

# Evaluation of Age Control Protocol (ACP) and ACP+ on ESP32

Umut Guloglu\*, Sajjad Baghaee†, Elif Uysal‡

\* † ‡ Department of Electrical and Electronics Engineering, Middle East Technical University, Ankara, Turkey

\*umut.guloglu@metu.edu.tr, †sajjad@baghaee.com, ‡uelif@metu.edu.tr

**Abstract**—Age Control Protocol (ACP) and its enhanced version, ACP+, are recently proposed transport layer protocols to control Age of Information of data flows. This study presents an experimental evaluation of ACP and ACP+ on the ESP32 microcontroller, a currently popular IoT device. We identify several issues related to the implementation of these protocols on this platform and in general on short-haul, low-delay connections. We propose solutions to overcome these issues in the form of simple modifications to ACP+, and compare the performance of the resulting modified ACP+ with that of the original protocols on a small-delay local wireless IoT connection.

**Keywords**—Age of Information (AoI), Experimental AoI, ACP, ACP+, Internet of Things, IoT, ESP32

## I. INTRODUCTION

In the IEEE special report on Internet of Things issued in March 2014, the term of Internet of Things (IoT) was described as: “A network of items — each embedded with sensors — which are connected to the Internet.” IoT has applications in numerous domains such as intelligent infrastructures, smart healthcare, smart agriculture, and many more.

Remote monitoring, control and automation applications of IoT in areas including, but not limited to, the above domains, have gained importance in recent years. The freshness of information updates, such as those involving sensor measurements, is often important to decision-making in computing, accessing, and storing information. For instance, in tracking an object, the latest true and fresh location information of the object enables remote monitoring to estimate the last position accurately [1].

Age of Information (AoI) [2] is a metric characterizing the freshness of an information flow. It is defined as the time that has elapsed since the newest data belonging to this flow, currently available at the destination, was generated at the source. As it is directly related to the freshness of status update based flows, age is gaining popularity as a new Key Performance Indicator (KPI) for machine-type communication systems such as IoT and other status update type applications. The analysis of AoI in queuing models [3], [4] revealed that this metric behaves quite differently from delay. For example, under First-Come-First-Served (FCFS) service, age exhibits non-monotone behaviour with load, i.e., as the load (arrival rate) is increased, age first decreases and then increases. This naturally calls for an optimization, where the packet generation rate can be optimized to keep the load at a point that minimizes age, and more importantly, service policies that favor newer packets. When possible, though, more freedom to this optimization is provided by controlling the packet generation process to respond to the flow’s instantaneous age.

Age-optimal generation of update packets for a single-hop network was investigated in [5], [6], while [7] characterized AoI in multiple server systems. A majority of the aforementioned studies focused on time average age. Often, in IoT systems, peak values of age, or age at certain decision instants may be more meaningful as a determinant of performance. To quantify the freshness decision epochs when the freshness of updates is only important at some decision epochs, age upon decisions (AuD) was introduced in [8]. While the theoretical models developed in the literature capture an abstraction of the transport layer to a certain degree, such models and the resulting optimizations require knowledge of network delay statistics, which are not often available in practice.

Recently, a number of efforts have considered AoI in real-life networks [9]. An open-source network emulator was used to investigate the AoI in wireless access in [10]. A similar study in [11] combined emulation and real-world AoI measurement experiments, reporting AoI measurements for an end-to-end data flows traversing wireless/wired links, affected by various medium access, transport, and network layer scenarios. Effect of synchronization error on AoI measurement and a solution for calculating an estimate of average AoI without any synchronization requirements has been shown in [12]. Authors in [13] utilized several transport protocols such as TCP, UDP, and WebSocket to measure AoI on wired and wireless links. Their study includes practical issues such as synchronization and selection of hardware along with transport protocol and their effects on AoI measurements.

Efforts to carry the intuition from theory to practice has included dynamically adaptive approaches such as those that employ Machine Learning methods to age optimization in a network [14]. Modifications have been proposed in the transport and application layers to control congestion with an age objective, to keep the “pipe just full, but not fuller” as suggested in [15]. Recent protocols in that vein include [16]–[18]. Shreedhar et al. [18] have introduced the Age Control Protocol (ACP) for multi-hop IP networks, enabling timely delivery of updates by adapting the sending rate. They also compared the performance of ACP with the Lazy Policy in their study. The aim of the Lazy policy is to transmit data with an interval around Round Trip Time (RTT). An improved version of ACP, named ACP+, was presented in [19]. According to [19], ACP+ has superior performance for timely delivery of updates over fat pipes and long paths.

