. If he/she cannot understand some of them, ignore and focus other students to get the point. And then change the focus to another student who was ignored before, so on so forth. Now let us try this approach. Teacher notifies the students about his/her condition that there are some students already speaking. So, a student will understand that their teacher is already occupied with another conversations, and if it is not important, student may wait to speak or give up. Thanks to cooperation, teacher will understand and reply to everyone efficiently. So, how are we supposed to implement such an intelligent but also simplistic approach briefly mentioned above? Using some artificial intelligence or machine learning approaches may not be better as they will be heavy for our layer. Here we introduce a game theoretical approach which is very light and easy to implement and scalable.

2.2.2. Implementation

In this section we will describe details of the proposed AQMoT algorithm. We consider a two-player game, where the first player (player 1) is the sender and the second player (player 2) is the destination which can be gateway, a communication node or a mid-gateway etc. In this game, player 1 will not be waiting for a possible signal from player 2 saying that "I'm congested, please stop!". We will try to figure out how possibly player 1 is congested or congestion-candidate using some simple information such as RTT, time-outs, dupacks etc. Moreover player 2 will be transparent to player 1 in terms of queue status so that player 1 will be able to understand if its packet will be dropped at player 2 side if the packet manages to reach. We will be touching these details further later. The game is quite simple. We will try to reduce the following story to game theory payoff tables (Aumann et al. 1985).

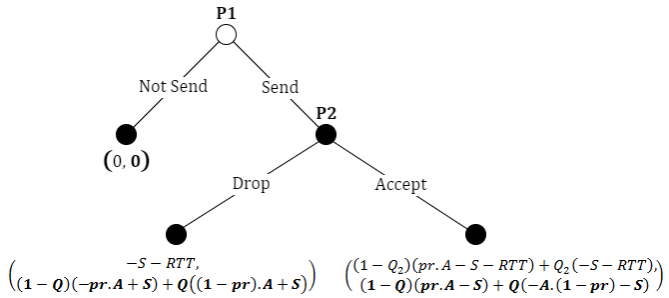

Figure 2. Extensive-form game formulation, where destination is Player 2 and source is Player 1

In Figure 2, an extensive form game model (Kuhn et al. 2016) is illustrated where the first action (Send, Not Send) is played by the player 1 (P1 - sender), and the second action (Drop, Accept) is performed by the player 2 (P2 - receiver). The payoff values of the players in the game outcomes depend on the variables defined in Table 1. Both sides are focused to grant the greatest payoff by their action. Player 1 has the advantage of observing the latest status of Player 2. Q is the percentage of occupancy (busyness) of queue at the destination illustrated as in Figure 3. This function is similar to the drop function for other AQM algorithms like RED and WRED. Q_{min} is a minimum threshold for the destination to define itself as empty. Until this threshold in terms of Queue Average (Q_{avg}), destination is not tended to drop any packet, but accept all. Q_{max} is another threshold which is close to maximum possible queue length of destination (buffer size). Beyond this threshold, destination is fully tended to drop all packets to keep its buffer at ease not to cause overflown buffer or some network problems. Between Q_{min} and Q_{max} thresholds, destination defines its occupancy as linear and tended to drop some of the packets by evaluating the packet size and its priority. In this implementation, without loss of generality, we will set our occupancy function as linear, while other functions (quadratic, exponential, etc) can also be used as desired. Q_{min} and Q_{max} are pre-set and static values just after the network is up and shared with any source via handshake process. Current occupancy percent (Q) is also shared by gateway (the destination) to sensors (the sources) when a sensor establishes a new connection via handshake, and with any acknowledge packets for incoming packets as illustrated in Figure 4. Q_2 is another information used at the side of sender. This is based on Q plus other circumstances like timeouts, dupacks. If Q_2 is tended to be greater than 100% which is not feasible, than it is assumed as 100%. The calculation methodology will be handed over in the next section. If player 1 (the sender) doesn't see any possible overhead at its own side and find out the possible action of player 2 (the destination) which is an "accept" for a low-prioritized, small sized packet under normal circumstanced network, both player 1 and player 2 will receive a good amount of payoff. The more prioritized packet is, the more payoff will be received by sensor and gateway. We will try both sides to boost to complete a transfer with high-prioritized packets. But also, sender will pay the cost of size of packet and average RTT. If packet is big and average RTT is high, then player 1 needs to pay more from earnt payoff. It is assumed as an effort which should be reduced from total award. On the other hand, if there is a possible high occupancy at player 2 (the destination), and there is some network issues which is sensed from timeouts or dupacks and still player 1 tries to send a packet, as player 2 may not receive it or drop it even if received, there will be futile effort at the side of Player 1. So, there will be no awards, but effort is lost in terms of S and RTT.

