# Wireless Sensor to Cloud Delay: A Network Hardware and Processing Perspective for Industrial Internet of Things

H. Murat Gürsu, Samuele Zoppi, Wolfgang Kellerer
Chair of Communication Networks
Technical University of Munich, Germany
Email:murat.guersu@tum.de

***Abstract*—Internet of things applications require lower and lower delay to enable different applications such as healthcare monitoring. Most of the prior work focuses on enabling this through upcoming and expensive 5G communications. Another alternative to realize a solution is the ever-existing wireless sensor networks. The real-time communication through sensors has been of interest for industrial applications and is currently available for scales down to 100 ms. When we are interested in delays of two orders of magnitude lower, low processing power of sensors and internal hardware delay becomes significant. In this work we analyse existing sensor networks for mostly neglected delay contributors in a real testbed. We demonstrate that even the existing and cheap hardware can potentially meet the real-time requirements. These results are further utilized to provide discussions for an optimized medium access control delay.**

## I. Introduction

The industrial internet of things (IIoT) are being extended to lower delay requiring applications such as patient health monitoring [1]. Such time-sensitive applications also with the scale of the sensors pose new research challenges. The main research effort focuses answering these challenges for 5G under 3GPP framework. But, another direction is to optimize the already existing IEEE 802.15.4 based devices, that we will refer to as Wireless Sensor Networks (WSN), for similar capabilities [2].

The accepted delay guarantees in WSN is mostly around 100 ms [3]. The main challenge is to enable low latency for multi-hop communication through synchronized coordination. Nevertheless, the IIoT requires tens of ms order delay which is even challenging for a single hop WSN. On the other hand it is not impossible. We believe that investigation of limitations of the standard and available implementations, can lead to cheap solutions for some use-cases of IIoT.

### A. Contribution

In this work we provide a detailed sensor to cloud delay analysis for a single hop network. Our main contribution is providing hardware and processing aspects for communication delay under IIoT scope. In order to do so we provide a general **Interface-Processing** framework in Sec. II. **A sensor to cloud delay measurement** with different sensor boards in a testbed are shared with insights in Sec. III. Finally, we relate the effect of these delays to Medium Access Control (MAC) and possible future work.

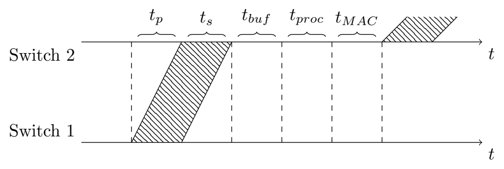

Fig. 1: The switch timings diagram for general delay calculation.

## II. Interface-Processing Framework

There are multiple contributors to the delay of a single packet that is transmitted through a wireless sensor network device. In general, the delay is defined as the time that is required from the generation of the packet, until the reception of the packet at the end point. A general framework [4] to model a layer 2 switch delay is illustrated in Fig. 1. The delay $T$ per hop is $t = t_{buf} + t_{MAC} + t_s + t_p + t_{proc}$, where $t_{buf}$ is the queuing delay, $t_{MAC}$ is the average MAC waiting delay, $t_s$ sending delay, $t_p$ is the propagation delay and $t_{proc}$ and the L2 processing delay.

The $t_s$ is well investigated in the WSN literature and $t_p$ is negligible in short distance wireless communication. But, $t_{proc}$ is not negligible for sensor networks that use i.e., 8 MHz processors, where processing delay may be in the order of 1 ms.

