Scheduling at the Edge for Assisting Cloud Real-Time Systems

Lorenzo Corneo
Uppsala University
Sweden
lorenzo.corneo@it.uu.se

Per Gunningberg
Uppsala University
Sweden
per.gunningberg@it.uu.se

ABSTRACT

We study edge server support for multiple periodic real-time applications located in different clouds. The edge communicates both with sensor devices over wireless sensor networks and with applications over Internet type networks. The edge caches sensor data and can respond to multiple applications with different timing requirements to the data. The purpose of caching is to reduce the number of multiple direct accesses to the sensor since sensor communication is very energy expensive. However, the data will then age in the cache and eventually become stale for some application. A push update method and the concept of age of information is used to schedule data updates to the applications. An aging model for periodic updates is derived. We propose that the scheduling should take into account periodic sensor updates, the differences in the periodic application updates, the aging in the cache and communication variance. By numerical analysis we study the number of deadline misses for two different scheduling policies with respect to different periods.

CCS CONCEPTS

• **Computer systems organization → Distributed architectures**; *Cloud computing*; *Sensor networks*;

KEYWORDS

Edge computing; cloud computing; sensor network; Industrial IoT; energy efficiency

ACM Reference Format:
Lorenzo Corneo and Per Gunningberg. 2018. Scheduling at the Edge for Assisting Cloud Real-Time Systems. In *TOPIC '18: Theory and Practice for Integrated Cloud, Fog and Edge Computing Paradigms 2018 Workshop, July 27, 2018, Egham, United Kingdom.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3229774.3229777

1 INTRODUCTION

Cloud computing is increasingly becoming attractive for real-time systems. One attractive advantage is that major part of the control application execution can be off-loaded from the sensing and actuating devices to the cloud at the cost of communication. This off-loading is particularly important for battery operated devices.

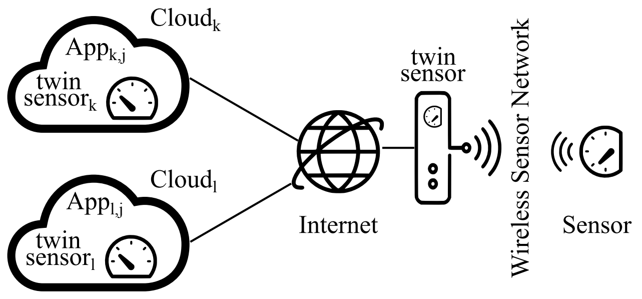


Figure 1: A real-time cloud based system with an edge server supporting an energy constrained sensor. The sensor data is replicated at the "twin sensors".

Although there are many scalability advantages with control logic in the cloud, the approach also has several consequences. First, since cloud servers are often provided at relatively long distances from the controlled system, there will be a significant communication delay between the actual sensing event and the application execution. In addition, a cloud server is designed to support many simultaneous users, which may cause application scheduling delays that may be significant. The data will age during the communication [6, 7], i.e. the information quality will decline. We here use the metric age of information to measure the quality, defined as the age of the data since it was generated. Eventually, the data will be too old and not valid anymore for a correct real-time response, i.e. the data has become stale and it will cause a real-time deadline miss.

Edge and fog computing [2] have been proposed to handle the distance problem of the cloud. The idea is to "delegate" time and safety critical functionality of the control logic to a server close enough to the controlled system, typically located at the network edge.

We consider real-time cloud-edge-sensor architectures composed of sensor devices located on a local wireless network with an Internet gateway, an edge server co-located with the gateway and a cloud infrastructure, hosting several applications that need timely data from the sensors. See figure 1. The edge acts as a smart caching proxy between the sensor devices and the clouds. If the edge caches sensor data, it can respond to multiple application requests at different clouds as long as the data is fresh enough for them.

In particular we consider periodic applications and battery operated sensor devices that are duty cycled in order to save energy. The caching can reduce the number of sensor accesses in order to save energy but at the cost of some aging in the edge that increases the probability of stale data. From energy conserving point of view it is desirable to have as long period as possible between duty cycles while still meeting the real-time requirements from the most stringent update periods of the applications.