Table 1. Variables of the proposed game model

pr	*Priority of packet from 0 to 1.*
S	*Size of packet from 0 to 1. It is calculated as the ratio of the packet size to the Maximum Segment Size (MSS).*
RTT	*Moving average of RTT occurred till then. 0 means low RTT, 1 means very high RTT.*
A	*An award. (-A stands for a penalty.) It should be greater than sum of maximum possible value of S and RTT.*
Q	*The percentage of busyness of queue at destination according to defined function.*
Q_2	*Drop probability at source. It is based on Q and additionally some other decision inputs (like timeouts, dupacks).*

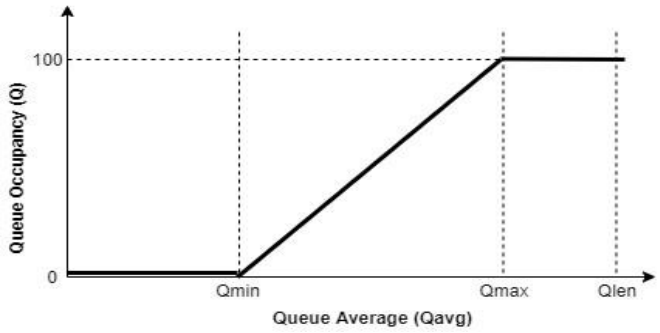

Figure 3. Queue occupancy function by queue average

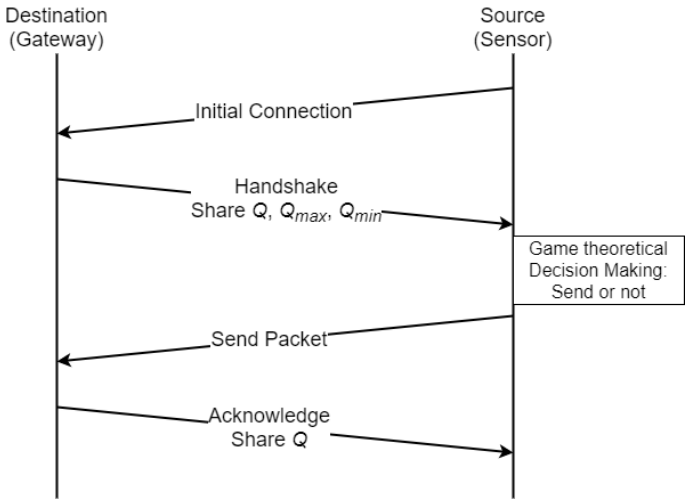

Figure 4. Flow diagram of queue information sharing from gateway to sensor

Now let us describe the rationale behind the formulated game illustrated in Figure 2. If receiver has low or no occupancy (Q is nearly 0), sending a packet will grant a payoff correlated with the priority of the packet. Low-prioritized packet grant low payoff etc. Player 1 always pay for the size of packet and packet delay RTT from its award. An award A will be deterministic for our players behavior. Increasing A will make both sides greedy and to grant it both sides will try to complete a packet transfer even in an occupied destination and loose network. Decreasing A will make them to be at the safe side. They will take less risk. Accepting the packet for player 2 will give same amount of award excluding RTT as RTT does not matter at the gateway side. It is the round trip delay experienced by the sender (sensor, in our scenario).

Now let us consider the outcome where player 1 sends and player 2 drops in a non-occupied queue. Player 1 will get the same reward mentioned above, but player 2 will receive a penalty correlated with the priority of packet. This action is actually undesired. We try to avoid player 2 not accepting packets while available to keep the system utilized. Last action for Figure 2, not sending the packet in an available system which makes no difference at gateway side. So that gateway will not be granted or fined any payoff. Player 1 also never receives anything. We want to keep them away from this lazy action by zero-grants as much as not needed. But if needed, they would profit from loss.