The goal of this paper is to report experimental test results of ACP and ACP+ over shorter paths, and low rates, in an IoT

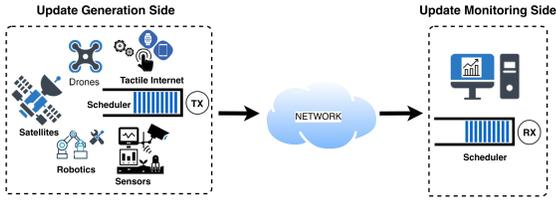


Fig. 1: A status update communication setup.

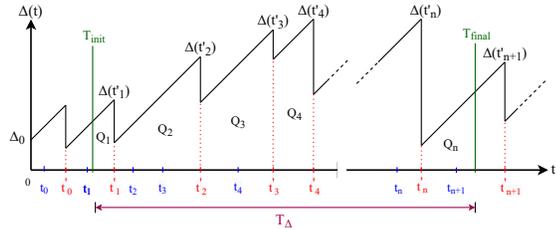


Fig. 2: Sample path of the age process:  $\Delta(t)$

scenario, and to suggest how the protocol may be modified for this scenario. The outline of the rest of the paper is as follows. Section II provides the definition of age and an expression for the computation of its time average on a given packet trace. Descriptions of ACP and ACP+ and their distinctions are presented in Section III. Section IV describes the testbed used in our study. Section V provides a discussion related to the issues and solutions of using ACP and ACP+ on the ESP32 microcontroller. Results and conclusions are presented in Section VI and VII, respectively.

## II. AOI: DEFINITION AND COMPUTATION

Consider a status update scenario (Fig.1), where a process running at a destination node needs samples (status updates) from a remote source. After the generation of an update at timestamp  $U(t)$  at the source, that update will start aging. When the receiver is not fed with fresh updates, the process at the receiver may suffer due to stale data. Once a fresh update is received, the age on the monitoring side is updated with the age of the recently arrived update. The status age,  $\Delta(t)$ , is calculated as  $\Delta(t) = t - U(t)$ , where  $t$  is the time of the newest update available at the monitoring side.

Based on this definition, age is a continuous-time continuous-valued process with sample paths that follow a sawtooth pattern (Fig. 2). Without loss of generality, let us assumed that the observation begins at  $t = 0$  and update generation starts at  $t > 0$  with empty transmitter and receiver queues. The initial age is  $\Delta(0) = \Delta_0 \geq 0$ . The source generates status updates at  $t_0, t_1, \dots, t_n$ , which reach the monitor at  $t'_0, t'_1, \dots, t'_n$ .

The area under the age graph within a time window of length  $T_\Delta = T_{final} - T_{init}$ , (Fig. 2) normalized by  $T_\Delta$  provides the time average age within this window:

$$\bar{\Delta} = \frac{1}{T_\Delta} \int_{T_{init}}^{T_{final}} \Delta(t) dt = \frac{\sum_{i=1}^n Q_i}{T_\Delta} \quad (1)$$

In practice, consider a packet trace including  $n$  updates and age graph as shown in Fig. 2. The area under this graph is made up of  $n$  trapezoidal areas, where  $Q_1, Q_i$  for  $2 \leq i \leq n-1$  and  $Q_n$ , are calculated as (2),(3) and (4).

$$Q_1 = \frac{1}{2}(T_{init} + t'_1 - 2t_0)(t'_1 - T_{init}) \quad (2)$$

TABLE I: Decision algorithm of ACP

$\delta_k$	$b_k$	ACP targets to:
$> 0$	$> 0$	Decrease the backlog
$> 0$	$< 0$	Increase the backlog
$< 0$	$> 0$	Increase the backlog
$< 0$	$< 0$	Decrease the backlog

$$Q_i = \frac{1}{2}(t'_i + t'_{i-1} - 2t_{i-1})(t'_i - t'_{i-1}) \quad (3)$$

$$Q_n = \frac{1}{2}(T_{final} + t'_n - 2t_n)(T_{final} - t'_n) \quad (4)$$

## III. AGE CONTROL PROTOCOL (ACP) AND ACP+

ACP and ACP+ are transport layer protocols designed to minimize the age of information at the monitor (receiver) by altering the updating (sending) rate of the source (sender) according to the estimated network conditions. In this section, we describe ACP and the differences between ACP and ACP+.

ACP is built on the User Datagram Protocol (UDP), and operates on each end host of a session and consists of two phases: the initialization phase, and the ‘‘epochs’’ phase. In the initialization phase, the source sends a certain number of packets to the monitor, waits for the ACK packets, and computes a Round-Trip Time (RTT) estimate based on these. The purpose of this phase is to configure the initial update rate of the source. This phase is followed by the epochs phase, divided into periods called epochs. In each epoch, labeled with  $k$ , the average AoI ( $\bar{\Delta}_k$ ) and average backlog ( $\bar{B}_k$ ) (average number of packets sent to the monitor, but not yet acknowledged) are computed. ACP decides to increase or decrease the backlog according to changes in  $\bar{\Delta}_k$  and  $\bar{B}_k$ . The decision algorithm of the protocol is shown in Table I, where  $\delta_k$  is defined as  $\bar{\Delta}_k - \bar{\Delta}_{k-1}$  and  $b_k$  is defined as  $\bar{B}_k - \bar{B}_{k-1}$ .