The edge completely controls the communication to the sensor and caches the most recent sensor data value in a "digital twin sensor" (i.e. virtual sensor) [1], see figure, which is in turn exported to the corresponding twins at the clouds. The sensor value will age in the edge until it is renewed by a new sensor reading at a duty cycle. The cached value is pushed to the twins according to the periodicity of the applications. A problem with this approach is that the sensor cycle and the applications have different periodicity. This causes varying edge aging of data for the applications compared to when all applications do their own individual reading directly to the sensor. Aging increases the risk for stale data for correct real-time execution compared to direct access to the sensor and must be controlled at the edge. Our contributions are the following:

- Novel insights on how data ages in edge for periodic real-time cloud applications.
- An edge scheduling strategy that moves aging in the edge to aging in the cloud in order to decrease deadline misses.

We believe that we are among the first to study how a smart edge server could support cloud based real-time applications using duty cycled sensor devices.

In the following section we set a scenario and discuss related work. In section 3 our cloud-edge-sensor system is described and modelled with some simplifying assumptions. The model includes communication latencies and aging processes for multiple applications with different update frequencies. Section 4 describes when a scheduler in the edge may push data to the applications with respect to the aging at the cloud, edge and the periodicity of the applications. Then in the following section we evaluate two scheduling principles; a basic one determined by the periodicity of the applications and a smarter one that also take into the account the skew between the application periods. In section 5 we evaluate the two principles with numerical analysis with respect to how long time the data in the application is valid given the age of information and real-time constraints of the applications. Our purpose of the evaluation is to show the possible gain with a smart scheduling. The paper is finalized with a conclusion section.

2 SCENARIO AND RELATED WORK

As an illustrating scenario of a real-time cloud based system, consider autonomous forklift trucks in a factory, moving goods on the factory floor. To justify the need for delegating time critical functions to the edge, assume humans moving more or less randomly on the factory floor which requires the trucks to reliably and swiftly avoid hitting them. The anti-collision function is preferably located in the truck or a close-by server. On the other hand, a logistic program on where to pick up goods and to calculate the best route for the trucks is preferably done in the cloud since it needs all the truck positions and data about the location and destination of the goods. Still, the varying communication delays may cause stale data at the cloud.

Fog computing [2], or also called edge computing, is a paradigm that has been proposed to supply to the limitations of cloud computing. For example, fog computing can be used to enable highly responsive (e.g., low latency) cloud services, mask cloud outrages [9] and enforce privacy [3]. Edge computing is particularly used to enable mobility and one of the most successful implementations are cloudlets [10]. Another strong point of edge computing is that it can be used for offloading computation from constrained devices [4]. Previous effort in coupling data freshness and caching is presented in [11] in the context of Information Centric Networking.

The metric age of information is a concept that was firstly introduced by Kaul et al. [6] in the context of vehicular networks. The age of information is a metric for measuring data freshness and is formally defined as $\Delta(t) = t - u(t)$, where t is the current time and $u(t)$ is the time stamp of the freshest sensor update received by the cloud.

3 OUR CLOUD-EDGE-SENSOR SYSTEM

Consider a system consisting of a sensor device, an edge server, two or more clouds running different applications, as illustrated in figure 1. An application is periodically scheduled according to specifications, e.g a frequency of 4Hz (every 250ms). Hence, for periodic applications it is deterministically given when in real time the application should execute, given a start time for the first period.

Assume for simplicity, without sacrificing generality, only two applications App_k and App_l, one in each cloud. The applications are periodically executed with the frequencies f_k, f_l respectively, where $f_k > f_l$. Each cloud has a "digital twin sensor"[1], a data structure that is updated with the most recent value of the real sensor via the edge. An application will then conceptually read the twin in the same manner as if it would have been running on the sensor device or edge. This twin value is time-stamped by the sensor device when it is read, which means that an application can decide if it is stale or not according to its own clock and real-time specifications.[1] Intuitively the update frequency of the digital twin should be higher than the frequency of the application. But this is not enough, the age of the value has also to be within real-time boundaries. The value is aged by both the communication delays as well as waiting times for transmission resources.

