# Bounded transmission latency in real-time edge computing: a scheduling analysis

Pietro Fara
*Scuola Superiore Sant'Anna*
Pisa, Italy
pietro.fara@santannapisa.it

Gabriele Serra
*Scuola Superiore Sant'Anna*
Pisa, Italy
gabriele.serra@santannapisa.it

Federico Aromolo
*Scuola Superiore Sant'Anna*
Pisa, Italy
federico.aromolo@santannapisa.it

***Abstract*—With the recent advancements in computing power and energy efficiency, embedded system platforms have become capable of providing services that previously required computational support from cloud infrastructures. Accordingly, the edge computing paradigm is becoming increasingly relevant, as it allows, among other advantages, to foster security and privacy preservation by processing data at its origin. On the other hand, these systems demand predictability across the IoT-edge-cloud continuum. Regardless of the communication link, real-time tasks at the edge send data on the network, employing one or more transmission queues. For a system designer, analyzing the timing behavior of a task becomes challenging when each task has to wait for a variable amount of time before sending a packet. This paper analyzes the transmission behavior of a network of nodes regarding the latency introduced when dealing with a communication interface. The proposed analysis provides necessary conditions under which the data traffic is guaranteed not to exceed the transmission queue limit, thus avoiding unbounded waiting times on task execution, while a response time analysis technique is provided to ensure the schedulability of periodic tasks executing in each node of the network. An experimental campaign was carried out to evaluate the schedulability performance obtained with different system configurations when the proposed analysis was applied.**

## I. Introduction

Networked embedded systems have become pervasive in a large number of application domains, ranging from agriculture to transport. The functional behavior of these systems directly impacts the safety of living beings and objects. Recent technology advancements also made it possible to realize more powerful embedded platforms capable of providing services that previously required computational support from remote servers to be accomplished. In this scenario, the edge computing paradigm is becoming increasingly popular. This paradigm leverages the computational capabilities of modern embedded platforms to enable the processing of large amounts of data closer to its origin, drastically reducing the amount of data sent to the cloud, thus facilitating timely computation, reducing energy consumption, and preserving data privacy.

Edge computing is becoming a key driver of innovation in several fields, such as the automotive industry. In this case, vehicles are augmented with connected-car services that provide advanced functionalities and enhanced user experience. However, automotive manufacturers must carefully design the networking services behind the operational curtain. When the functions performed by the edge-computing system need to interact with the external environment, as is typical of a cyber-physical system such as a vehicle, time predictability across the edge-to-cloud continuum is crucial to ensure the safe behavior of the overall design. In addition, the usage of system resources must also be kept to reasonable amounts. These growing requirements can be approached by devising improvements in network architecture, computing infrastructure, and data availability. When dealing with interconnected systems, the edge computing nodes are expected to deliver the result of the computation within a predefined deadline, for instance, by transmitting a chunk of data across the network. Implementing packet transmission at the node level impacts the timing behavior of the system and calls for investigating different strategies to schedule transmission-related activities in a time-predictable fashion. Indeed, software tasks running in an edge node commonly place packets in a transmission queue during their execution. When the transmission queue is full, the task must wait for a potentially unbounded amount of time before transmitting a packet. Furthermore, the device transmission protocol takes some time to be carried out, and the transmission device must be used as a shared resource protected by a specialized real-time synchronization mechanism. These design peculiarities make it particularly challenging to analyze the timing behavior of these kinds of systems, and deriving a precise bound on the delay introduced by a complex edge network is considered a challenging endeavor. Nevertheless, it is interesting to analyze the type of latencies introduced by the transmission device and how they must be accounted for when analyzing the overall timing behavior of the edge node. Indeed, analyzing the timing behavior of a task becomes challenging when each task has to wait a variable amount of time to send a packet. As a simple example, consider a system composed of two tasks scheduled under a fixed-priority schema, as shown in 1. The higher priority task $\tau_1^1$, during its execution, sends several packets of data. Packets are copied into the queue of the transmission device. Then, at some point, the lower priority task $\tau_2^1$ arrives and must, in turn, send several packets in the network. However, assuming the transmission queue is already full, the lower priority task must wait for the queue to have at least one free slot before it can start transmitting. In this scenario, it is possible that a deadline miss can occur unless a precise estimate of the time required to clear a transmission queue slot is provided.

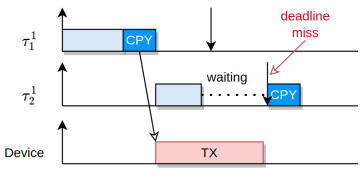


Fig. 1. Fixed-priority schedule of two tasks in an edge node. Both tasks require the transmission device. The higher-priority task copies several data packets into the device queue. The lower priority task must, in turn, send several packets out. Assuming the transmission queue is already full, it must wait until there is enough space. Therefore, the deadline for the lower-priority task might be missed.

This work presents a transmission model to analyze the different sources of latency introduced when accessing a communication interface in an edge network scenario. The proposed model ensures that the data traffic does not exceed the transmission queue limit in order to avoid unbounded waiting times during task execution. Furthermore, a response time analysis is proposed to evaluate the schedulability of all software tasks in the network under a multiple-node scenario, such as those commonly adopted in crowdsensing applications or when dealing with smart sensors network.

**Contributions.** In summary, this work makes the following contributions: *i)* it presents a detailed analysis of the latency that a packet can experience during the transmission from an edge node; *ii)* it derives an analysis that prevents the device transmission queue from being full, bounding the time required for a task to send data through the communication peripheral; *iii)* it derives a scheduling analysis considering a multiple-node scenario; *iv)* it reports the results of an experimental evaluation of the proposed response time analysis approach in terms of the schedulability ratio over randomly generated task sets.