A *wireless sensor operation* is decomposed into **processes** $t_{proc_i}$ and **interfaces** $t_{int_i}$. Each layer functionality is defined as separate processes and the communication between each process as an interface as seen in Fig. 2 and summarized as:

- **Application Process** interrupts the other processes to read the sensor data and forwards the readings in a packet format to the communication stack through an interface called *Northbound interface*.
- **Communication Stack Process** appends headers for networking purposes to the packet and forwards it either (1) over the *SouthWbound interface* to

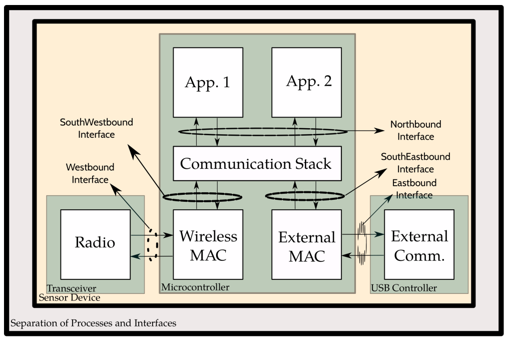

Fig. 2: A wireless sensor system on chip decomposed to interfaces for possible parallelization of a packet processing and avoiding any pipeline queuing.

the Wireless MAC process, or through the *SouthEbound interface* to the external MAC process (,e.g., USB or Ethernet).

- **Wireless MAC Process** decides the right time to forward the packet over the *Westbound interface* to the radio transceiver circuit.
- **Radio Process** converts the digital information to analog among other PHY processes over the *Wireless interface*.
- **External MAC Process** forwards the packet to the external communication circuit through *Eastbound interface*.

## III. Sensor to Cloud Delay

*1) Scenario:* In this work we consider a single hop wireless transmission from a sensor to a gateway that is forwarded over a USB connection to a cloud. To investigate the sensor to cloud delay in a practical way in the following sections, we provide measurements over two different wireless sensor set-up, with Zolertia 1 [5] and OpenMote [6]. These platform are selected for a WSN proof of concept due to their popularity. Both of the sensors support IEEE 802.15.4 PHY communication. The higher layer implementation is from OpenWSN [7] for TSCH and we adapted the for MAC layer as discussed in the Sec. IV. The transmission takes place over the non-interfered channel 26.

The delay is calculated from the generation of the packet until it reaches the cloud. The scenario is summarized in Fig 3 as a 10 process-interface chain. We will investigate the delay in three main parts: as sensor, gateway and cloud.

On the cases where our testbed network lacks the interface or the mentioned process we share the state of the art results or possible modifications as a theoretical perspective.

### A. Sensor Delay

We start with investigating the initiation of the data in the sensor.

*1) **Application Process**:* We do not want to constrain ourselves to any sensor/network application. We abstract this with general processing delay. For instance in [8] it is shown that just processing of a tactile signal can take up to 15 ms in an Arduino based set-up. Of course this depends heavily on hardware, implementation and the application type. One way to optimize this is through definition of efficient function blocks as investigated for actuators in International Electrotechnical Commission (IEC) 61499. A recent work has investigated implementation of these blocks to increase re-usability [9].

*2) **Northbound Interface**:* Even though a general standardization for IIoT northbound interface is lacking, separate communities have started working on this interface for their use-case. For instance, IEEE Standard for a Smart Transducer Interface for Sensors and Actuators Network Capable Application Processor (NCAP) Information Model IEEE 21451 introduces a Network Abstraction Logical Interface Specification. An exemplary implementation can be found in [10] where dynamic implementation of sensor applications are enabled.

*3) **Communication Stack Process(1)**:* The high layer processing of the packet is composed of transport layer and networking layer In our specific example we consider a UDP packet.

In order to investigate the effect of the payload size, we vary it from 1 to a maximum of 77 bytes. As the maximum packet size is 133 bytes in IEEE 802.15.4, we have to take into account the 56 Bytes added on NET, MAC and PHY layers. In Fig. 4 the y-axis depicts delay in (ms) while the x-axis shows the varying payload in bytes.

The delay in on average around 0.94 ms for Z1 and 0.36 for OM and a small overhead is added with increasing payload that is $39\,\mu$s for Z1 and $0.39\,\mu$s for OM. The delay difference is not directly reflected by the clock speed of the microprocessor that is 16 MHz for Z1 and 32 MHz for OM.