Let us examine the scenario where player 2 is occupied. If player 1 sends the packet to occupied player 2, then player 1 must pay for the futile effort ($-S - RTT$). In all payoffs, according to the Q value, left-hand side or right-hand side will be dominant. If Q is high, then the side which is multiplied with Q is dominant, if Q is low then the side which is multiplied with $(1 - Q)$ is dominant. If the packet is too prioritized, then the punishment aimed to be less due to the left-hand side. We do not want to blame too much if player 1 wants to send a very prioritized packet to an occupied destination. Trying to accept a packet for player 2, will make it lose in the same way excluding RTT. So, for the next scenario, player 1 sends to occupied player 2 and player 2 drops; player 1 will be fined for the efforts done, but player 2 will be granted with an award of A by dropping the packet. This is also correlated with the unimportance of packet. Not accepting a low-prioritized packet will grant more as usual.

2.2.3. Numerical Results

In this title, we will simulate how AQM algorithm works with real life use cases. We will instantiate some environments to have better understanding about the proposed AQM algorithm. Now let us give some example environments and analyze the corresponding game models. Let us imagine an environment where the sensor wants to send a medium sized packet (S =0.5) with medium priority (pr =0.5) on a high-speed link with low RTT value. Award is set to 4 which is a moderate incentive value and there is no occupancy at the destination and no recent timeouts or dupacks, thus $Q = Q_2 = 0$. The normal-form representation of the extensive form game for this scenario is illustrated in Table 2. In this game, Send, Accept is a Nash Equilibrium (Kuhn et al. 2016), because both do not gain more by changing their actions. For the sender, it would be OK to send the packet, because it will be awarded with a reward of 1.4 and also it obviously knows that destination will accept the packet.

Table 2. Payoff Table for given environment

		Player 2	
		Accept	**Drop**
Player 1	**Send**	(1.4 , 1.5)	(-0.6 , -1.5)
	Not Send	(0 , 0)	(0 , 0)
	***pr:** 0.5 **S:** 0.5 **RTT:** 0.1 **A:**4 **Q:**0 **Q_2:**0*		

Now, let us imagine the same environment, but we would like to send very prioritized packet such as $Pr = 0.9$. The corresponding payoff table is shown in Table 3. Sending the packet will reward both side 3 and 3,1 respectively with Send, Accept action which has greater payoffs than the previous one. Another identical environment but with highly occupied destination (Q = 0.7), with some timeout and dupack scenarios, so that Q_2 is 0.8. Table 4 shows the corresponding game. Here, there is a considerable occupancy at the destination, but also there is a high prioritized packet. So, still sending it will be the best selection although the rewards are lower than the previous cases.

Table 3. Payoff Table for given environment for high prioritized packet

Player 1 \ Player 2	Accept	Drop
Send	(3 , 3.1)	(-0.6 , -3.1)
Not Send	(0 , 0)	(0 , 0)
***pr:** 0.9 **S:** 0.5 **RTT:** 0.1 **A:**4 **Q:**0 **Q_2:**0*		

Table 4. Payoff Table for given environment for high prioritized packet in a highly occupied destination with some network issues at sender

Player 1 \ Player 2	Accept	Drop
Send	(0.12 , 0.3)	(-0.6 , -0.3)
Not Send	(0 , 0)	(0 , 0)
***pr:** 0.9 **S:** 0.5 **RTT:** 0.1 **A:**4 **Q:**0.7 **Q_2:**0.8*		

Now, let us consider an environment where the sender realizes more timeouts and dupacks compared to the previous cases, and the Q_2 value is 0.9 at sender side. The payoff values are now shown in Table 5.

Table 5. Payoff Table for given environment for high prioritized packet in a highly occupied destination with more network issues at sender

Player 1 \ Player 2	Accept	Drop
Send	(-0.24 , 0.3)	(-0.6 , -0.3)
Not Send	(0 , 0)	(0 , 0)
***pr:** 0.9 **S:** 0.5 **RTT:** 0.1 **A:**4 **Q:**0.7 **Q_2:**0.9*		

Here when we consider destination, it seems to accept the packet, if the packet can make through its way to destination. However, sending the packet will give a negative payoff to the sender. So, here not sending the packet would be the best choice as there is no loss in this action for sensor. This scenario can be commented as there is some network issues and the destination is highly occupied, so no need to take risks on the network for this packet and it might be better to relax destination side by not sending this packet. The flow after "not send" action will be revealed in the next section. Now, we consider another scenario where a medium-low priority packet is sent to a medium-low occupied destination and there exists some network issues. In the resulting payoff table shown in Table 6, it is clearly seen that Player 2 will drop the packet if player 1 sends. So here, it is not possible to collect 0.36 payoff in the action Send, Accept. If player 1 sends the packet, player 2 drops it and player 1 gets –0.6. So that, not sending the packet will make it profit from the loss with zero-grant, in other words, zero-loss.