Since this work only addresses how transmission operations are scheduled, other aspects, such as packet recovery strategies, are not discussed as they depend on the target application and the adopted network protocol.

To the best of our knowledge, this is the first paper presenting a scheduling analysis on a distributed scenario that takes into account transmission latencies.

## II. Related Work

Edge computing has become an increasingly popular networking paradigm due to increased computational power and decreased physical size of specialized sensor/actuator embedded devices. Edge devices of this kind are designed to satisfy the demands of applications that are both computation-intensive and delay-sensitive. On the other hand, ensuring a predictable timing behavior and resource utilization requires specialized modeling and analysis techniques to guide the design of the system. Numerous works in the literature focus on investigating techniques related to the concept of quality of service (QoS) for delay-sensitive applications running on the edge. The performance evaluation of real-time services in networks is, however, particularly challenging. Real-time services require firm QoS guarantees, usually formulated by computing an end-to-end delay bound for each node's packet transmission, reception, and computation. Several results in the literature rely on the application of Network Calculus [1], a theory for deterministic network performance analysis. For instance, Mei et al. [2] analyzed the overall delay bound for packet transmission by employing stochastic network calculus (SNC) for mobile edge computing (MEC) networks. MEC is a promising computing paradigm that leverages the availability of Internet of Things (IoT) devices and infrastructure. In particular, MEC is a distributed computing architecture that allocates computation resources to computing servers deployed at the edge of the network to handle different levels of QoS. An overview of this architecture was presented in [3]. Based on MEC, numerous studies investigate how to improve the overall performance of edge-to-cloud applications in terms of latency. Ren et al. [4] formulated an optimization problem to solve the problem of splitting the computation of a single task into two parts, one executed at the edge and the other one on the cloud. Another approach to optimize transmission latency, based on data compression, was presented in [5]. This approach analyzes three different models: local compression, edge-cloud compression, and partial compression offloading. In the case of compression offloading, an optimization problem is provided to find an optimal configuration that minimizes the overall latency. The increasing requirements of processing time in mobile applications may also represent another source of increased latency. Jia et al. [6] proposed a heuristic offloading method for computation-intensive applications in MEC, taking into account both cloud service latencies and computation time and leveraging load-balancing techniques. Yi et al. [7] introduced a technique to offload computation between client and edge nodes and distribute computation among nearby edge nodes to provide low-latency video analytics closer to a mobile device and its user. Latency is also strongly dependent on how transmission queues are managed. Aamir et al. [8] proposed a buffer management scheme to contain the packet loss ratio. Another queue management mechanism, based on dividing the main queue into two sub-queues, has been developed in [9]. These works focus on reducing the end-to-end latency in data transmission across the edge network. Differently from these works, this paper provides a real-time schedulability analysis for nodes sending packets periodically, together with a bound on the expected usage of the transmission resource.

A common requirement in the literature concerning wireless sensor and actuator networks (WSAN), especially in the field of industrial wireless communication, is to satisfy the end-to-end timing requirements of all tasks. Given that stringent timing and reliability requirements for WSANs are similar to those in place for tasks in real-time scheduling, recent works focused on exploring multiprocessor scheduling theory's application to wireless edge networking. The works by Lu et al. [10] and Gutiérrez-Gaitán et al. [11] provide an overview of the most significant research efforts in this direction. In particular, Gutiérrez-Gaitán et al. [11] takes combines existing analyses for multiprocessor scheduling with the modeling of the transmission channel; however, the focus is on deriving an

end-to-end delay analysis rather than a scheduling analysis at the node level and at the overall system level.

## III. System model

This work analyzes delays introduced during data transmission from/to edge nodes in a replicated-node scenario. Commonly, this system architecture can be adopted when designing crowdsensing applications or network of smart sensors. Under this scenario, $z$ refers to the number of edge nodes in the set $\mathcal{E} = \{E_1, \ldots, E_z\}$ taking part in the network. We consider the case in which each edge node consists of an embedded system platform with a single processor (i.e. a microcontroller). On each node $k$, a set $\Gamma_k = \{\tau_1^k, \ldots, \tau_n^k\}$ of $n$ periodic tasks is executed. Tasks in $\Gamma_k$ are scheduled on the processor by a preemptive fixed-priority scheduling algorithm and activated simultaneously without any initial offset. Each periodic task $\tau_i^k$ is characterized by a worst-case execution time (WCET) $C_i^k$, a release period $T_i^k$, and a relative deadline $D_i^k \leq T_i^k$, assuming that task $i$ of each edge node $k$ has the same period and same deadline such that $T_1^1 = T_1^2 = \ldots = T_1^z$ and $D_1^1 = D_1^2 = \ldots = D_1^z$ respectively. We denote with $H$ the hyper-period among all periods of tasks $\tau_i^k$, given by the least common multiple of the periods.

The set of higher-priority tasks concerning $\tau_i^k$ that execute on the same node $E_k$ is denoted by $hp(i,k)$.

Each node is connected to a communication network `CN`. To be as general as possible, this work does not take into account a specific type of network link, with the advantage that the proposed analysis can be applied to both wired and wireless networks. The edge node exposes, on the network `CN`, an interface capable of sending and receiving data. Consider the following; we use `NI` to refer to the node interface representing the interfaces exposed by a node.

Data transmission via the `CN` occurs by acting on several memory-mapped device registers.

In every type of network, especially in those that share the communication channel (e.g., wireless networks), a packet transmission may suddenly result in a packet loss due to the contention of several transmission collisions or interference from the environment. For analysis, the `CN` is modeled as a network with error loss probability $P(e)$, where $e$ is the event of losing a packet.