When an ACK packet arrives at the source, the exponentially weighted moving average (EWMA) of RTT ( $\bar{RTT}$ ), and the EWMA of inter-ACK arrival times ( $\bar{Z}$ ) are calculated. ACP uses these to infer network conditions and set an appropriate update rate ( $\lambda_k$ ), which will be used during the next epoch. By changing  $\lambda_k$ , the protocol can alter the backlog and thereby control the age. For that purpose, ACP chooses an action and aims to change the average backlog by  $b_{k+1}^*$  packets according to changes in  $\bar{B}_k$  and  $\bar{\Delta}_k$ , where  $b_{k+1}^*$  is the desired backlog change. The computation of the update rate is given by (5).

At each epoch, one of the following three actions, described in Table II, are chosen: ‘‘INC’’, ‘‘DEC’’, and ‘‘MDEC’’. Each action results in a certain corresponding  $b_{k+1}^*$ . The value of step-size parameter  $\kappa$ , see Table II, is constant during packet transmission. It is crucial for ACP as the actions ‘‘INC’’ and ‘‘DEC’’ aim to change the average backlog by  $\kappa$  packets. ACP uses the ‘‘MDEC’’ action to obtain a multiplicative decrease when ‘‘DEC’’ does not decrease  $\bar{B}_k$  sufficiently fast.

$$\lambda_k = \frac{1}{\bar{Z}} + \frac{b_{k+1}^*}{\min(\bar{RTT}, \bar{Z})} \quad (5)$$

ACP also assigns the length of each epoch ( $\bar{T}$ ) using (6) at epoch boundaries. The epoch duration must be short enough to respond to the changes promptly and long enough to provide an accurate estimate of network conditions.

TABLE II: Table of actions and corresponding  $b_{k+1}^*$  values

The Action	$b_{k+1}^*$
Increase (INC)	$\kappa$
Decrease (DEC)	$-\kappa$
Multiplicative Decrease (MDEC)	$-(1 - 2^{-\gamma})B_k$

$$\bar{T} = 10 \times \min(\overline{RTT}, \bar{Z}) \quad (6)$$

ACP+, an enhanced Age Control Protocol, is based on ACP, with several modifications:

- **Computation of  $\lambda_k$ :** While ACP uses (5) to update  $\lambda_k$ , ACP+ uses a slightly altered equation where  $\min(\overline{RTT}, \bar{Z})$  is replaced by  $\overline{RTT}$ .
- **Setting  $\kappa = 1$  and clamping  $\lambda_k$ :** The other substantial difference is that  $\kappa$  is set to 1 in ACP+, and  $\lambda_k$  is clamped in order not to alter the update rate dramatically. In [19] authors specified the clamping boundaries as  $1.25 \times \lambda_{k-1}$  (Max update rate) and  $0.75 \times \lambda_{k-1}$  (Min update rate) in which  $\lambda_{k-1}$  stands for the update rate of the previous epoch.
- **Modification in Eq. (6):** ACP+ uses (7) to set  $\bar{T}$ . This is obtained by replacing  $\min(\overline{RTT}, \bar{Z})$  in (6) by  $\frac{1}{\lambda_k}$ . This roughly corresponds to ACP+ targeting to send about 10 packets over an epoch duration.

$$\bar{T} = \frac{10}{\lambda_k} \quad (7)$$

#### IV. THE TESTBED

Our motivation is to determine whether ACP and ACP+, which were successfully tested on long-haul, high rate connections, also successfully minimize the age of information in small-delay real-world IoT networks, and if not, explore whether certain modifications can be proposed to better cater for these settings. For this purpose, we designed a test setup (Fig. 3), which contains two ESP32 low-power systems with on chip microcontrollers and integrated Wi-Fi, and a cellular phone used as a Hotspot. In our setup, one of the ESP32 devices is assigned as a source (TX), whereas the other is appointed as a monitor (RX). The transmitter node sends packets using UDP. By listening to the serial port of TX, data logging in each epoch is done. It follows by visualizing and analyzing the results via MATLAB software.

In this experiment, ACP/ACP+ are implemented in the transmitter node. The transmitter sends packets, and the receiver reflects the incoming packet back to the sender in place of an ACK. The transmitter node calculates all necessary information related to ACP/ACP+ and decides the action which must be taken. The receiver node acts as an echo server in this experiment.