How is the digital twin updated? The most straightforward way is the *push* model. The sensor device reads sensor data at wake-up periods and pushes the data to the edge twin for further pushes to the applications. Another model is *request/response*. Here the application instead requests an update to its twin when needed from the edge, i.e. a *pull* model. The edge in turn could also request an update from the sensor before delivering the response.

Communication. The communication path of data to the cloud has two distinct parts with different characteristics. The first part is between the sensor device and edge, typically it is a wireless sensor network optimized for small distances, energy conservation and predictable resource sharing. The second part of the path, between the edge and a cloud, is assumed to be a fully powered, high bandwidth network with regular Internet characteristics. The cloud service could be far away causing a substantial communication latency and when the network is shared there will also be considerable variance. The second part is normally dominating in terms of communication latency. The variance in the first part can in most circumstances be considered insignificant compared to the second.

[1]We assume that all clocks are enough synchronized and that clock drifts between synchronizations are not significant.

Table 1: Parameters for data aging and timing bounds.

$\Delta_a(t)$	Age of data at application period.
τ_Δ	Maximum acceptable age of data.
u_i	Time-stamp for the i:th sensor data.
r_i	Start of application for i:th data.
d_{se}	Packet delay between sensor and edge.
d_{ec}	Packet delay between edge and cloud.
$d_{v,i}$	Delay variance for i:th data.
t_{exec}	Tolerable length of application execution time.
σ	Delay before start of execution time.
t_c	Minimum required execution time.
v_i	Valid execution time for i:th data.

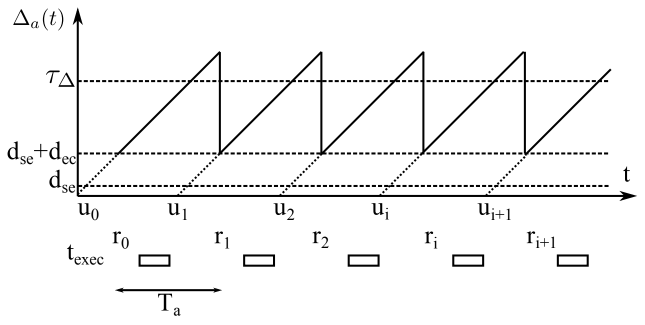
The communication latency can also be divided into a fixed, deterministic part and a varying part. The varying part includes data buffering, network contention and re-transmissions. Within the fixed we include propagation delays and transmission times assuming a known packet size. Since the wireless sensor network latency is both relatively small and typically time bounded we take the simplifying assumption to consider it as a fixed part. For this paper, packet losses are assumed to be handled by a reliable transport service. Packet re-transmissions may cause very long delays that breaks the age of information bounds. Other packet loss organization can be considered for real-time system, e.g. for periodic updates an idempotent approach may be favorable [5].

Edge. The edge server is fully powered and has the necessary computational capacity for processing and storing sensor data as well as servicing all requests from the applications. The edge also implements a digital twin that is updated by the sensor device and exported to the applications. It will initially request all applications' real-time requirements, including the update frequencies in order to optimize digital twin updates and sensor readings. Given these requirements, the edge can instruct the sensor device to change the length of the duty cycle in order to meet the aggregated requirements.