The `NI` provides an output (and, respectively, an input) buffer organized as a first-in-first-out (FIFO) queue of $q^{\text{NI}}$ elements, each system's architecture such registers consists of performing several writing and reading operations on the memory-mapped device registers. The minimum read/write rate to access such registers is indicated with $\beta$ (in terms of bytes per time unit), while the maximum rate is denoted by $\beta^{max}$. Furthermore, the `NI` guarantees a minimum transmission rate on the link indicated by $\alpha$ (bytes per time unit).

To send data out from `NI`, the content of a packet must be copied from the task memory to the device queue. Memory write times are generally shorter than read times; however, we denote with $\gamma$ the minimum read/write rate to access the memory. The model considers an equal rate for read/write operations as it does not particularly affect the results of this work. If the `NI` transmission queue is full, a task must wait to send data until one of the slots becomes empty.

To summarize, when a task needs to transmit $x$ bytes of data, the `NI` needs **(i)** at most $x/\gamma$ time units to read the data from the task memory, **(ii)** at least $x/\beta^{max}$ time units and at most $x/\beta$ time units to copy the mentioned data into the `NI` queue and **(iii)** at most $x/\alpha$ time units to transmit such data into the communication link.

After its computation time, a task $\tau_i^k$ sends $M_i$ data packets over the network in broadcast. We assume that tasks with the same index in different nodes send the same number of packets $M_i$. Every edge node receives the packets. Each packet has a fixed size of $b$ bytes, i.e., a size equal to that of the FIFO queue elements. Not every periodic task may produce data that must be sent over the network; hence $M_i$ can also be zero. For the sake of readability, in the following sections, we use $M_i^* = M_i + \lceil M_i \cdot P(e) \rceil$ to denote the total number of packets sent, where $\lceil M_i \cdot P(e) \rceil$ are those retransmitted due to package loss. In this analysis, if the transmission fails, only one retransmission is made. Furthermore, we are assuming that, if both packet transmissions fail, the receiving nodes do not wait for data, but they still use the last received one.

When a packet arrives at the receiving node, the `NI` notifies the receipt through interrupts. The `NI` raises an interrupt every time a packet is received. The corresponding *interrupt service routine* (ISR) is in charge of reading the packets in the queue by reading on the `NI` device registers and copying them into a memory buffer shared with the task interested by the packet. Each ISR introduces an overhead of at most $\sigma^{\text{ISR}}$ time units due to its activation and completion management.

The network interface `NI` is considered a shared resource among the tasks in $\Gamma_i$ that send or receive data, i.e., among the tasks for which $M_i > 0$. Therefore, data transmission is performed through the `NI` in a mutually-exclusive way. Thus, each task must acquire and release a lock before transmitting and receiving each packet. The immediate priority ceiling (IPC) [12] locking protocol is adopted to avoid priority inversion phenomena while protecting the `NI` among the tasks for which $M_i > 0$. Therefore, the critical sections in which a task $\tau_i^k$ accesses the `NI` are accessed in mutual exclusion. According to the chosen protocol, a task could be blocked during the critical section of the transmission phase due to the exclusive use of the devices. In the following, let $PT$ represent the total blocking time experienced by each task due to the non-preemptive transmission phase of a task.

## IV. Transmission analysis

This section provides an analysis of the transmission latencies introduced by the `NI`, deriving a condition to ensure that the number of packets sent through the `NI` peripheral does not exceed the size of the FIFO queue to avoid introducing unbounded blocking.

### A. Latency modeling

*Definition 1:* The transmission latency $\Delta_{NI}$ due to a packet transmission through the `NI` is defined as the time that elapses

since a packet is stored in the transmission queue by the sender to the time the packet can be considered sent out and thus available for the subsequent node of the network.

In the following subsection, the latency introduced by NI is studied employing a model commonly used to account for delays in networks of routers [13][14]. Under this model, $\Delta_{NI}$ can be decomposed as a sum of four different components: a propagation latency $d_{prop}$, a transmission latency $d_{trans}$, a processing latency $d_{proc}$, and a queuing latency $d_{queue}$.

Therefore, the total latency due to the NI packet transmission can be computed as:

$$\Delta_{NI} = d_{prop} + d_{trans} + d_{proc} + d_{queue}$$

This analysis aims to show that the transmission latency $\Delta_{NI}$ can be bounded rather than providing an exact estimation of the latency $\Delta_{NI}$. In order to show that there exists a static upper bound $\Delta_{NI}^{max}$ for the latency $\Delta_{NI}$, each component is analyzed separately. In the following subsections, we investigate the latency sources characterizing each transmission latency component $\Delta_{NI}$.

*1) Propagation latency:* Once data bits are pushed into one end of a link of the CN network, they must propagate to the other end. The propagation latency depends on the channel length and the signal propagation speed for the given link medium. In practice, it can be computed as the distance between the link's endpoints divided by the propagation speed across the channel. The signal propagation speed is affected by multiple factors. Although some specific wireless networks (e.g., wireless acoustic networks) may have a propagation latency of up to several milliseconds, in the majority of cases, if the node is connected employing a network cable or a wireless network, the latency due to the propagation speed is in the order of nanoseconds. Furthermore, $d_{prop}$ differs per pair of edge nodes because the distances among nodes can significantly vary. Hence, we consider $\sigma_p^{s,r}$ as the propagation latency for transmitting a packet from node $E_s$ to node $E_r$.

*2) Transmission latency:* The transmission latency is an amount that represents the time required by the device to transmit a chunk of bytes into the link and depends on the device's bandwidth. It is computed as the number of bytes to be pushed in the link divided by the transmission bandwidth. Given that each task exchanges data packets with a fixed size of $b$ bytes and that the transmission rate over the link is given by $\alpha$, we can compute $d_{trans}$ as $d_{trans} = \frac{b}{\alpha} = \sigma_d$, with $\sigma_d$ being a constant factor to account for the transmission time demanded by each packet.