We primarily compared ACP with ACP+ and the Lazy Policy in [18]. The Lazy Policy essentially tries to keep the sending period around RTT, hence the update rate is set to  $\frac{1}{\overline{RTT}}$  at the end of each epoch. This policy tries to keep the backlog around 1 following the guidelines given in [15], and was used as a simple benchmark following [18].

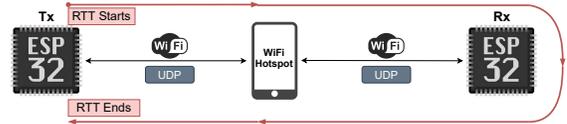


Fig. 3: The schematic of the testbed

A set of 5 different experiments were carried out, running Lazy, ACP+, and ACP side by side. In each experiment, there were 5 runs for each of the 8 different  $\kappa$  values. In each run, the transmitter sent 10,000 packets to the receiver. Each packet had a 4-byte payload.

In the second set of experiments, we focused on improving ACP+ for small-delay networks and testing this modified (improved) version. The original ACP+ clamps  $\lambda_k$  if it is bigger than  $1.25 \times \lambda_{k-1}$  or smaller than  $0.75 \times \lambda_{k-1}$ . During the experiments, we noticed that ACP+ clamps approximately 81% of the designated  $\lambda_k$  values and realized the boundaries of clamping are far from  $\lambda_{k-1}$ , which makes ACP+ a greedy protocol for small-delay networks. Therefore, we modified ACP+ by altering clamping boundaries as  $1.1 \times \lambda_{k-1}$  and  $0.9 \times \lambda_{k-1}$  and ran the test 5 times for the original and modified ACP+ versions with two different epoch length scenarios. These scenarios are detailed in Section V. We also ran ACP with  $\kappa$  value giving the best result in terms of AoI to compare with the new ACP+ version. The estimated average  $\bar{\Delta}_k$  values under the original and modified ACP+ versions are reported in Table III. Lastly, we traced the age over time for ACP+ and Lazy Policy and tested whether the feedback mechanism, see Sec. V, is successful in peak age violation cases.

#### V. DISCUSSIONS

Several issues that we have recognized to possibly affect the success of the protocols under study will be discussed next. This will be followed by suggested solutions. Some of the issues are due to the particular device hardware (which we believe will be typical of other IoT devices), while others are due to the parameter settings of ACP, or ACP+.

##### A. Issues

###### • Issue 1: Unnecessary Queuing Delay in ESP32's

While sending and retrieving the packets, it is noticed that the measured RTT values are larger than the expected ones on occasion. This issue has been encountered in independent work [13], which suggested an underlying buffer management error for its cause. It is observed that ESP32 starts to process the incoming packet after several packets arrive at the buffer.

###### • Issue 2: Error in Average Age Calculation

The average age in each epoch is calculated by dividing the area under the age graph by the duration of the epoch. However, there is a fringe effect due to the time interval between the arriving time of the last received data and the ending, or starting, time of the epochs, i.e.,  $Q_1$  and  $Q_n$  in Fig. 2. This results in a small overestimation of average age, which may incorrectly alter the chosen action of ACP and ACP+ in the case of small epoch durations.

###### • Issue 3: Failure of The System Due to Issue 1

Assume Issue 1 occurs, and packets incur a long delay in the queue. In this case,  $\overline{RTT}$  and  $\bar{Z}$  increase, which

triggers the system to decrease  $\lambda_k$ . Normally, the “INC” action increases  $\bar{B}_k$  by  $\kappa$  packets according to (5), if the calculated  $\overline{RTT}$  and  $\bar{Z}$  are true values. However, in our case, ACP estimates the network conditions like an intercontinental network and sends data with a slow rate. If the problem continues, it always overestimates  $\overline{RTT}$  and  $\bar{Z}$ , and cannot increase the update rate to the optimal levels since it is the right move according to the algorithm. Thus, it enters a loop that is tough to exit.

### B. Proposed Solutions

- **Solution 1: Increasing the Length of the Epochs**

Increasing the length of the epochs is a solution for Issue 1 and Issue 2. In [18], [19], the epoch length is calculated by (6) and (7) for ACP and ACP+, respectively. This can be called  $\bar{T}_{10}$  case. In our design, we defined the length as  $30 \times \min(\overline{RTT}, \bar{Z})$  for ACP and  $\frac{30}{\lambda_k}$  for ACP+ by aiming to increase the length. This case is named  $\bar{T}_{30}$ . Except in the test to compare these two cases, the  $\bar{T}_{30}$  case is used in the rest of this study.