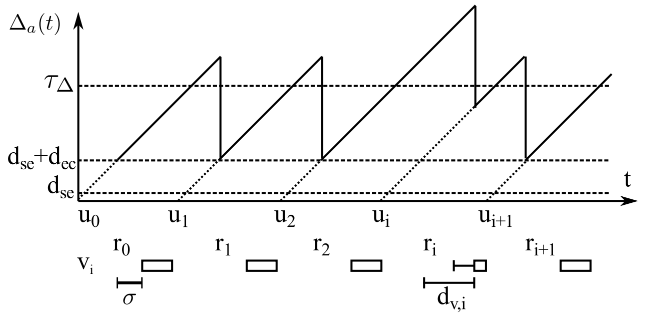
3.1 System behaviour

We now observe how sensor data ages in our system. Assume only one application in one cloud for the time being. The application is specified to run with frequency f_a, or with period T_a=1/f_a (from now on, we use period and frequency interchangeably). For a single application the sensor should produce data with at least the same frequency f_a. In other terms, the sensor will produce data periodically at time u_i, such that u_{i+1}=u_i+T_a. That data would then age on its way to the application. We assume u_0 to be the time when the sensor device starts producing updates.

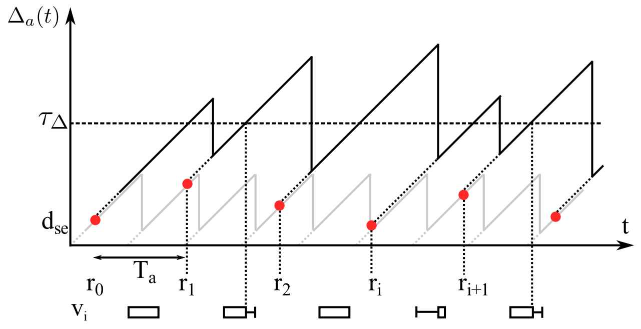
For further explaining the aging process, consider first the case with deterministic communication delays, as in figure 2(a). For a single application there is here no edge caching and the sensor data is directly propagated from the edge to the application. The delay part from the sensor to the edge is denoted d_{se} and the second part, from the edge to the twin in the cloud, is denoted d_{ec} in the figure. At time r_i=u_i+d_{se}+d_{ec} the data appears in the cloud twin with age $\Delta_a(t)$ = r_i-u_i and the application is now enabled to execute. Intuitively, the application will perceive the age $\Delta_a(t)$ behaving like a "saw tooth" function that grows linearly with time and drops whenever a new update is received with a new time stamp.



(a) Aging of data with application period T_a



(b) Aging of data with variance delay



(c) Aging of data by edge server and variance delay

Figure 2: Aging process and validity period (below the graphs).

The execution could start immediately at r_i, or in the general case with a delay σ, i.e. at time $r_i + \sigma$. The maximum allowed execution time t_{exec} is given by the application requirements. This execution interval is illustrated in the figure as an unfilled bar. The parameter t_c specifies the actual computational time needed for the execution and must be less than t_{exec}. It allows the application scheduler to decide where in the t_{exec} interval the actual computation will take place, e.g. according to an Earliest Deadline First scheduling [8].

For periodic applications we therefore identify *two timing bounds that both must be met.* The first is that the execution must take place in the period t_{exec}, with start at $r_i + \sigma$. The second bound is on the age of data. Within t_{exec} the data must also be valid with respect to age. In figure 2(a) we use τ_Δ to set the maximum allowed data age. It must be below τ_Δ at least t_c during a t_{exec} interval to be valid.

Both conditions must be fulfilled otherwise we have a deadline miss. Taken together they form a validity period, v_i, that the application only can execute in. For example, a system may specify that it should periodically produce results every 10 seconds but no later than 11 seconds but can tolerate up 2 second old data when it starts the execution.

In Figure 2(a), where we have no variance the design parameters τ_Δ, t_{exec}, σ and r_0=d_{se}+d_{ec} can hence be set to completely avoid deadline misses.

In Figure 2(b), delay variance in the second communication part is introduced as $d_{v,i}$. The other parameters are the same as for Figure 2(a). The data item for r_i is now exposed to an additional delay. As a result, the previous data in the twin will have to stay longer before being replaced, possibly beyond τ_Δ. In the figure, at time $r_i + \sigma$, the planned starting time of the execution, the existing value in the twin is too old and the application has to wait for an update that arrives at $r_i + d_{v,i}$. When $d_{v,i} > \sigma$ the valid execution period v_i hence becomes shorter. At some point it is shorter than t_c and then we have a deadline miss. Thus, with an increasing delay variance the probability of misses will also increase.