The results demonstrate the performance of the transport plus networking stack from OpenWSN and could be optimized for specific scenarios.

*4) **SouthWbound Interface**:* The Wireless MAC can be implemented on a separate microcontroller to get rid off delay due to processing. However, this separation to get rid of the processing queuing delay will translate into an interface delay. In our exemplary set-up we do not have this interface.

In state of the art similar interface delays can be observed in [11] where a MAC accelerator is implemented on FPGA and the added interface delay can be clearly observed. In [12] a multi-radio and multi-processor gateway is implemented such that communication stack has to use a bus based interface to separate the packets to each MAC process. The separation of processes results in a fixed delay overhead. Thus, even though the interfaces come with a delay cost it also provides determinism and thus should be the selected solution for real-time requiring applications.

*5) **Wireless MAC Process**:* Our aim here is to extract main processes such as selecting the wireless carrier,

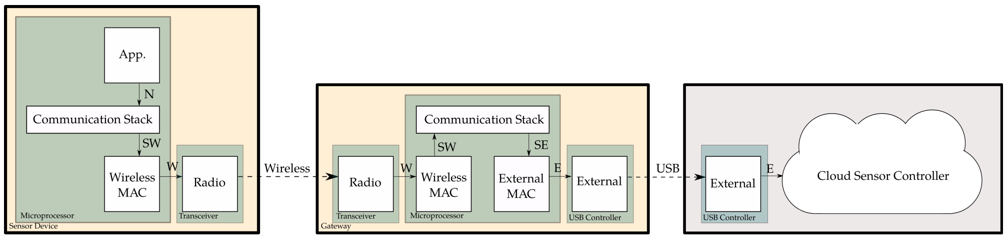

Fig. 3: The sensor to cloud delay with processes and interfaces. The scenario depicted here is a one-hop, one-user communication. Without loss of generality this can be extended to many.

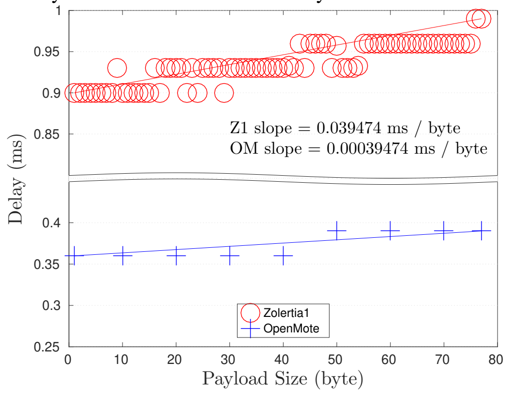

Fig. 4: Communication Stack Processing delay on OpenWSN for UDP packet creation until forwarding to the MAC layer. The processing time is intrapolated with the line plots and the measurements are given with markers.

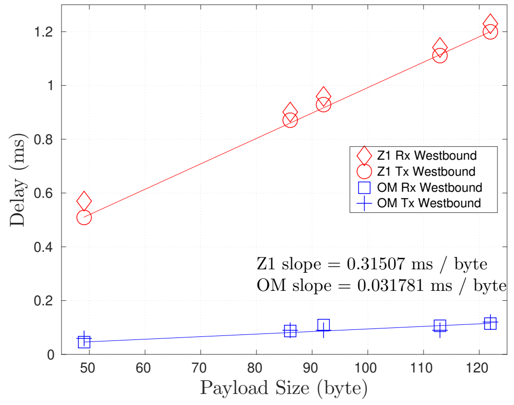

Fig. 5: Westbound transmission and reception delay

duplicating packet for re-transmissions and measure the delay impact for OpenWSN.

For Z1 the Rx Processing delay is 330 $\mu$s and the Tx Processing delay is 210 $\mu$s. For OM, the Rx Processing delay is 180 $\mu$s which is almost half due to increased processing speed. However, the TX Processing delay is 30 $\mu$s, as the packet duplication benefits from the sharing the same memory between wireless chip and the microcontroller. The Wireless MAC reception delay includes the cyclic redundancy check for error detection and thus higher in both sensors. *Wireless MAC* delay did not change with changing the packet size so it stems mostly due to fixed header operations.