Suppose Issue 1 is occasionally encountered. In that case, an increase in the age and thereby an abnormal increase in the area under the AoI curve is expected. By increasing the epoch length sufficiently, the effect of this temporary increase may be mitigated. Of course, as the length is increased, the system reacts more slowly than the one with a shorter length case to sudden packet traffic variations and RTT changes. We tested the outcomes of increasing the epoch length. According to the results (Section VI), it is deduced that even if this is a considerable issue in dynamic networks, it is beneficial for small-delay networks like the one under consideration.

- **Solution 2: Adding a Feedback Mechanism**

This is a solution for Issue 3, and it can only be used in one-hop networks with one client and one server. It is designed for test purposes. The update rate must be increased to exit from the loop caused by Issue 3. If ACP increases the update rate, the buffer is filled in a shorter time. Thus, the difference between the measured age value, which is not correct due to Issue 1, and the actual age value decreases. For this purpose, we designed a feedback algorithm that will be active when the age exceeds a predefined peak age threshold value. In our experiments, 200 ms is set as the peak age threshold. In this algorithm, the minimum value of RTT of each epoch is calculated. If the peak age is violated, the feedback mechanism is activated, and the new  $\overline{RTT}$  is calculated by (8).

$$\overline{RTT} = \frac{\overline{RTT} + \zeta \times RTT_{min,epoch}}{\zeta + 1} \quad (8)$$

$\zeta$  in (8) is set to 0 initially. After each epoch, the mechanism checks whether there exists a peak age violation or not. If the peak age is violated,  $\zeta$  is increased by 1, and it becomes 0 otherwise. In other words, if the violation occurs consecutively, the aggressiveness of the mechanism increases. As a result,  $\overline{RTT}$  decreases considerably, which leads to an increase in the update rate. The pseudocode of the mechanism is shown in Algorithm 1.

This feedback mechanism aims to exit from the loop and return to the optimum update rate’s vicinity. As explained

---

### Algorithm 1 Feedback Mechanism for Test Measurements

---

**Input:**  $\bar{\Delta}_k, \bar{T}$  and  $\zeta \leftarrow 0$   
**while true do**  
  **if**  $\bar{\Delta}_k > Peak\ Age\ Threshold$  **then**  
     $\zeta = \zeta + 1$   
  **else**  
     $\zeta \leftarrow 0$   
    update  $\overline{RTT}$  and wait  $\bar{T}$

---

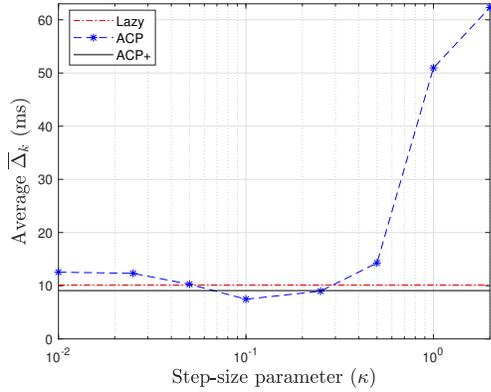
previously, it is designed for peer-to-peer communications over a one-hop system. If there is more than one source or more than one monitor, or another source sending data using the same access point, it is not possible to know whether Issue 1 or the traffic congestion is the reason for the violation.

## VI. RESULTS

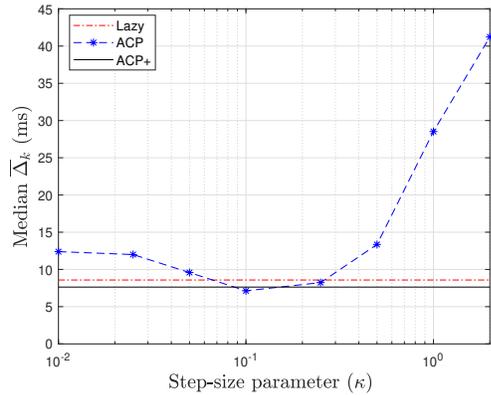
To examine the effect of the step-size parameter ( $\kappa$ ) on AoI in ACP, a series of experiments have been conducted. Fig. 4a and Fig. 4b compare ACP with different  $\kappa$  values, original ACP+ and Lazy Policy in terms of empirical  $\bar{\Delta}_k$  (average age over an epoch). Experimentally, ACP with  $\kappa = 0.1$  is the best from the point of average  $\bar{\Delta}_k$ , see Fig. 4a, and median  $\bar{\Delta}_k$ , see Fig. 4b. Smaller  $\kappa$  values lead to higher age as ACP cannot react in time. On the other hand, larger values of  $\kappa$  also lead to higher age because ACP becomes over-greedy, causing redundant  $\lambda_k$  oscillations. The cumulative distribution function (CDF) of  $\bar{\Delta}_k$  under ACP protocol is depicted in Fig. 5. The reason for the flatter curves in the  $\kappa = \{0.5, 1, 2\}$  cases is the unnecessary oscillations as stated before. As a result, it appears that  $\kappa$  must be chosen with care to obtain optimal results. However, as noted in [19],  $\kappa$  is a parameter difficult to set in general. ACP with the same  $\kappa$  value might work differently in different time intervals and networks. The main strength of ACP+ is the avoidance of the choice of  $\kappa$ .