Algorithm 1 illustrates the pseudo-code for checking the validity of data. At the start of each execution period the application must first check if the age of the data in the twin sensor is within age boundaries during the time t_{exec}. If that is not the case the application is put on hold awaiting an update to the twin. When an update arrives the application once again reads the age of information and checks against the remaining execution time.

Algorithm 1 Pseudo-code of the application.

```
1:  function APPLICATION($\tau_\Delta$, $t_{exec}$, $t_c$)
2:      if $\Delta_a(t) + t_c < \tau_\Delta$ then
3:          do_job()
4:      else
5:          $\delta$ = expected_time_next_update()
6:          if $\delta + t_c < t_{exec} \wedge \Delta_a(t+\delta) + t_c < \tau_\Delta$ then
7:              do_job()
8:          else
9:              misses++
10:     APPLICATION($\tau_\Delta$, $t_{exec}$, $t_c$)
```

3.2 Multiple applications

We now consider the scenario in Figure 2(c) with two applications with periods T_k and T_l, where $T_k < T_l$. In this scenario the edge is active in caching sensor data. In order to meet the tightest update period the sensor duty cycle is consequentially set to T_k. This means that the twin at the edge will see an "aging saw tooth" with period T_k (see lower saw tooth in the figure). When the twin is updated with a new value the edge will immediately forward it to App_k, which will see its own "saw tooth", like the one for the single case.

The second application App_l has a longer period. In figure 2(c) we otherwise use the same parameters for App_l as for figure 2(b), including the delay variance to illustrate the impact of caching. In 2(c) the edge forwards data at every $r_i - d_{ec}$. The red dots are put at these times and they also indicates how much the data has aged before it is forwarded. The dashed lines from the dots are

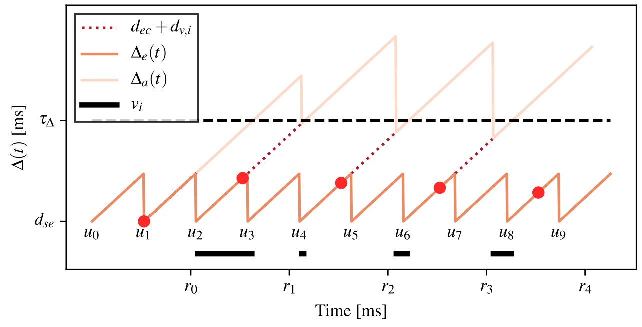


Figure 3: AITE - the edge pushes sensor updates to the digital twin according to the application period without considering aging at the edge and communication variance.

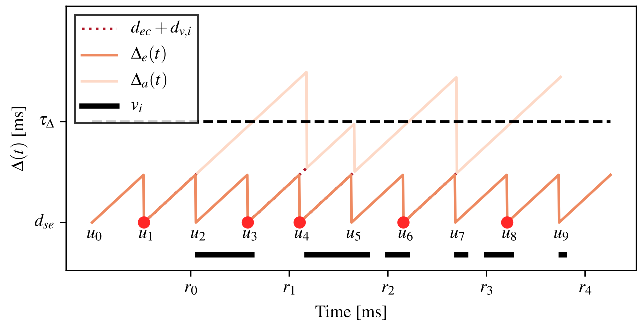


Figure 4: AITC - the edge pushes sensor updates to the application as soon as they get available.

communication times for the data to reach the cloud twin. It can be observed that the dots form a regular pattern, with equivalent distance on the edge aging teeth. This is a consequence of that T_l is not a perfect multiple of T_k. When T_l=nT_k. i.e. a multiple, all the red dots will have same age of information. The age of the data at the dots can be calculated as:

$$\Delta_l(r_i - d_{ec}) = d_{se} + [(i \cdot T_l) \bmod T_k] \tag{1}$$

The valid execution periods, v_i, are drawn at the bottom of figure 2(c). For event r_1, v_1 is shortened due an aging at the edge. For event r_i, v_i is shortened by an extended communication delay (i.e. caused by delay variance). The valid execution may be shortened in both ends and cause deadline misses.