Table 6. Payoff Table for given environment for mid-low prioritized packet in a mid-low occupied destination with medium packet in a high-speed link network with some network issues.

Player 1 \ Player 2	Accept	Drop
Send	(0.36 , -0.1)	(-0.6 , 0.1)
Not Send	(0 , 0)	(0 , 0)
***pr:** 0.4 **S:** 0.5 **RTT:** 0.1 **A:**4 **Q:**0.3 **Q_2:**0.4*		

However, this does not mean that this packet will be vanished and never be sent to destination again. Not sending a packet which is going to be dropped would result in a high utilization and effort. Waiting for a better time to send it will be a better and efficient choice. If award is increased, let us say $A = 8$, then both sides will have more dare to fulfill their mission. Source will force itself to send the message to destination no matter what and the destination will force itself to accept the packet unless there are extraordinary situations. This scenario can be observed in Table 7.

Table 7. When award increased, both sides observe that they resist to extraordinary situations.

Player 1 \ Player 2	Accept	Drop
Send	(1.32 , 0.3)	(-0.6 , -0.3)
Not Send	(0 , 0)	(0 , 0)
***pr:** 0.4 **S:** 0.5 **RTT:** 0.1 **A:**8 **Q:**0.3 **Q_2:**0.4*		

2.2.4. Deep Diving

So far, we explained the proposed method in general terms and try to give a better understanding on what we are aiming to do, briefly. Now we will deep dive into the proposed algorithm to describe the details of the variables used in the game model, and how to determine them. First struggle is how to determine the Q_2 which is based on Q and increased with some dupack and timeouts. The value of increment is called Q_a, hence $Q_2 = Q + Q_a$. The value of Q_a is proportional to some score α which is between 0 and 100. Actually, α is sum of two non-negative scores, one is α_t which is related to timeout events, and the other is α_d which is related to dupacks. Hence $\alpha = \alpha_t + \alpha_d$. If there is no congestion and all the ACKs are received in time, then $\alpha = 0$ and there is 100 score to fill up. When a timeout occurs for a sent packet, α_t would be incremented by 2^n where n is the number of consecutive timeouts. However maximum value of α cannot exceed 100. When an ACK of a packet is received successfully after a timeout event, α_t is reset to 0 (but α_d stays as is). Moreover, any dupacks will add 1 to α_d and normal ACKs will subtract 1 from the score granted from dupacks. α_t and α_d are separate scores that do not affect each other, however their sum, α, cannot exceed 100. Then, Q_a is determined based on α. In our model, $Q_a = \alpha/100 \cdot Q/2$, hence the maximum value of Q_a cannot

exceed $Q/2$. To be more specific what we are trying to do that here; we actually do not want this part to strongly affect Q_2 as Q itself is already robust and decision-maker. That's why we wanted its effect as half of what Q is. This is still open for discussion and will be fine-tuned in the future works. For example, if Q is 0.3, then Q_2 can be as much as 0.45. If $\alpha = 50$, then $Q_2 = 0.3 + 0.075$.