Fig. 6 shows the estimated  $\bar{\Delta}_k$  values versus time graph for original ACP+, modified ACP+ and Lazy Policy. As seen in the figure, there are several peaks in both versions of ACP+. We notice that only the first peak of the original ACP+, blue line in Fig. 6, occurs due to Issue 1. The second and third peaks exist since  $\lambda_k$  is increased too much (see Fig. 7), i.e., the clamping boundaries are too far. The reason for the peak in the modified one is over-increased  $\lambda_k$  as well, but the age is relatively lower than the original ACP+. Modified ACP+ generally gives better results with low variance compared to the original ACP+ and Lazy Policy. The empirical variances of  $\bar{\Delta}_k$  are calculated as 13.38, 20.93, and 3.24 for Lazy Policy, original ACP+, and modified ACP+, respectively. Top-right plot in Fig. 6 is a magnified part of the trace from  $t = 35.6$  sec to  $t = 40.1$  sec.

Fig. 8a and Fig. 8b demonstrate the CDFs of  $\bar{\Delta}_k$  and  $\overline{RTT}$  for Lazy Policy, ACP with  $\kappa = 0.1$ , and both versions of ACP+, respectively. There is an error margin in  $\overline{RTT}$  because of Issue 1 explained in Sec. V, the temperature of ESP32’s, or other interfering signals. It is observed that the system did not encounter Issue 1 in ACP experiments with  $\kappa = 0.1$ , but this problem is observed in ACP+ cases. Even though these problems may affect the results, it is expected that the differences in measured  $\overline{RTT}$ ’s between the protocols mainly



(a) The empirical average  $\bar{\Delta}_k$  results



(b) The empirical median  $\bar{\Delta}_k$  results

Fig. 4: The empirical  $\bar{\Delta}_k$  results for Lazy Policy, ACP with different  $\kappa$  values and ACP+

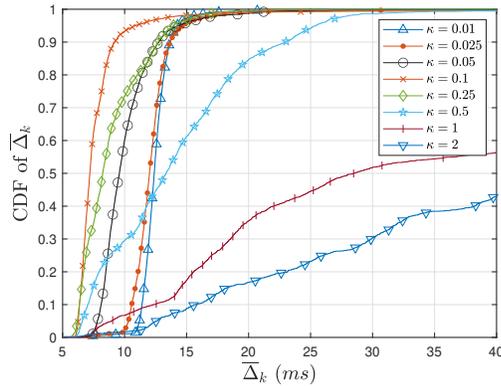


Fig. 5: The empirical AoI CDFs for ACP with different  $\kappa$

occur due to queueing delays in the hotspot and ESP32's. It can clearly be seen that modified ACP+ gives the best results in terms of AoI, whereas ACP gives the best results in terms of  $\overline{RTT}$ . One can deduce that modified ACP+ succeeded in minimizing AoI even if  $\overline{RTT}$  is higher than ACP.

We have conducted another test to measure the success of the designed feedback. Fig. 9 shows the trace of AoI and RTT values of ACP with  $\kappa = 0.5$ . We set the peak age threshold as 200 ms. When  $t = t_v$ ,  $\overline{RTT}$  is equal to 201.4 and  $\bar{\Delta}_k$  is equal to 262.6. In other words, there is a peak age violation that triggers the feedback mechanism. After feedback activation,

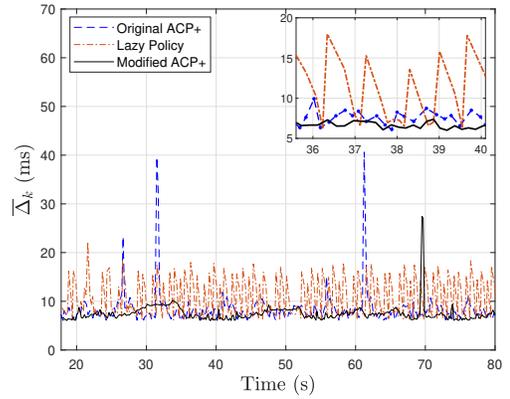


Fig. 6: The trace of Lazy Policy and both versions of ACP+

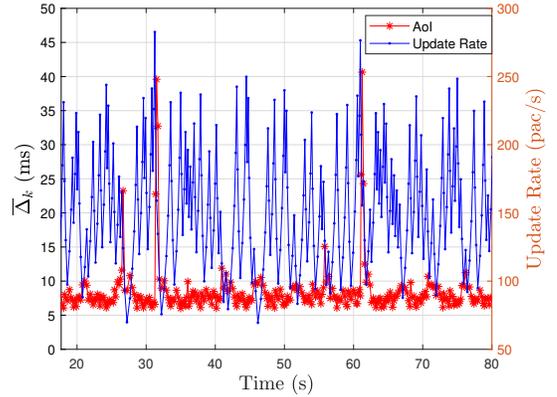


Fig. 7: AoI and update rate variations for original ACP+

the decrease of  $\overline{RTT}$  is observed. It decreases to 165.02 ms and triggers an increase on  $\lambda_k$  and thereby decreasing  $\bar{\Delta}_k$ .