*3) Processing latency:* The processing latency represents the time required to read and write the device registers involved in the transmission and to perform the necessary register shifts. This time is fixed for each packet and does not depend on the packet length. Therefore, we consider $d_{proc} = \sigma_o$, with $\sigma_o$ being the maximum data-size-independent processing overhead.

*4) Queuing latency:* The queuing latency represents the time each packet must wait in the queue before transmitting it to the link. The queuing latency will be null when the queue is empty, and no other packet is transmitted. Otherwise, the queuing latency for a packet depends on the number of currently enqueued packets waiting for transmission.

Assuming that each packet transmission requires a time equal to the transmission latency $d_{trans}$ plus the processing latency $d_{proc}$, the transmission of the $N$-th packet in the queue will start after $(N-1)\cdot(d_{trans}+d_{proc})$ time units have elapsed. Therefore, the maximum queuing latency can be computed as $d_{queue} = (q^{\text{NI}} - 1) \cdot (\sigma_d + \sigma_o)$.

*5) Overall latency upper bound:* Given that the propagation latency was modeled as having a dependency on a specific pair of nodes $(E_r, E_s)$, with $E_r \in \mathcal{E}$ and $E_s \in \mathcal{E}$ as the receiving node and the sending node respectively, the resulting value of the overall transmission latency $\Delta_{NI}$ will also depend on the specific pair. Let $\Delta_r^s$ represent the transmission latency between the nodes in the pair $(E_r, E_s)$. The value of $\Delta_r^s$ can be upper bounded as $\Delta_r^s \leq q^{\text{NI}} \cdot (\sigma_o + \sigma_d) + \sigma_p^{s,r}$.

### B. Queuing analysis

In this subsection, we analyze how the sending and the receiving queues of the NI are filled and emptied over time and how this can affect the schedulability of the task set. The proposed transmission model considers the NI queues as finite queues. Therefore, when the queue is full, a task cannot push another packet into the queue and must perform a wait or decide to abort the send operation. To avoid unbounded waiting times or packet drops, the task must perform the send operation when the queue has at least one available free slot. Analogously, when receiving, if the receiving queue is full, then the packet will be lost.

In the following, we derive the necessary conditions on the sending and receiving behavior of each node to ensure that, for each sending operation, there are enough available packet slots in the sending queue. On the other hand, when receiving packets, we ensure that there are enough available packet slots in the receiving queue.

### C. Sending

Assume that the queue can host infinite packets to prove the following lemma. Recall that $\alpha$ indicates the transmission bandwidth. Let $B(t)$ represent the total amount of bytes for which transmission has been requested through the NI within the interval $[0, t)$. We define the utilization of the transmission interface $\overline{U}^{\text{NI}}(t)$ as $\overline{U}^{\text{NI}}(t) = \frac{B(t)}{\alpha t}$.

*Lemma 1:* The latency component caused by queuing $d_{queue}$ can be bounded during the entire system service only if

$$\overline{U}^{\text{NI}}(H) \leq 1$$

**Proof.** Given that the transmission bandwidth $\alpha$ is constant, if $\frac{B(H)}{\alpha H} > 1$, then $B(H) > \alpha H$, therefore the rate at which bytes arrive at the queue exceeds the rate at which the NI device can transmit the bytes. Consequently, the queue will tend to grow with no bound, making the queuing latency infinite. Note that the behavior of the task set repeats every $H$

time unit; therefore, the transmission pattern also repeats every $H$. Hence, testing the inequality for $t > H$ is unnecessary. □

In the following timing analysis, we assume, as a necessary condition, that $U^{\text{NI}}$ is less than or equal to 1. Given that all tasks $\tau_i^k$ are periodic, $U^{\text{NI}}$ can be computed as $U^{\text{NI}} = \sum_{i=1}^{n}(M_i^* \cdot b)/T_i$. Even if the condition $U^{\text{NI}} \leq 1$ holds, the nature of $\overline{U}^{\text{NI}}(t)$ impacts the latency. If a packet arrives every $\frac{b}{\alpha}$, it surely finds the queue empty. In reality, multiple packets can arrive simultaneously, in bursts. Hence, $U^{\text{NI}}$ does not represent what is happening to the packet queue. Therefore, we must ensure that the current packet burst does not exceed the number of available slots in the queue.

We begin by bounding the amount of data sent within arbitrary time windows. The following lemma holds if the task set $\Gamma_k$ is schedulable. Therefore, if the task set is not schedulable following the response-time analysis, then the resulting response time bounds should not be considered valid (e.g., they are not representative of the lateness of the tasks). For the sake of readability, in the following lemmas, we are using $M_i^*$ to consider all the packets sent, including those retransmitted because of package loss.

*Lemma 2:* In any time window of length $t$, the tasks in $\Gamma_k$ can input in the NI queue at most $g(t)$ bytes of data, where

$$g(t) = \min\left\{\sum_{i=1}^{n}\left\lceil\frac{t+T_i}{T_i}\right\rceil M_i^* b,\ \beta^{max}t\right\} \tag{1}$$

**Proof.** This Lemma was already proved in [15], Lemma 1. □

The above lemma can derive a safe condition under which the NI queue is never full.

*Lemma 3:* No task can find the NI queue full when sending a packet if

$$\forall t > 0,\quad g(t) - \alpha t \leq q^{\text{NI}} \cdot b. \tag{2}$$

**Proof.** This Lemma was already proved in [15], Lemma 2. □

Lemma 3 does not represent a practical and efficient test since it requires checking the inequality for any positive value of $t$. The following lemma shows how to limit the test to a finite number of checkpoints.