Another struggle is to determine packet priority. This can be set at application layer where we will determine this at the transport layer. A packet priority can be overridden by any higher layers at sensor side. But this priority should be used cautiously. Setting all the priorities as the highest will incur the general system to determine which packet is important or not. Besides of overriding the priorities, as default approach the payload is read within a TCP envelope. Sensors usually create same values within a specific time interval. Thus, if a sensor value is the same with the previous one, this will decrease the importance of packet. Any identical value will decrease 0.1 score over 1 as priority. Any different value will reset this priority to important state with score 1. Now, "not send" actions require special care where we will touch in detailed here. Let us think from the aspect of MQTT level. There are lots of values that are queued to be sent. MQTT can use priority in its queue itself. When a value desired to be sent, a TCP packet shows up to wrap this value in transport layer and transfers to network layer. In this layer, it can also be accessed to payload of TCP packet. The game theoretical decision is implemented here, if it happens to decide not to send the packet, a waiting period for this packet appears. This waiting period depends on the absolute value of the greatest negative payoff of source or destination in Send action. For example, in Table 6, it is 0.1 which comes from Send, Accept action. In Table 5, it is 0.24. This represents the value how easily source picked this action if it decided not to send due to Q_2, or how easily destination picked this action if it decided to drop due to Q. This extracted value is multiplied by Award A to find how much to wait in terms of seconds, such as $4 * 0.24 = 0.96$ sec. After this waiting period, transport layer will be notified as the packet refused. So that, TCP would learn that specific packet was unable to be sent. As it has retransmission policy this packet will be resent, or it will be discarded if maximum retransmission count has been reached. If the packet is resent, then its priority will be increased by 0.1 and decisions would be re-evaluated using game theory. An overview of the algorithm is revealed in the pseudocode given in Figure 5.

The last, but not the least variable is award A. This award can be set static which should not be a problem and it is 3 as default. But customizing this award according to senders will provide many advantages. Awards boosts the source and destination to send and accept the packet. If award is high enough, and if a packet is decided to be sent, the more payoffs will be granted for sender. Award can be determined by receiver and sent to the sender with an ACK packet using "option" header in TCP. Receiver such as gateway can store the senders and can find their value time intervals or their importance of values by reading the payload. So, frequent message senders can be awarded less than rare message senders. Assume that a temperature sensor sends value within 5 second cycles, but an anomaly detector camera sends value only when an intrusion or suspected actions happens which rarely occurs only per 6 hours averagely. So, gateway can give high award to camera, on the contrary low award to temperature sensor which will throttle it. Moreover, sensors can override award value on their own and share it with destination via handshake process if it is a reliable device. But it should be overridden with caution, otherwise that device might turn into a zombie exploiting the network and destination.

```
Algorithm 1: AQMoT pipeline pseudocode
Result: Decide to send or not send the packet
initialization;
while A packet is not sent or not discarded do
    if TCP Retranmission and retransmission max try count reached then
        Packet discarded;
        If needed Application Layer will try to send it from the stratch;
    else
        Construct the pay-off table of the game and decide;
        if Nash equilibria action is [Send-Accept] then
            Send the packet to destination;
            Accept the packet at destination and acknowledge the source;
            if Acknowledged then
                Reset timeout score from Q_a;
                Decrease Q_a by 1 for dupack score;
                Go for next packet;
            else
                if Not acknowledged then
                    Timeout occurs at TCP side, resend packet with same conditions;
                    Q_a will increase at source side by 2^n ;
                else
                    if Duplicate Acknowledge then
                        Q_a will increase by 1
                    else
                    end
                end
            end
        else
            if Nash Equilibra action is [Not Send] then
                Do not send the packet;
                Wait seconds corresponding to absolute value of the greatest negative
                 payoff of source or destination in Send action;
                Increase the priority by 0.1;
                Return refuse to transport layer;
            else
                Packet Discarded;
            end
        end
    end
end
```

Figure 5. Pseudocode of the proposed approach.

3. Comparison and Discussion

3.1. Comparison

AQMoT is an IoT specific scalable AQM implementation algorithm to cope with queue and congestion effectively. Out of scope of IoT, it might not be appropriate. But, in a contemporary world circumstances IoT is a must and widely spread. So, this algorithm may have a remarkable acceptance comparing other IoT-domained AQM implementations.

AQMoT has also some limitations. Firstly, it may not work as proposed above in an environment with end-to-end encryption. If security is applied at the application layer, such as MQTTS (MQTT secure), the content will be encrypted which disallow us to read the content, and leads us to a problem of determination of priority. In this case, priorities should be defined at application layer and should be passed to transport layer and network layer in the end. Otherwise, the priority determination approach can be renewed such that all the packet priorities are defined as 0.5 for a

specific packet, and the priority is incremented by 0.1 if it is not sent and dedicated to re-send process. Packets can be distinguished by their sequence numbers reside in TCP headers. The second limitation of AQMoT is that it requires space in TCP headers to share some information between sender and destination (such as Queue information, Award amount, etc.). Moreover, some packet processing and memory issues are added such as reading and comparing the contents and the priorities. However, these requirements are minimal and it would not be a big deal for the nowadays devices.