*6)* ***Westbound Interface****:* The radio process is usually implemented on a separate chip than the MAC processes. This is due to distinctive delay requirements of the two processes. Thus, how this interface between these two process is implemented can create a bottleneck in terms of minimum delay.

The OM has the CC2538 chip that is composed of the radio chip and the ARM microcontroller. The Z1 has the CC2520 chip and a separate microcontroller MSP430. For Z1 the inter-radio-microcontroller communication is handled via Serial Peripheral Interface (SPI) protocol. The radio chip and the microcontroller has separate memories and the packet has to be copied between to memory of each chip, before the chip can access it. The CC2538 solves this problem by allowing the radio chip to read the memory of the microcontroller.

The measured interface delay is evaluated In Fig. 5 with x-axis depicting the packet size varies from 50 bytes to 122 bytes and y-axis depicting the delay in ms. With a a payload of 120 bytes for the Z1 the delay surpasses 1.2 ms while for OM it is at maximum 150 $\mu$s. This result, as reflected with an order of magnitude difference in delay, showcases the importance of interface delay.

*7)* ***Radio Process****:* The radio chip transmission process delay is fixed to 0.6 ms in both sensors with all payload sizes. This includes the radio turn on time and is also non-negligible and necessary for long battery life for sensors.

*8)* ***Wireless Interface****:* The transmission delay incorporates the data-rate provided by the transceiver and the packet size. A packet of 50 byte that is transmitted with a 250 kbps results in 1.6 ms delay and increases to 4.2 ms with maximum payload.

## *B. Gateway Delay*

There radio chip reception delay is negligible with varying payload. The Westbound Interface delay is almost same as the Tx delay. The Wireless MAC, SW interface and Communication Stack is discussed in Sec. III-A5 and in not repeated here.

*1)* ***SouthEbound Interface****:* In case the external interface needs to be heavily used, as would be the case in a gateway, the handling of the external interface MAC on a separate microcontroller is necessary. This is commonly used in Wi-Fi routers that has separate processors for dealing with backbone data forwarding. The separation of these processes impose an interface delay that is investigated in state of the art [12].

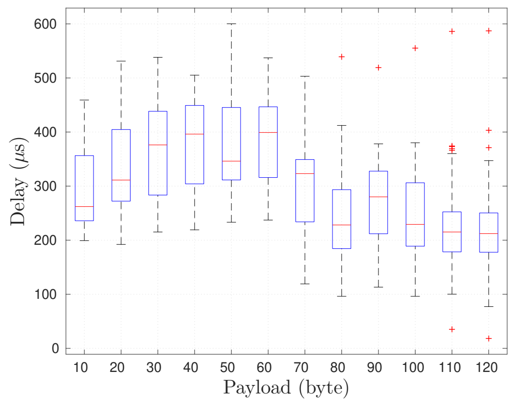

Fig. 6: USB chip processing delay combined for PC and Sensor

*2) **External MAC Process**:* The external MAC process is responsible for setting timings and creating the UART packet. The UART packet is transferred to the External Interface chip. We have investigated the OpenWSN, external MAC process implementation that is the OpenBridge. The resulting delay is below $30\,\mu$s thus negligible.

*3) **Eastbound Interface**:* On the Eastbound interface, the microcontroller sends UART messages to the CP2102 chip and the chip receives and converts these to USB messages and sends it to the connected interface with the USB 2.0 protocol. The UART protocol is simple and it is limited to the baudrate settings of the chip. The protocol is not protected with any error detection code so increasing the baudrate may increase error probabilities. For this reason we set the baudrate to 115200 bauds per second. This results in 11.02 ms delay for 127 bytes and 4.42 ms for 51 bytes.