*Lemma 4:* No task can find the NI queue full when sending a packet if, $\forall t \in \Phi,\quad g(t) - \alpha t \leq q^{\text{NI}} \cdot b$, where

$$\Phi = \bigcup_{i=1}^{n}\{kT_i + \epsilon \leq t^*, k = 0, 1, 2, \ldots\} \cup \{\psi\}, \tag{3}$$

with

$$t^* = \frac{2\sum_{i=1}^{n} M_i^* b}{\alpha - \sum_{i=1}^{n}\frac{M_i^* b}{T_i}}, \tag{4}$$

$$\psi = \left\{t \leq t^* \ \mid\ \sum_{i=1}^{n}\left\lceil\frac{t+T_i}{T_i}\right\rceil M_i^* b = \beta^{max}t\right\}, \tag{5}$$

and $\epsilon > 0$ is an arbitrarily small positive number.

**Proof.** This Lemma was already proved in [15], Lemma 3. □

The result in Lemma 4 provides a practical test to ensure that no task can find the NI queue full.

### *D. Receiving*

In order to derive necessary conditions that ensure bounded behavior on the receiving end, it is necessary to estimate the maximum number of packets that can be enqueued in the sending queue at any given time. In fact, the number of packets that are present in the sending queue contributes to the queuing latency $d_{\text{queue}}$. Recall that the queuing latency $d_{\text{queue}}$ depends on the status of the sending queue when the transmission is performed. This latency will affect the arrival pattern of packets in the receiving queue.

The number of packets contained in the sending queue can be bounded according to the following lemma.

*Lemma 5:* The NI queues never contain more than $Q^{\text{MAX}}$ packets, where

$$Q^{\text{MAX}} = \max_{t\in\Phi}\left\{\left\lceil\frac{g(t)-\alpha t}{b}\right\rceil\right\}, \tag{6}$$

and $\Phi$ is defined as in Lemma 4.

**Proof.** This Lemma was already proved in [15], Lemma 5. □

Given that the queuing is managed in FIFO order, the maximum time a packet can be delayed while being in the queue is guaranteed to be smaller or equal than the cumulative transmission time of all the preceding packets in the queue, which can be at most $Q^{\text{MAX}} - 1$. It follows that the queuing delay $d_{\text{queue}}$ can be bounded as:

$$d_{\text{queue}} \leq (Q^{\text{MAX}} - 1) \cdot b/\alpha. \tag{7}$$

Given this less pessimistic upper bound on $d_{\text{queue}}$, the value of $\Delta_r^s$ can be estimated as

$$\Delta_r^s = Q^{\text{MAX}} \cdot (\sigma_o + \sigma_d) + \sigma_p^{s,r}. \tag{8}$$

When receiving data, a single node $E_r$ may potentially receive $M_i$ packets from each task of the other $z-1$ nodes. Hence, Lemma 2 has to be modified to account for the fact that the node is receiving packets from all the other nodes.

*Lemma 6:* In any time window of length $t$, the tasks can receive in the NI queue at most $h_s(t)$ bytes of data from node $E_s$, where

$$h_s(t) = \min\left\{\sum_{i=1}^{n}\left\lceil\frac{t+T_i}{T_i}\right\rceil M_i b,\ \alpha t\right\}. \tag{9}$$

**Proof.** This lemma is analogous to Lemma 2, except for the coefficient $\alpha$, which replaces the coefficient $\beta^{max}$ in Lemma 2. In this case, the queue is filled with a rate $\alpha$ because the NI receives packets from node $E_s$ through the communication channel. The lemma follows. □

Given that all the transmissions are performed in broadcast, each node receives packets from the other $z-1$ sending nodes. Hence, the condition concerning the filling of the receiving queue is slightly different because it must consider that the amount of received packets is greater. Furthermore, the following lemma assumes $t-\Delta_r^s$ as an argument for $h_s(t)$ to account for the transmission latency between node $E_s$ and node $E_r$. When node $E_s$ transmits a packet, $E_r$ will only receive it after $\Delta_r^s$ time units.

*Lemma 7:* NI never finds the queue full if

$$\forall t > 0 : t - \Delta_r^s > 0, \qquad \sum_{\substack{E_s \in \mathcal{E} \\ E_s \neq E_r}} h_s(t - \Delta_r^s) - \alpha t \leq 0$$

**Proof.** Assume, by contradiction, that at a particular time instant $t_1$ we have $\sum_{\substack{E_s \in \mathcal{E} \\ E_s \neq E_r}} h_s(t_1 - \Delta_r^s) - \alpha t_1 > 0$. This means that the sending nodes are transmitting with an aggregate rate greater than the receiving rate $\alpha t$ of the receiving node, resulting in a packet loss. The lemma follows. □

Similarly to Lemma 3, we can reduce the number of points to be tested to a finite amount, as shown in the following lemma.

*Lemma 8:* NI never finds the queue full if, $\forall t \in \Phi_r$, $\sum_{E_s \in \mathcal{E}, E_s \neq E_r} h_s(t - \Delta_r^s) - \alpha t \leq 0$, where

$$\Phi_r = \bigcup_{\substack{E_s \in \mathcal{E} \\ E_s \neq E_r}} \bigcup_{\tau_i \in \Gamma_s} \{kT_i + \epsilon + \Delta_r^s \leq H, k = 0, 1, 2, \ldots\} \tag{10}$$

**Proof.** Similarly to Lemma 3, the checking points are the activation of each job of each task of each edge node except the receiving node $E_r$. At every activation time, the time $\Delta$ needed by the packet to arrive at node $E_r$ must be added. Given that each node can be at an arbitrary distance from $E_r$, we have a different $\Delta_r^s$ for each. Finally, if the condition $\sum_{\substack{E_s \in \mathcal{E} \\ E_s \neq E_r}} h_s(t - \Delta_r^s) - \alpha t \leq 0$ is satisfied for each checking point between 0 and the hyperperiod $H$, then it is always satisfied. Therefore, the lemma follows. □

Note that the checking points $\Phi_r$ must be obtained for each receiving node $E_r$ to let the Lemma 7 holds for every node.