Now, we will provide a conceptual comparison of AQMoT and alternative queue management methods that can be applied in IoT environment. The first method to be considered is ECN although it may not be counted as an active queue management approach. In ECN method, a node can notify another node if it arises a congestion or not by using IP headers. In this implementation, An ECN-enabled TCP receiver node will return with a congestion signal to this notification signal by setting a flag in its next TCP Acknowledgment. ECN is also able to reduce packet losses thanks to proactive congestion control; this leads to an energy and bandwidth save which is crucial for IoT network called Constrained Node Networks. This method can contribute any AQM algorithm to satisfy comfort at the network buffer. But this implementation is too general and it has no specific features for IoT environment. There are plenty of types of nodes within an IoT network where generalization might not work very well. All the nodes require special assistance and individual interest. Some resource-exploiting nodes can cause a general outage as there is no specific throttling mechanism as we proposed.

RED is a generic AQM approach to handle queues in any domain. Although it has an ease of deployment, it is not adaptive to different types of network domains and it might not fit well to IoT domain. However, this approach has inspired researchers to develop many novel approaches. One of the mentioned approaches based on RED is WRED that has different calculation way of average queue and drop function for different traffic classes. WRED, is likely better at applying QoS-Sensitive congestion avoidance. It has reasonable approach to drop and mark packets compared to RED but it is still not fully suitable to IoT network. Because it does not value the content and the sender. It fully focuses on the destination.

Another mentioned variance of RED is ARED. In this approach, the queue average is observed and decided to be more or less aggressive while marking and dropping the packets even though a queue is more or less busy. It adapts itself in accordance to current status. Similar to RED, ARED has two thresholds min_{th} and max_{th} where queue average may oscillate around these limits so that it can define its behavior such as being aggressive or conservative. It also adapts drop probability in these times to maintain more adaptive approach for the variable network conditions. This is also a desired thing in IoT domain. ARED shows up with the best performance in a highly congested environment, however it diminishes its performance when it encounters with an environment with multiple RTT flows. It is also good at fine-tuning the trade-off between goodput and delay performance (Grazia et al. 2017). However still as it does not value the content and network exploiting nodes, and since sender nodes do not cooperate as they do in AQMoT approach, it may not be fully “adaptive” to IoT environment.

XRED which values the content is a good approach. Because by valuing the content, all arriving packets are assigned to some priorities and these priorities are used in the packet drop decision. This aims to maintain the network by handling important packets and getting rid of useless packets. This work has a commonality with our AQMoT approach which cares about content as well. In IoT networks, significance of the content is very important and for fair and effective queue management, it should be considered. However, this approach has the same disadvantages of ours in aspect of content reading. In addition, still there is no cooperation between sender and destination, many packets can fade away due to drop possibilities causing packet losses.

GREEN uses some network conditions such as *MSS*, *RTT*, outgoing link capacity, and number of active flows. Using such kind of terms to define the drop possibility makes it more reasonable. Moreover, this algorithm proposes a good fairness index between flows (Feng et al. 2002). According to (Grazia et al. 2017), PIE, which also overcomes GREEN, has a resistance to all the stress factors, and it has one of the best trade-offs between channel exploitation and delay. Coping with channel exploitation is actually a big part of coping with the whole IoT network. It has also latency-based drop possibility calculation which is not fully random and can be considered as smart dropping. This approach also has a goal to guarantee high-link utilization. However, still it has no cooperation between sender and destination and content awareness which may cause some overhead in IoT network. This still does not avoid sending packets which possibly will be dropped by destination or within network which may cause a futile effort in such an IoT dumb data traffic.

CHOKe and its variants provide fairness by penalizing unresponsive traffic. Since these approaches are stateless, they are simple to implement and require less resources. AQMoT differs from these studies by adding a little intelligence to sensor nodes to avoid them sending data aggressively, instead of penalizing them after the congestion occurs.