*4) **USB Process**:* The Z1 chip contains the CP2102 that has an embedded $156\,\mu$s timeout to form a USB packet out of the UART reception. This timeout based packet forming can add additional delay and it can vary from 0 to $156\,\mu$s. We did not measure this separately but we observed the variation caused by it. The OM on the other hand has FT232R USB UART IC that is connected through FTDI that has a 16 ms timeout which creates a delay bottleneck.

In order to characterize the chip processing delay we measured the end-to-end delay of a packet transmitted from the PC to the sensor and back. We used an echo application at the sensor. The driver at the PC acknowledges the USB packet after the transmission and this can be sniffed through Wireshark. We can measure the time from the transmission of the packet until its acknowledgement, this includes one-way transmission. If we subtract this value and the UART delay from the end-to-end delay measurement, we get the USB chip transmission and the PC processing delay since the echo application at the sensor takes negligible time. The resulting delay in $\mu$s with varying payload is illustrated in Fig. 6. Interestingly, the delay decreases with increased packet size, which is unexpected. We project that this

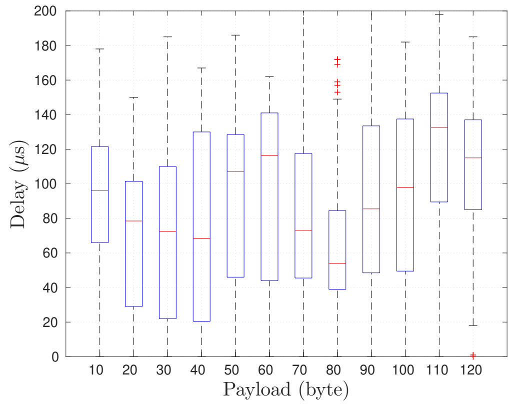

Fig. 7: Cloud USB delay due to polling slots of USB 2.0

is stemming from faster UART transmission speed than advertised.

## C. Cloud

The cloud, the end point for the data collection, is abstracted here as an edge cloud that provides low delay high processing power capabilities to the sensor applications.

*1) **Cloud USB Process**:* The USB controller chip we have on the laptop that is an Intel Corporation 8 Series/C220 Series Chipset Family USB EHCI. It runs USB 2.0 specification [13]. There is also processing delays stemming from this chip. We have measured the delay by sniffing the USB message exchange on the chip and there is a consistent $10\,\mu$s delay between the packet reception on the chip and the ACK transmission to the USB chip on the sensor.

*2) **Cloud Eastbound Interface**:* The USB interface is controlled via a bus controller that implements a polling based communication. The polling slots can be decreased to $125\,\mu$s for USB 2.0. The polling slots are for various devices connected to the bus. The generation time of the packet can result in a waiting time if it does not match the polling slot. To measure this waiting time 10000 packets are generated and forwarded to the USB controller over the bus. The measured minimum delay out of all the measurements is selected as the *lucky* packet that is generated at the exact instance of a polling slot. Thus, the minimum delay only consists of the constant processing delay. We removed this value from all measurements to reflect only the delay due to waiting time of polling slots as shown in Fig. 7. We see that the delay varies between 200 $\mu$s and 0 such that the bus controller has a polling slot each 2 slots for the USB communication

*3) **Cloud Sensor Controller Process**:* Here we assume that there is an edge cloud placed as close to the gateway as possible. Thus, no further variable delay is induced due to IP network. From this point on there is also no hardware or interface delay but purely processing delay. However, this processing delay can also surmount to noticeable levels. In [14] authors provide an FPGA