## V. Timing analysis

This section presents a response time analysis compatible with the system model presented in section III, which accounts for packet transmission with error recovery through retransmission. Recall that, as discussed in Section III, each job of a task $\tau_i^k$ sends $M_i$ packets during its period $T_i$, with a probability $P(e)$ of incurring a transmission error due to the communication channel. If $\tau_i^k$ does not have any packet to send, then $M_i = 0$. The response time analysis for each node consists in deriving an upper bound $\overline{R}_i^k$ on the worst-case response time $R_i^k$ for each task in the node. Then, in case $\overline{R}_i^k \leq D_i^k$ holds for each task $\tau_i^k$ in $\Gamma_k$, the task set of node $E_k$ is deemed schedulable. Then, if all task sets executing in all system nodes are schedulable, the overall system is deemed schedulable. In the following, we show how to derive a response time upper bound $\overline{R}_i^k$ for a generic task $\tau_i^k$ of a task set $\Gamma_k$.

### A. Bounding the response times

First, the following lemma provides an upper bound on the interference suffered by a task $\tau_i^k$ due to the execution of higher-priority tasks executing on the same node.

*Lemma 9:* Under rate monotonic scheduling, the interference $I_i^{k,hp}$ suffered by a job of task $\tau_i^k$ due to the execution of higher-priority tasks on the same node is bounded by

$$I_i^{k,hp} = \sum_{\tau_j \in hp(i,k)} \left\lceil \frac{R_i^k}{T_j^k} \right\rceil \left( C_j^k + \frac{M_j + \lceil M_j P(e) \rceil}{\beta} b \right),$$

where $hp(i,k)$ represents the set of higher-priority tasks of $\tau_i^k$ in $\Gamma_k$.

**Proof.** The higher-priority interference generated on a generic task $\tau_i^k$ under fixed-priority scheduling can be calculated as shown in [12] (Equation 4.18). In the proposed system model, the computation time of a task $\tau_j^l$ is composed of the worst-case execution time $C_j^k$ plus the time spent to write the $M_j$ packets into the transmission device register/memory. Taking into account the probability of error $P(e)$ of the communication channel, $\lceil M_j P(e) \rceil$ packets must be retransmitted. Hence, multiplying $M_j + \lceil M_j P(e) \rceil$ by the size of each packet $b$ yields the total amount of data to transmit for each job. Dividing the resulting amount by the memory-to-device transmission rate $\beta$, the total execution time spent to transmit the packets is obtained. The lemma follows. □

The following lemma bounds the interference generated by the ISR in charge of handling packet reception on each node.

*Lemma 10:* The interference generated on a job of $\tau_i^k$ by interrupt service routine (ISR) instances that handle arriving packets is bounded by

$$I_i^{k,\text{ISR}} = \sum_{\substack{E_s \in \mathcal{E} \\ E_s \neq E_k}} \sum_{\tau_j^s \in \Gamma_s} M_j \cdot \sigma^{\text{ISR}} \left\lceil \frac{R_i^k + \Delta_k^s}{T_j^s} \right\rceil.$$

**Proof.** Every node in $\mathcal{E}$, except for node $k$, is considered as sending packets to the tasks in node $k$. When a packet arrives, an ISR is triggered to handle the reception of that packet. An ISR instance can be considered as a higher-priority task for all tasks in the task set $\Gamma_k$. The length of the interval in which the ISR can cause interference on $\tau_i^k$ is given by the response time $R_i^k$ plus the transmission delay $\Delta_k^s$. In fact, if a task from a given node $E_s$ starts sending a packet at $t = -\Delta_k^s + \epsilon$, with $\epsilon > 0$ arbitrarily small, an additional ISR instance will interfere with $\tau_i^k$. The resulting transmission delay $\Delta_k^s$ is thus modeled as a release jitter for the purpose of performing the analysis. As a result, the number of times that the ISR interferes with $\tau_i^k$ is given by $\left\lceil \frac{R_i^k + \Delta_k^s}{T_j^s} \right\rceil$, which, multiplied by $\sigma^{\text{ISR}}$ and the number of packets $M_j$ to be sent,

and summed for each task of each edge node, gives the total interference due to ISRs. The lemma follows. □

The following lemma provides an upper bound on the blocking time experienced by a job when accessing shared resources in the NI.

*Lemma 11:* The blocking time that a job of $\tau_i^k$ can experience is bounded by

$$PT = \frac{b}{\gamma} + \frac{b}{\beta}.$$

**Proof.** If a task $\tau_j^k$ with lower priority than $\tau_i^k$ starts executing the critical section at $t = T_i^k - \epsilon$, with $\epsilon > 0$ arbitrarily small, it cannot be preempted by $\tau_i^k$ until it completes its critical section. By design of the communication mechanism, the critical section ends when a packet of size $b$ is successfully copied into the transmission queue. Hence, with a memory read rate $\gamma$ and a memory-to-device transmission rate $\beta$, the blocking time is given by $\frac{b}{\gamma} + \frac{b}{\beta}$, and the lemma follows. □

Finally, with the above lemmas in place, it is possible to calculate the response time for each task $\tau_i^k$ as the least positive fixed point of the following equation, in terms of the variable $R_i^k$:

$$R_i^k = C_i^k + \frac{M_i + \lceil M_i \cdot P(e) \rceil}{\beta} b + I_i^{k,hp} + I_i^{k,ISR} + PT, \quad (11)$$

where $C_i^k$ is the computation time of the task, and $M_i + \lceil M_i \cdot P(e) \rceil$ is the number of packets transmitted by a task instance when considering the probability of packet loss.