3.2. Discussion

In Table 8 algorithms are compared in terms of many aspects. "Content Awareness" points to intelligent actions taken according to the packet content, which is a feature satisfied by XRED and AQMoT. Both algorithms run some logic according to content located in the packet. Others operate regardless of what the content is. "Ease of deployment" mentions about ease of implementation and deployment to the plenty of network nodes. AQMoT has medium level in this category, because some nodes sometimes need to override some priority values which requires extra development at the nodes. Moreover, it requires an update in the devices in Operating System level. Well-known and base algorithms like ECN, RED has a high level of ease of deployment as they are general approaches in this topic. The more complicated algorithms they run, the harder to deploy them. Adaptive approaches like ARED has some struggles to deploy as the logics and algorithms they run may not be compatible for some kinds of nodes. "Fully IoT compatible" means whether an algorithm is developed specifically, efficiently, optimized, and compatibly for many IoT environments. Here, AQMoT is specifically designed for IoT so that it has the highest score in this scope. "Other domains" points to the capability of the algorithm to be efficiently and compatibly used in other domains. Many of the algorithms can be used in different domains other than IoT, but AQMoT can cause overhead. For example, if it is used in streaming services, the system suffers from content reading and tracking. General approaches like RED and ECN has a huge implementation area within many kinds of domains. "Complexity" points to algorithm

Table 8. Overall comparison of algorithms

	Content Awareness	Coop	Ease of Deployment	Fully IoT Compatible	Other Domains	Complexity	E2E Semantics	Intelligence
ECN	No	Sometimes	Good	No	Yes	Low	High	No
RED	No	No	Good	No	Yes	Low	High	No
WRED	No	No	Good	No	Yes	Mid-low	High	Yes
ARED	No	No	Medium	Medium	Nearly	Medium	High	Yes
GREEN	No	No	Medium	Mid-Low	Yes	Medium	High	Yes
XRED	Yes	No	Medium	Low	Nearly	Medium	Medium	Yes
PIE	No	No	Good	High	Nearly	Medium	High	Yes
AQMoT	Yes	Yes	Medium	Yes	Some	Medium	Medium	Yes

complexity. RED is the simplest and the ancestor for many of them. The others actually extended the implementation to do some specific things, such as drop function calculation etc. which adds extra complexity to algorithms. Although AQMoT has a decision-making algorithm within its nature, it is as simple as possible. So the complexity should not be assumed more than a normal level. "E2E semantics (End to End semantics)" points to compatibility throughout all the nodes within network regardless of whether it uses secure channel or different implementations in mid-nodes. AQMoT and XRED can be affected if a secure channel preferred which blocks both algorithms to read the content as it is encrypted. "Coop" means cooperation between source (sender) and destination where they both value current conditions of each other. Besides AQMoT, ECN has also cooperative approach where it signals the sender if it feels itself as congested and sender behaves according to this signal. Destination is clear about its condition to sender in order to push it to coordinate. Other algorithms do not have such kind of approach, they do not care about the opposite side. "Intelligence" means that if algorithm runs some intelligent operation to save itself from randomness, adapts itself to some conditions. For example, PIE uses different approach at calculation drop function by using queueing delay and its tendency to more or less. XRED also uses content tracking to determine priority which is a simple intelligence. ECN and RED algorithms are simple and constant algorithms which do not change in any condition. On the other hand, AQMoT has different decisions and results with respect to many kinds of conditions as described before.

4. Conclusions

In this study, we propose a novel AQM algorithm which specifically, efficiently and compatibly works in IoT domain. The main aim is to avoid futile efforts of dumb IoT nodes, by throttling them if they are exploiting too much resources by sending a lot of low-priority data. For this purpose, we describe a novel light-weight decision-making process based on game theory, where senders (sensors) get feedback from gateways, and make sending decisions accordingly. In this process, senders and receivers are also aware of priorities and contents of the packets. Thus, the proposed approach has a potential of providing better quality of service and fairness. To the best of our knowledge, this is the first AQM approach based on game theory, which is specifically designed for IoT networks. Although the proposed algorithm is candidate to good results, it is still open to further development. There are several operations that should be adjusted to find the best practice, such as determining the Q values, the variables used in game model and the occupancy function. Occupancy function can be exponential or adaptive rather than linear which will be further examined as a future work. Moreover, a detailed experimental study is planned in an active IoT environment.

5. Acknowledge

This work is supported by Koc Digital R&D center.