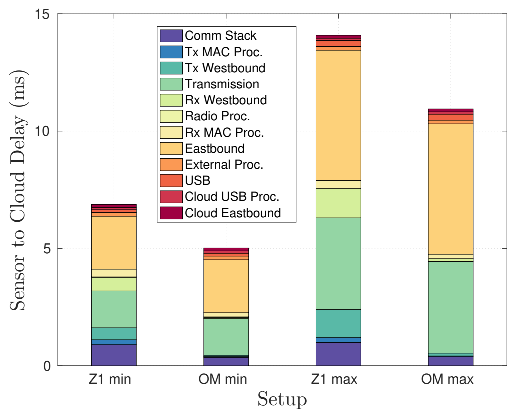

Fig. 8: Sensor to Cloud delay with two different sensor network types Zolertia 1 and OpenMote. The packet size is also selected as 46 and 122 Bytes. The delay contributors are ordered in the sequential way the packet experiences them.

implementation for a industrial sensor controller for parallel processing of multiple sensors at the same time that reduces the delay 90 ms for 10 sensor applications. Thus, a fast parallelized cloud controller can be necessary for real-time requiring sensor applications.

### D. *Total Sensor to Cloud Delay*

In this section we want to provide the total sensor-to-cloud delay analysis as illustrated in the Fig. 8 with stacked delay values for different sensors (Z1, OM) and with different payload (46, 122 bytes).

Firstly, the eastbound delay is the biggest delay contributor followed by the wireless transmission delay. The magnitude of the eastbound delay stems from the limited baudrate which is the serial communication between *the micro-controller* and *the UART to USB chip* [1]. IoT Gateways should consider implementing faster interfaces to solve such problems. With decreasing packet size, the importance of the processing delay grows, as most processes are adding a fixed delay to all packets. Another observation is that the PHY headers take up to 6 Bytes and all the other layers occupy as much as 45 Bytes and this is another problem especially for IPv6 compatibility as it is the scope of OpenWSN to enable 6LoWPAN for WSNs.

The delay figures represented in Fig. 8 is a minimum sensor to cloud delay if no loss is observed and the schedule is adjusted perfectly. Previous work [15], [17] have investigated the effect of packet loss probabilities on single hop networks thus we will emphasize the scheduling aspect.

## IV. Discussions: Wireless Medium Access Control

In order to emphasize on synchronization and determinism we will consider a TDMA based contention free MAC protocols.

[1]The OpenMote actually has the USB chip FT232R which comes with 16 ms timeout to form packets, this is note illustrated in the plot due to ease of visualization.

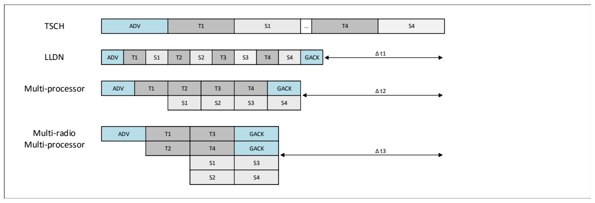

Fig. 9: Different MAC schemes considering also the processing time

We consider 4 possible schedules: TSCH, LLDN, multi-processor gateway and a multi-processor multi-radio gateway as illustrated in Fig. 9. The downlink slots are illustrated in cyan, the uplink slots are illustrated in dark grey and the serial slots are illustrated in light grey. The scenario illustrates a slotframe with 4 uplink users.

We start with investigating the TSCH, where 1 slot is required for advertisement, 4 slots for transmitting messages from 4 sensors and 4 slots for the serial communication. In TSCH each slot is of size 9.372 and 5.52 ms for Z1 and OM respectively. Resulting in a slotframe duration of 84.34 and 46.98 ms for Z1 and OM respectively. Each user can transmit 11.7 and 21.2 packets per second with this slotframe duration.

In TSCH an ACK is sent directly within each slot. However, in LLDN, instead of unicast ACKs a broadcast GACK slot is used such that the slot size is decreased to 2.85 and 1.94 ms respectively for Z1 and OM. This gives a slotframe duration of 28.5 and 19.4 ms respectively for Z1 and OM. The reduction in frame size results in 35 and 52 packets per second almost tripling the rate. LLDN implements the group acknowledgements in a star topology, to benefit from radio switch times for each sensor, which explains the improvement from TSCH. It also strips the higher layer headers and leaves this problem to the gateway for backbone forwarding.