## VI. Experimental results

This section reports the results of an experimental evaluation of the proposed timing analysis under different system configurations. The experimental campaign adopts realistic parameters and compares the schedulability performance obtained for the proposed approach with the performance obtained by applying a response time analysis to the same system when all transmission activities are suppressed.

**Experimental setup.** The task sets considered in the experiments have been generated as follows. Given a target task set utilization $U$, and the number of tasks $n$ to be generated, the task set generator by Emberson et al. [16] was used to create $N$ task sets of $n$ tasks, such that the cumulative utilization of each task set is given by $U$. The generator was configured to randomly select integer task periods from a log-uniform distribution with range $[T^{\min}, T^{\max}]$. The generator then provides a WCET $C_i^k$ for each task, such that the ratio between the WCET and the generated period corresponds to the utilization assigned to that task. The number of packets sent by each task $M_i$ was randomly selected from a uniform distribution with range $[0, M^{\max}]$. When $M_i$ is 0, the task sends no packet. Then, the WCET $\overline{C}$ provided by the task generator is inflated by the additional execution time required for the transmission of the $M_i$ packets to obtain the final WCET for the task, i.e., $C_i^k = \overline{C}_i^k + PT_i$, where $PT_i$ is the transmission cost, which depends on the value selected for $M_i$. Finally, all tasks were assigned implicit deadlines (i.e., $D_i = T_i$) and rate-monotonic priorities (i.e., the smaller the period, the higher the priority level). Note that, in these experiments, the same task set is used for all nodes in the network, although the analysis can support different values for the WCETs of the tasks located within other nodes.

| Parameter | Value | Description |
|---|---|---|
| $N$ | 500 | Number of task sets evaluated for each configuration |
| $U$ | 0.8 | System utilization |
| $n$ | 10 | Number of tasks per edge node |
| $T^{\min}$ | 5000 | Task minimum period ($\mu s$) |
| $T^{\max}$ | 500000 | Task maximum period ($\mu s$) |
| $M^{\max}$ | 5 | Maximum number of packets sent by each instance of a task |
| $\alpha$ | 15 | NI bandwidth ($MB/s$) |
| $b$ | 16 | Number of bytes per packet |
| $Q$ | 8 | NI queue size (packets) |
| $\gamma$ | 13.24 | Minimum read/write rate to access memory ($MB/s$) |
| $\beta$ | 13.24 | Minimum read/write rate to access device registers ($MB/s$) |
| $\beta^{max}$ | 56.47 | Maximum read/write rate to access device registers ($MB/s$) |
| $\sigma^{\text{ISR}}$ | 2 | ISR overhead ($\mu s$) |
| $P(e)$ | 0.2 | Packet loss probability |
| $z$ | 10 | Number of nodes in the network |

TABLE I
NOMINAL SETTINGS OF THE PARAMETERS DEFINING THE SYSTEM CONFIGURATION.

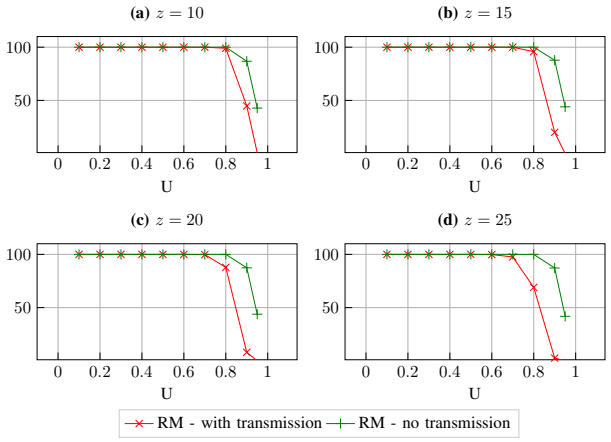


Fig. 2. Schedulability ratio (y-axis, in percentage points) as a function of the system utilization $U$ with different values of $z$.

Realistic values for parameters such as the memory access rates $\beta$, $\beta^{max}$, and $\gamma$ were derived by profiling packet transmission on the Xilinx Ultrascale+ SoC platform, with Cortex-A cores running at 1.2 GHz. In this case, 725, 170, and 170 clock cycles were obtained by profiling the respective rates. The values for the parameters $U$, $z$, and $M^{\max}$ varied across the experiments. The default values for these parameters, together with the value of the other system parameters, were set according to the system configuration in Table I.

**Experimental results.** The first set of experiments evaluated the schedulability ratio of the system concerning the system utilization generated for each node, obtained when testing $N = 500$ task sets per utilization point. The number of nodes $z$ is varied among the experiments. Figure 2 depicts the results under four different configurations. Compared to the response time analysis with suppressed transmissions, when

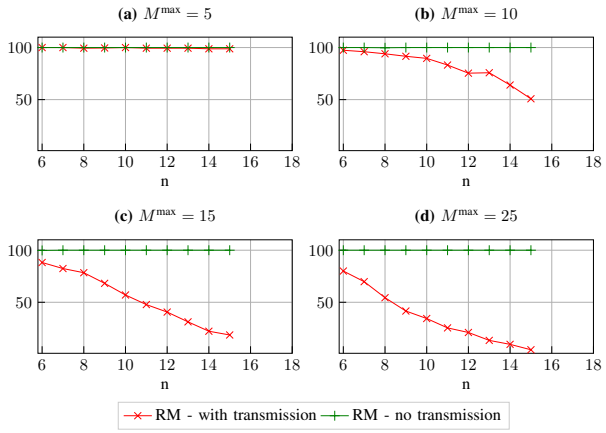


Fig. 3. Schedulability ratio (y-axis, in percentage points) as a function of the number of tasks $n$ with different values of $M^{\max}$.

the number of nodes $z$ or the utilization $U$ increases, the proposed approach exhibits a more significant drop in schedulability performance. Indeed, broadcast packet transmissions sent by each node increase the interference caused by the ISR in charge of receiving packets on each node. This behavior must be considered when designing edge applications in which nodes can join the system dynamically.