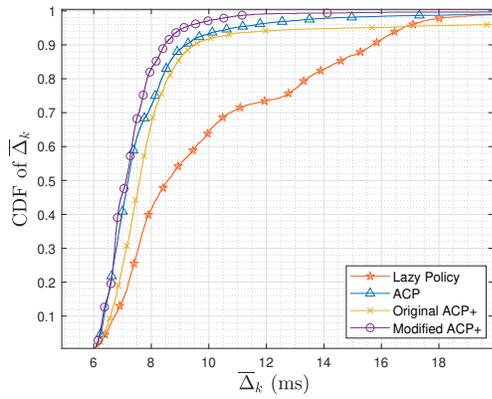
As stated in Solution 1, we multiplied  $\min(\overline{RTT}, \bar{Z})$  by 30 for ACP and  $\frac{1}{\lambda_k}$  by 30 for ACP+ to designate  $\bar{T}$  so far, i.e.,  $\bar{T}_{30}$  case is used. In  $\bar{T}_{30}$  case, we observe that modified ACP+ outperforms the original one in terms of average AoI (see Table 3). Lastly, we examine the effects of  $\bar{T}$  over the age of information. One can see that the average age increases when (7) is used to determine the length, which stands for the  $\bar{T}_{10}$  case for ACP+. The reason is that ACP+ updates  $\lambda_k$  frequently and creates redundant oscillations in the  $\bar{T}_{10}$  case. Even though original ACP+ might have higher resiliency to network condition alterations, modified ACP+ offers prospering results for small-delay networks in both  $\bar{T}_{10}$  and  $\bar{T}_{30}$  cases.

## VII. CONCLUSIONS

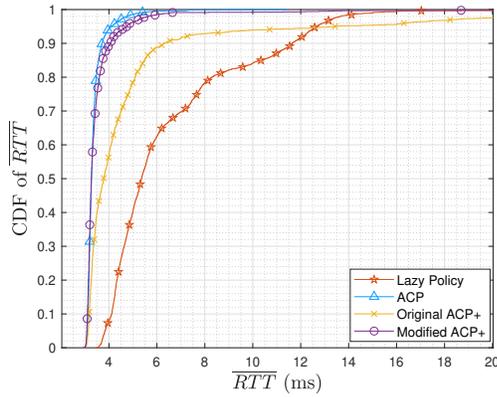
This study evaluated ACP and ACP+ [18] on IoT devices, specifically ESP32's. We have reported several issues related to applying ACP and ACP+ in this setting and suggested solutions to solve these issues. A feedback mechanism was designed to exit from the loop which ACP and ACP+ enter due to ESP32's buffer management error. AoI measurements in real-world networks under ACP for different  $\kappa$  values, the

TABLE III: The estimated average AoI values

	Original ACP+	Modified ACP+
$\bar{T}_{10}$	9.15 ms	8.93 ms
$\bar{T}_{30}$	9.01 ms	7.54 ms



(a) The empirical AoI CDFs



(b) The empirical RTT CDFs

Fig. 8: The empirical CDFs for Lazy Policy, ACP with  $\kappa = 0.1$  and two versions of ACP+

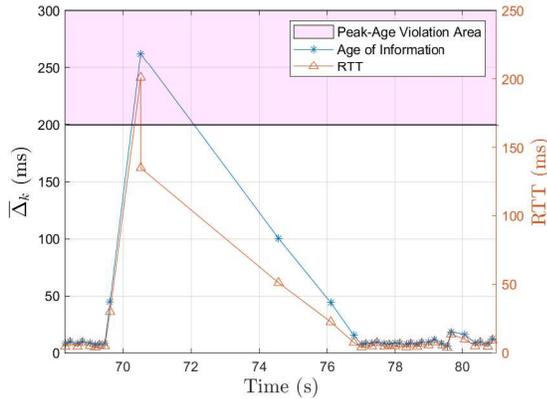


Fig. 9: Feedback Test

Lazy Policy in [18] and ACP+ have been conducted. Results indicate that the original ACP+ is too greedy for small-delay networks. A modified version of ACP+ was designed to solve this problem of the original ACP+, and results show that modified ACP+ gives superior results. As future work, we plan to make clamping boundaries RTT-dependent such that ACP+ become compatible and deployable for every IoT device in every network.