Packet rate can be doubled if a different processor is used for external forwarding or if a faster interface is used on the gateway processor. Then a *SouthEBound* delay $s_1$ may be added to each slot to move the data from one processor to another and serial forwarding and wireless reception is parallelized.

On top of this a multi-radio gateway can enable parallel reception of data from multiple sensor to decrease the number of slots waited until another packet is transmitted. This is enabled with multiple processors that communicates with multiple radios. This also comes with the cost of *SouthWBound* delay $s_2$ that is added to the slots. An important reminder is the number of orthogonal wireless channels is limited to 16 in IEEE 802.15.4 standard and this imposes a maximum number of parallel wireless transmission.

## V. Related Work

Any type of delay analysis has been an interesting topic for WSN. However, an analysis that encapsulates all sensor-delay related details and measurements for data to cloud integration is missing.

There are several works that investigated the topic from different aspects. An application layer perspective to sensor to cloud delay is taken in [19]. The authors have taken Constrained Application Protocol (CoAP) based measurements and provided delay insights that investigates application protocol inefficiency. This solution has taken an *on the top* approach. Another work [20] investigates the totally opposite directions for task scheduling on sensor networks. The multi-threading possibilities for delay reduction is investigated. So neither of the works provides the communication stack related approach. The former deals with the application layer and the latter is general processing.

A more communication stack focusing approach is taken for a real-time operation system for WSN communications in [21]. The abstraction layers provide resources to a real time operation system RTOS. This system runs separate communication processes such that they interact in a real-time basis. The use of RTOS enables *right-in-time* interaction between different processes. The function abstraction enables also multi-processor support. The multi-processor communication over an inter-processor communication (IPC) interface. However, any detail about this important interface, such that which communication protocol should be used, is not given. There, UART protocol is used, but clearly this is only feasible for a two processor scenario. Communication layer interfaces are also defined and message exchanges are pre-set. However, variety of delays in the interfaces are not emphasized.

## VI. Conclusion

In this paper we have analyzed the IIoT suitability of the IEEE 802.15.4 standard namely WSN compliant off the shelf sensors. We investigated a scenario that takes a IIoT sensor to cloud delay under investigation. We have provided a *Interface-Processing* framework to investigate the sub-problems separately.

We have shown that there are a lot of aspects that are overlooked that can add significant delay for a IIoT application. Lastly we have tackled MAC delay given the previous processing and interface delays and forecasted possible solutions that incorporated such delays for scalable applications.

Future work can focus on solving these problems to provide a IIoT enabled WSN. An additional natural extension is to have a Mobile Edge application to showcase low delay and message sending rate to support IIoT applications.

## References

[1] J. H. Abawajy and M. M. Hassan, "Federated internet of things and cloud computing pervasive patient health monitoring system," *IEEE Communications Magazine*, vol. 55, no. 1, pp. 48–53, 2017.

[2] P. Gallotti, A. Raposo, and L. Soares, "v-glove: A 3d virtual touch interface," in *2011 XIII Symposium on Virtual Reality*, pp. 242–251, May 2011.

[3] C. Wu, D. Gunatilaka, M. Sha, and C. Lu, "Real-time wireless routing for industrial internet of things," in *2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pp. 261–266, IEEE, 2018.

[4] V. Ballingam, K. Christensen, and F. Noel, "Analysis of client/server transaction delay through a local area network switch," in *Proceedings of SOUTHEASTCON'96*, pp. 571–577, IEEE, 1996.

[5] W. Zolertia, "platform, z1 datasheet."

[6] X. Vilajosana, P. Tuset, T. Watteyne, and K. Pister, "Openmote: open-source prototyping platform for the industrial iot," in *International Conference on Ad Hoc Networks*, pp. 211–222, Springer, 2015.