4 SCHEDULING POLICIES

We now describe two scheduling policies in which the edge will provide mechanisms for sending sensor updates to the digital twins in the cloud according to applications execution periods.

The first scheduling policy is called "Age in the Edge" (AITE) and is designed to deliver sensor updates to an application within the execution time t_{exec}. Figure 3 shows how the age of sensor updates is evolving for this policy. In the lower part of the graph, we recognize the saw teeth at the edge and the validity periods.

The main drawback with AITE is the aging at the edge. The data is already aged when pushed to the cloud and it may then also be exposed to network congestion prolonging the aging further. An intuitive solution to improve the situation would be to more frequently push updates towards the digital twin. For example, all sensor updates could be forwarded to the twin. This solution will however increase the number of updates in the network which is not desirable.

The second scheduling policy is named "Age in the Cloud" (AITC) and is illustrated in figure 4. It is designed to send the updates at the beginning of a saw tooth rather than let them age at the edge before pushing them as in AITE. Since it is predictable which updates will be aged at the edge it is possible to reschedule the updates so that there will always be a fresh value. Such a predicted aged update could be sent as soon as it arrives, which is the case for u_4 or the update could be delayed to the next fresh value as for u_3. When it is sent earlier the update will age in the cloud instead but has the advantage that it may tolerate communication variance better. When it is sent later, the data is fresher and it may tolerate better the age boundary even if it arrives later in the t_{exec} period.

In AITC the validity period will increase compared to the AITE scenario. We show differences between AITE and AITC in figures 3 and 4 respectively.

5 EVALUATION

In this section we evaluate the aforementioned AITE and AITC scheduling policies from the point of view of three different metrics. First of all, we observe the experienced age at the cloud $\Delta_c(t)$ upon reception and at application execution $\Delta_a(t)$. Then, we analyze the amount of time an update is usable before breaking the deadlines τ_Δ, $\tau_\Delta - \Delta_c(t)^+$[2]. Finally, we discuss the validity execution interval of an update, v_i. These metrics are important as they reflect application's execution flexibility and probability of experiencing real-time misses.

In our evaluation setup, the sensor device produces a new value once every 100ms. We assume a fixed communication delay between the sensor device and the edge of $d_{se} = 1$ms. The cloud infrastructure is placed at a transmission and propagation distance of $d_{ec} = 100$ms from the edge. The communication is affected by an additional delay with variance ($d_{v,i}$), according to a Pareto distribution with an expectation value of 14ms. We arbitrarily selected Pareto since it is commonly used for Internet delay distributions, but the same general behavior and main conclusions would apply to other distributions. The application in the cloud is executed once every $T_a = 190$ms with an execution time of $t_{exec} = 152$ms starting from the beginning of the application period (i.e. $\sigma = 0$) and $t_c = 0$. The upper bound on age of information for this application is set to $\tau_\Delta = 210$ms.

We perform numerical evaluations based on the aging model, the above parameters and the two scheduling principles described in previous section, illustrated in figures 3 and 4. We collect the results from 5000 application readings of the twin sensor in the cloud. These 5000 readings are affected by 5000 delay samples from the Pareto distribution. The maximum value of these samples is 161ms.

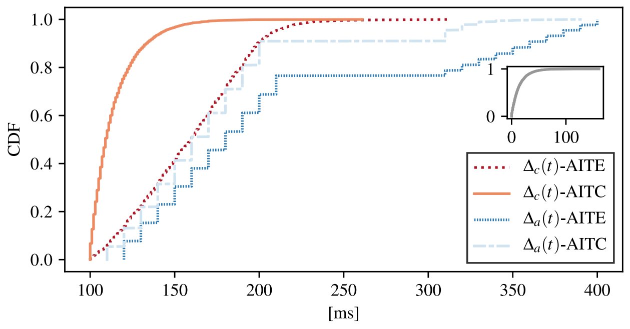


Figure 5: CDF of age of information for AITE and AITC upon updates reception at the cloud, $\Delta_c(t)$, and at application, $\Delta_a(t)$. On the right the variance distribution is shown.