#### VIII. ACKNOWLEDGEMENTS

This study has been supported by TUBITAK grants 117E215 and 119C028. We also would like to thank Tanya Shreedhar for discussions.

#### REFERENCES

- [1] S. Baghaee, S. Z. Gurbuz, and E. Uysal-Biyikoglu, "Implementation of an enhanced target localization and identification algorithm on a magnetic wsn," *IEICE Transactions on Communications*, vol. E98.B, no. 10, pp. 2022–2032, 2015.
- [2] S. Kaul, R. Yates, and M. Gruteser, "Real-time status: How often should one update?" in *2012 Proceedings IEEE INFOCOM*, March 2012, pp. 2731–2735.
- [3] S. K. Kaul, R. D. Yates, and M. Gruteser, "Status updates through queues," in *2012 46th Annual Conference on Information Sciences and Systems (CISS)*, March 2012, pp. 1–6.
- [4] E. Najm and R. Nasser, "Age of information: The gamma awakening," in *2016 IEEE International Symposium on Information Theory (ISIT)*, July 2016, pp. 2574–2578.
- [5] R. D. Yates, "Lazy is timely: Status updates by an energy harvesting source," in *2015 IEEE International Symposium on Information Theory (ISIT)*, June 2015, pp. 3008–3012.
- [6] B. T. Bacinoglu, E. T. Ceran, and E. Uysal-Biyikoglu, "Age of information under energy replenishment constraints," in *2015 Information Theory and Applications Workshop (ITA)*, Feb 2015, pp. 25–31.
- [7] A. M. Bedewy, Y. Sun, and N. B. Shroff, "Optimizing data freshness, throughput, and delay in multi-server information-update systems," in *2016 IEEE International Symposium on Information Theory (ISIT)*, July 2016, pp. 2569–2573.
- [8] Y. Dong, Z. Chen, S. Liu, P. Fan, and K. B. Letaief, "Age-upon-decisions minimizing scheduling in internet of things: To be random or to be deterministic?" *IEEE Internet of Things Journal*, vol. 7, no. 2, pp. 1081–1097, 2020.
- [9] E. Uysal, O. Kaya, S. Baghaee, and H. B. Beytur, "Age of information in practice," *CoRR*, vol. abs/2106.02491, 2021. [Online]. Available: <https://arxiv.org/abs/2106.02491>
- [10] C. Kam, S. Kompella, and A. Ephremides, "Experimental evaluation of the age of information via emulation," in *MILCOM 2015 - 2015 IEEE Military Communications Conference*, Oct 2015, pp. 1070–1075.
- [11] C. Sönmez, S. Baghaee, A. Ergişi, and E. Uysal-Biyikoglu, "Age-of-Information in practice: Status age measured over TCP/IP connections through WiFi, Ethernet and LTE," in *2018 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*, June 2018, pp. 1–5.
- [12] H. B. Beytur, S. Baghaee, and E. Uysal, "Measuring age of information on real-life connections," in *2019 27th Signal Processing and Communications Applications Conference (SIU)*, 2019, pp. 1–4.
- [13] H. B. Beytur, S. Baghaee, and E. Uysal, "Towards aoi-aware smart iot systems," in *2020 International Conference on Computing, Networking and Communications (ICNC)*, 2020, pp. 353–357.
- [14] E. Sert, C. Sönmez, S. Baghaee, and E. Uysal-Biyikoglu, "Optimizing age of information on real-life TCP/IP connections through reinforcement learning," in *2018 26th Signal Processing and Communications Applications Conference (SIU)*, May 2018, pp. 1–4.
- [15] L. Kleinrock, "Internet congestion control using the power metric: Keep the pipe just full, but no fuller," *Ad Hoc Networks*, vol. 80, pp. 142–157, 2018.
- [16] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "Bbr: Congestion-based congestion control," *ACM Queue*, vol. 14, September-October, pp. 20 – 53, 2016. [Online]. Available: <http://queue.acm.org/detail.cfm?id=3022184>
- [17] A. Alós, F. Morán, P. Carballeira, D. Berjón, and N. Garcia, "Congestion control for cloud gaming over udp based on round-trip video latency," *IEEE Access*, vol. 7, pp. 78 882 – 78 897, 06 2019.
- [18] T. Shreedhar, S. K. Kaul, and R. D. Yates, "An age control transport protocol for delivering fresh updates in the internet-of-things," in *2019 IEEE 20th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*, June 2019, pp. 1–7.
- [19] T. Shreedhar, S. K. Kaul, and R. D. Yates, "An empirical study of ageing in the cloud," 2021. [Online]. Available: <https://arxiv.org/abs/2103.07797>