[7] T. Watteyne, X. Vilajosana, B. Kerkez, F. Chraim, K. Weekly, Q. Wang, S. Glaser, and K. Pister, "Openwsn: a standards-based low-power wireless development environment," *Transactions on Emerging Telecommunications Technologies*, vol. 23, no. 5, pp. 480–493, 2012.

[8] M. Aziziaghdam and E. Samur, "Real-time contact sensory feedback for upper limb robotic prostheses," *IEEE/ASME Transactions on Mechatronics*, vol. 22, pp. 1786–1795, Aug 2017.

[9] D. Kleyko, E. Osipov, S. Patil, V. Vyatkin, and Z. Pang, "On methodology of implementing distributed function block applications using tinyos wsn nodes," in *Emerging Technology and Factory Automation (ETFA), 2014 IEEE*, pp. 1–7, IEEE, 2014.

[10] J. A. Guevara, E. A. Vargas, A. F. Fatecha, and F. Barrero, "Dynamically reconfigurable wsn node based on iso/iec/ieee 21451 teds," *IEEE Sensors Journal*, vol. 15, pp. 2567–2576, May 2015.

[11] T. Gomes, S. Pinto, F. Salgado, A. Tavares, and J. Cabral, "Building ieee 802.15. 4 accelerators for heterogeneous wireless sensor nodes," *IEEE Sensors Letters*, vol. 1, no. 1, pp. 1–4, 2017.

[12] M. Kohvakka, T. Arpinen, M. Hännikäinen, and T. D. Hämäläinen, "High-performance multi-radio wsn platform," in *Proceedings of the 2nd international workshop on Multi-hop ad hoc networks: from theory to reality*, pp. 95–97, ACM, 2006.

[13] U. Inter-Chip, "Supplement to the usb 2.0 specification," *The Universal Serial Bus Revision*, vol. 2.

[14] Q. Chi, H. Yan, C. Zhang, Z. Pang, and L. Da Xu, "A reconfigurable smart sensor interface for industrial wsn in iot environment," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 2, pp. 1417–1425, 2014.

[15] M. Gürsu, M. Vilgelm, S. Zoppi, and W. Kellerer, "Reliable co-existence of 802.15. 4e tsch-based wsn and wi-fi in an aircraft cabin," in *Communications Workshops (ICC), 2016 IEEE International Conference on*, pp. 663–668, IEEE, 2016.

[16] M. Vilgelm, M. Gürsu, S. Zoppi, and W. Kellerer, "Time slotted channel hopping for smart metering: Measurements and analysis of medium access," in *Smart Grid Communications (SmartGridComm), 2016 IEEE International Conference on*, pp. 109–115, IEEE, 2016.

[17] S. Zoppi, H. M. Gürsu, M. Vilgelm, and W. Kellerer, "Reliable hopping sequence design for highly interfered wireless sensor networks," in *Local and Metropolitan Area Networks (LANMAN), 2017 IEEE International Symposium on*, pp. 1–7, IEEE, 2017.

[18] M. Gürsu, M. Vilgelm, W. Kellerer, and E. Fazlı, "A wireless technology assessment for reliable communication in aircraft," in *Wireless for Space and Extreme Environments (WiSEE), 2015 IEEE International Conference on*, pp. 1–6, IEEE, 2015.

[19] C. Pereira, A. Pinto, D. Ferreira, and A. Aguiar, "Experimental characterization of mobile iot application latency," *IEEE Internet of Things Journal*, vol. 4, pp. 1082–1094, Aug 2017.

[20] N. Ericsson, T. Lennvall, J. Åkerberg, and M. Björkman, "A flexible communication stack design for time sensitive embedded systems," in *Industrial Technology (ICIT), 2017 IEEE International Conference on*, pp. 1112–1117, IEEE, 2017.

[21] Z. Pang, K. Yu, J. Åkerberg, and M. Gidlund, "An rtos-based architecture for industrial wireless sensor network stacks with multi-processor support," in *Industrial Technology (ICIT), 2013 IEEE International Conference on*, pp. 1216–1221, IEEE, 2013.