References

Ataç, C., and Akleylek, S. (2019). A survey on security threats and solutions in the age of IoT. *Avrupa Bilim ve Teknoloji Dergisi*, No. 15, (pp. 36–42).

Naik, N. (2017). Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. *2017 IEEE international systems engineering symposium (ISSE), IEEE, (pp. 1–7).*

Baker, F., and Fairhurst, G (2015). IETF Recommendations Regarding Active Queue Management. RFC 7567. https://doi.org/10.17487/RFC7567, URL https://rfc-editor.org/rfc/rfc7567.txt.

Toğay, C., Mutlu, G., Kurtuluş, D., and Özgür, F.(2019). Secure Gateway for the Internet of Things. *Avrupa Bilim ve Teknoloji Dergisi*, No. 16, (pp. 414–426).

Gomez C., J. C. (2019). TCP Usage Guidance in the Internet of Things (IoT). IETF.

Floyd, S., and Jacobson, V. (1993). Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on networking*, Vol. 1, No. 4, (pp. 397–413).

Freed, M., and Amara, S. K. (2006). Policy-based weighted random early detection method for avoiding congestion in internet traffic. US Patent 6,996,062.

Feng, W.-C., Kandlur, D. D., Saha, D., and Shin, K. G. (1999) .A self-configuring RED gateway. *IEEE INFOCOM'99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No. 99CH36320), Vol. 3, IEEE, (pp. 1320–1328).*

Hassan, M., and Jain, R. (2003). High performance TCP/IP networking, Vol. 29, Prentice Hall Upper Saddle River, NJ

Feng, W.-c., Kapadia, A., and Thulasidasan, S. (2002) GREEN: proactive queue management over a best-effort network. *Global Telecommunications Conference, 2002. GLOBECOM'02. IEEE, Vol. 2, IEEE, 2002, (pp. 1774–1778).*

Pan, R., Natarajan, P., Piglione, C., Prabhu, M. S., Subramanian, V., Baker, F., and VerSteeg, B. (2013). PIE: A lightweight control scheme to address the bufferbloat problem. *2013 IEEE 14th International Conference on High Performance Switching and Routing (HPSR)*, IEEE, (pp. 148–155).

Grazia, C. A., Patriciello, N., Klapez, M., and Casoni, M. (2017). Which AQM fits IoT better?. *2017 IEEE 3rd International Forum on Research and Technologies for Society and Industry (RTSI)*, IEEE, (pp. 1–6).

Aumann, R. J. (1985) .What is game theory trying to accomplish?. Frontiers of Economics, edited by K. Arrow and S. Honkapohja.

Kuhn, H. (2016). Extensive games and the problem op information. *In H. Kuhn and A. Tucker, editors, Contributions to the Theory of Games,* (pp. 193–216).

Hu, S., Sun, J., Xu, Q., & Kong, J. (2020). A Fairness-driven Active Queue Management Algorithm with Hash Table and Circular Buffer. In *2020 Chinese Control And Decision Conference (CCDC)* (pp. 2502-2506). IEEE.

Verma, L. P., & Kumar, M. (2020). An IoT based Congestion Control Algorithm. *Internet of Things*, *9*, 100157

Pan, R., Prabhakar, B., & Psounis, K. (2000, March). CHOKe-a stateless active queue management scheme for approximating fair bandwidth allocation. In *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No. 00CH37064)* (Vol. 2, pp. 942-951). IEEE.

Eshete, A. T., & Jiang, Y. (2013). On the transient behavior of CHOKe. *IEEE/ACM Transactions on Networking*, *22*(3), 875-888.

Lu, L., Du, H., & Liu, R. P. (2013). CHOKeR: A novel AQM algorithm with proportional bandwidth allocation and TCP protection. *IEEE Transactions on Industrial Informatics*, *10*(1), 637-644.

Abbas, G., Manzoor, S., & Hussain, M. (2018). A stateless fairness-driven active queue management scheme for efficient and fair bandwidth allocation in congested Internet routers. *Telecommunication Systems*, *67*(1), 3-20.

Hu, S., Sun, J., Liu, Z., & Xu, Q. (2018). A PI Queueing Delay Controller Enhanced by Adaptive CHOKe for AQM. *IEEE Access*, *6*, 57219-57229.