5.1 Numerical results

Figure 5 shows the cumulative distribution function of the age of information of the updates when they are received at the digital twin in the cloud, $\Delta_c(t)$, and when the application is executed, $\Delta_a(t)$. First of all, we discuss $\Delta_c(t)$. The distribution of the age of information for AITE is basically linear due to the misalignment between the sensor and application periods. This misalignment, in the long run, forces the age of information at the application to be uniformly distributed throughout the aging saw tooth. Furthermore, communication variance (showed in the small box at the right) is additive and extends age of information's upper-bound. The distribution of AITC, on the other hand, follows closely the variance distribution since all the updates are sent immediately upon reception at the edge. For this reason, the age of information $\Delta_c(t)$ follows the variance $d_{v,i} + d_{se} + d_{ec}$. These two CDFs confirm what can be expected about the relative parameters of edge aging and expected variance.

From the point of view of the application, $\Delta_a(t)$, AITE shows the worst performance. In fact, the misalignment of different periods impacts even more the already degraded distribution of $\Delta_c(t)$ for AITE. On the other hand and despite the same misalignment problem, AITC performs better than AITE. Intuitively, the better freshness exhibited by "$\Delta_c(t)$-AITC" approximately translates the same "$\Delta_a(t)$-AITC" closer to the origin on the X-axis. The previous insights are crucial to properly dimension real-time requirements, including τ_Δ.

Figure 6 presents the cumulative distribution function of the amount of time an update is usable before breaking the timing requirement τ_Δ, "$\tau_\Delta - \Delta_c(t)^+$". In other words, we can see for how long one update will be valid before it gets too old for correct processing. To better understand this plot, we provide the insight that the more the curve is distributed on the right, the better. In fact, the higher the numbers on the X-axis, the longer the validity time on age of information. We observe that, for AITE, roughly 4% of the updates is already too old for usage at the arrival at the cloud (value 0 in the plot). Maybe without surprise, AITC obtains better results. The validity in AITC is better because the updates are forwarded upon reception. Nonetheless, having a larger margin to τ_Δ is not

[2] Only positive values allowed, whenever negative the returned value is zero.

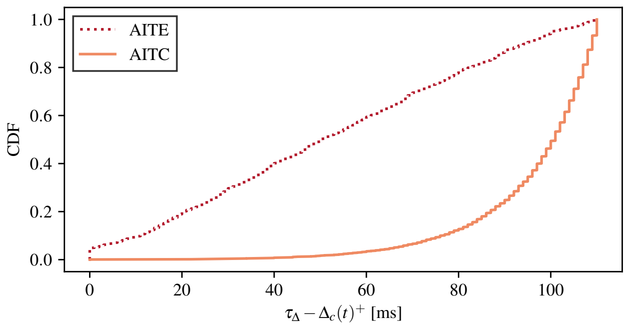


Figure 6: CDF of the amount of time an update is usable before breaking timing requirement τ_Δ.

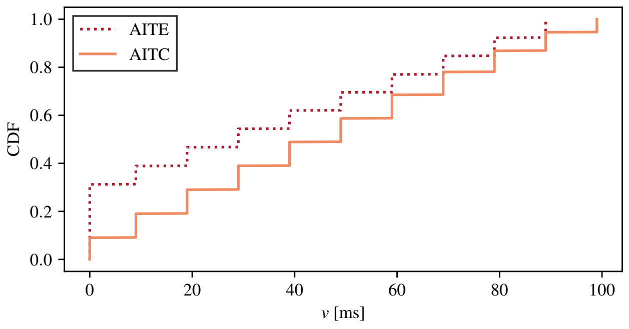


Figure 7: Validity v of the updates at the application.

enough since the update must also be valid during the application's execution time t_{exec}. This leads us to our last proposed evaluation metric.