In the second set of experiments, the schedulability ratio of the system is evaluated concerning the number of tasks $n$. In this case, the system utilization is set to $U = 0.8$, while the generation parameter $M^{\max}$ is varied across the experiments to control the number of packets each task sends. Figure 3 depicts the results under four system configurations. When the number of packets sent by each task is increased, the schedulability decreases rapidly with respect to the number of tasks $n$ due to the additional amount of data to be exchanged among nodes in the network, which can lead to excessive traffic.

Overall, these experiments show that considering transmission activities when designing edge network applications is fundamental to correctly assigning system resources, e.g., when operating under a congested network or when the number of nodes in the network is expected to vary dynamically.

## VII. Conclusions and Future Work

This paper presented an analysis of the different types of latency that can be introduced by the communication interface when a task scheduled in an edge node transmits packets to another node in a sensor network. The latency was modeled and analyzed to derive necessary conditions that ensure bounded transmission and reception latencies. Then, the system schedulability is verified according to a specialized response time analysis. Modeling the time behavior of the system is crucial when functions performed by the edge-computing system needs to interact with the external environment, e.g., in a wireless sensor and actuator network involved in vehicular crowdsensing applications.

When the proposed analysis was applied, an experimental campaign was conducted to evaluate the schedulability performance obtained with different system configurations. The results show the importance of explicitly accounting for the communication activities when designing an edge computing system, primarily when it operates under a congested network or when nodes can join the network dynamically.

Future work includes introducing a fault model that considers error-recovery strategies and provides a response time analysis model for multiprocessor platforms. In addition, future work should investigate the impact of specific hardware features, such as direct memory access, and account for a more general task scheduling model, for instance, accounting for sporadic releases of the tasks.

## References

[1] L. Bisti, L. Lenzini, E. Mingozzi, and G. Stea, "Estimating the worst-case delay in fifo tandems using network calculus," in *3rd International ICST Conference on Performance Evaluation Methodologies and Tools*, (Brussels, BEL), ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 5 2010.

[2] M. Mei, M. Yao, Q. Yang, M. Qin, K. S. Kwak, and R. R. Rao, "Delay analysis of mobile edge computing using poisson cluster process modeling: A stochastic network calculus perspective," *IEEE Transactions on Communications*, vol. 70, no. 4, pp. 2532–2546, 2022.

[3] D. Sabella, A. Vaillant, P. Kuure, U. Rauschenbach, and F. Giust, "Mobile-edge computing architecture: The role of mec in the internet of things," *IEEE Consumer Electronics Magazine*, vol. 5, no. 4, pp. 84–91, 2016.

[4] J. Ren, G. Yu, Y. He, and G. Y. Li, "Collaborative cloud and edge computing for latency minimization," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 5, pp. 5031–5044, 2019.

[5] J. Ren, G. Yu, Y. Cai, and Y. He, "Latency optimization for resource allocation in mobile-edge computation offloading," *IEEE Transactions on Wireless Communications*, vol. 17, no. 8, pp. 5506–5519, 2018.

[6] M. Jia, J. Cao, and L. Yang, "Heuristic offloading of concurrent tasks for computation-intensive applications in mobile cloud computing," in *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, (USA), pp. 352–357, IEEE, 2014.

[7] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, and Q. Li, "Lavea: Latency-aware video analytics on edge computing platform," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, (New York, NY, USA), pp. 1–13, Association for Computing Machinery, 2017.

[8] M. Aamir and M. A. Zaidi, "A buffer management scheme for packet queues in manet," *Tsinghua Science and Technology*, vol. 18, no. 6, pp. 543–553, 2013.

[9] L. Sad, "Parallelising reception and transmission in queues of secondary users," *International Journal of Electrical and Computer Engineering*, vol. 9, no. 4, p. 3221, 2019.

[10] C. Lu, A. Saifullah, B. Li, M. Sha, H. Gonzalez, D. Gunatilaka, C. Wu, L. Nie, and Y. Chen, "Real-time wireless sensor-actuator networks for industrial cyber-physical systems," *Proceedings of the IEEE*, vol. 104, no. 5, pp. 1013–1024, 2016.

[11] M. Gaitán Gutiérrez and P. Yomsi Meumeu, "Multiprocessor scheduling meets the industrial wireless: A brief review," *U. Porto Journal of Engineering*, vol. 5, no. 1, pp. 59–76, 2019.

[12] G. C. Buttazzo, *Hard real-time computing systems: predictable scheduling algorithms and applications*, vol. 24. USA: Springer Publishing Company, Incorporated, 2011.

[13] D. Bertsekas and R. Gallager, *Data Networks (2nd Ed.)*. USA: Prentice-Hall, Inc., 1992.

[14] J. Kurose and K. Ross, *Computer Networking: A Top-Down Approach*. UK: Pearson Education, Limited, 2010.

[15] P. Fara, G. Serra, A. Biondi, and C. Donnarumma, "Scheduling replica voting in fixed-priority real-time systems," in *33rd Euromicro Conference on Real-Time Systems (ECRTS 2021)*, (Dagstuhl, Germany), pp. 13:1–13:21, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

[16] P. Emberson, R. Stafford, and R. Davis, "Techniques for the synthesis of multiprocessor tasksets," in *1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, pp. 6–11, July 2010.