Figure 7 shows the cumulative distribution function of v, the validity interval of an update within the application's execution time t_{exec}. As for the previous plot, the more the CDF is distributed on the right, the better. Also in this case, AITE performs worse than AITC. For instance, 34% of the updates are not valid with respect to t_{exec} (CDF value 0). This confirms the hypothesis made in section 3.1 where we stated that both aging at the edge before transmission and communication variance reduce the validity v for such application. In contrast, AITC outperforms AITE as it is designed to deliver the freshest updates as close as possible to the application execution. From the figure we observe that we obtain only 9% of misses and we also have significant improvement in the validity of the received updates at the application.

5.2 Discussion

We evaluated two different scheduling policies, namely, AITE and AITC. We demonstrated that aging in the cloud is better than aging in the edge, especially when updates must travel through the Internet and are exposed to non-negligible communication variance. Furthermore, scheduling with knowledge about misalignment between periods helps real-time applications in meeting their timing requirements without the need to emit updates with higher frequencies, which would increase the network traffic and reduce sensors battery life.

One may argue that the communication model used is too simplistic, but this is done on purpose. In fact, the system already relies on numerous parameters spread over several distributed components. Adding further complexity will mask the base overall functioning of the system impacting the understanding of the proposed challenges and solutions.

6 CONCLUSION

We study the behaviour of a distributed real-time systems composed of sensor devices, wireless sensor networks, edge server(s) and a remote cloud infrastructure. We make use of the age of information as a main metric for evaluating the freshness of sensor updates at cloud based periodic applications with timing requirements. For periodic applications we identify two validity timing bounds on sensor data that both must be met. The first is that the data must not be too old for the application. The second is that even if the data is fresh enough it must also be available within the specified execution time interval of the application to be valid. We showed that an edge scheduler can play a fundamental role for delivering fresh data within a specified execution interval of a periodic application. We evaluated numerically two scheduling policies providing insight on how to improve the validity time interval of updates and we discussed trade offs among the several parameters of the system. The purpose of the evaluation is to show the potential gain of scheduling updates to applications with mismatched periodicity and duty cycled sensors. Even if the evaluation is not comprehensive we draw the conclusion that there are considerable possible improvements a designer of an edge-cloud based real-time system should be aware of. A more comprehensive study scaling for several applications, cloud services, more realistic communication delays and different scheduling parameters remains for future work.

7 ACKNOWLEDGEMENT

This work was supported by the Swedish Foundation for Strategic Research under the project "Future Factories in the Cloud (FiC)" with grant number GMT14-0032.

REFERENCES

[1] K. M. Alam et al. 2017. C2PS: A Digital Twin Architecture Reference Model for the Cloud-Based Cyber-Physical Systems. *IEEE Access* (2017).
[2] Flavio Bonomi et al. 2012. Fog Computing and Its Role in the Internet of Things *(ACM MCC '12)*.
[3] Nigel Davies et al. 2016. Privacy Mediators: Helping IoT Cross the Chasm *(ACM HotMobile '16)*.
[4] Kiryong Ha et al. 2014. Towards Wearable Cognitive Assistance *(ACM MobiSys '14)*.
[5] Pat Helland. 2012. Idempotence is Not a Medical Condition. *Commun. ACM* (2012).
[6] S. Kaul et al. 2011. Minimizing age of information in vehicular networks *(IEEE SECON 2011)*.
[7] S. Kaul et al. 2012. Real-time status: How often should one update?. In *2012 Proceedings IEEE INFOCOM*.
[8] C. L. Liu et al. 1973. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *J. ACM* (1973).
[9] M. Satyanarayanan. 2017. The Emergence of Edge Computing. *Computer* (2017).
[10] M. Satyanarayanan et al. 2009. The Case for VM-Based Cloudlets in Mobile Computing. *IEEE Pervasive Computing* (Oct 2009).
[11] S. Vural et al. 2017. Caching Transient Data in Internet Content Routers. *IEEE/ACM Transactions on Networking* (April 2017).