Real-Time Emulation and Evaluation of Large-Scale WirelessHART Networks: A Single HIL Transceiver Approach

Raymond J. Jayabal, *Member, IEEE,* Xiaojuan Zhang, *Member, IEEE*

***Abstract*—In this paper, we present our OMNEST-based WirelessHART Hardware-In-The-Loop (WH-HIL) network emulation platform. Leveraging on protocol constraints, it is able to emulate up to 240 motes communicating in real-time with a physical IEC 62591 compliant Gateway (GW) with just a PC connected to a single transceiver via USB, replacing racks upon racks of actual motes normally used for stress-testing. We present key design aspects and protocol considerations that made this implementation possible. Providing various performance metrics, we foresee this low-cost platform will be useful in understanding how WirelessHART networks behave in various deployment layouts, aiding in education as well as in the development of better routing, link scheduling and fault management algorithms. Operating in real-time, it may also serve as a crucial step towards cost-effective realization of digital twins for WirelessHART networks.**

***Index Terms*—Industrial wireless networks, WirelessHART, real-time emulation, hardware-in-the-loop, digital twins.**

I. Introduction

THE deployment of large-scale WirelessHART networks has become increasingly prevalent. The routing and link scheduling algorithms which have been instrumental to its proliferation in the industrial wireless communications arena are now being stretched to their real-world limitations. As in-field testing is difficult and costly, a software platform capable of emulating such deployments in real-time would not only aid in troubleshooting algorithmic issues but also those caused by device constraints such as memory size, computing power and real-time task scheduling performance. Immediate applications for such a system would also include interoperability and pre-deployment stress testing of the GW where the routing and link scheduling decisions are made. Currently such systems involve stacks of actual motes that are not only costly but also cumbersome to set up and maintain. In the long term, this emulation platform could serve as a vital component in developing digital twins for industrial processes employing WirelessHART networks.

With these goals in mind, we managed to develop just such a system: using a standard Windows10 PC running a modified OMNEST network simulation software [1], and connected to a single Texas Instruments (TI) CC2652 multiprotocol 2.4GHz transceiver module [2], we were able to emulate up to 240 WirelessHART motes communicating with an actual GW in real-time over long periods of time. Each virtual mote in the OMNEST simulator runs a fully IEC 62591 compliant Softing stack [3], exactly replicating the protocol and timing behaviors of an operational WirelessHART field device. To be able to multiplex so many virtual motes unto a single HILtransceiver while ensuring that the packets they transmit or receive are in exactly the same over-the-air slots scheduled by their individual Medium Access Control (MAC) layers, the design of the platform had to leverage on the protocol's slot timing and link scheduling constraints. To the best of our knowledge, our WH-HIL network emulation platform is the first of its kind not only in terms of the scale of multiplexing unto a single HIL transceiver, but also in utilising knowledge of the WirelessHART link scheduling constraints. Thus it constitutes the main contribution of our paper.

The rest of our paper is organized as follows: Section II briefly reviews the innovations in advancing WirelessHART network emulation. Section III presents the protocol analyses and key design aspects of the platform. Section IV covers its physical setup, configuration, operating procedure, and presents select performance evaluation results. Finally, section V recaps its current benefits and outlines future plans for it.

R. J. Jayabal and X. J. Zhang are with Institute for Infocomm Research (I²R), A*STAR, Singapore e-mail: {jraymond, xzhang}@ i2r.a-star.edu.sg.

II. Related Works

Early efforts on virtualizing WirelessHART networks have been at the software level on established simulators like Matlab, NS2/3 and Opnet [4,5,6], with just a basic set of the WirelessHART protocol running non real-time. Subsequently, researchers began integrating the HIL concept into WirelessHART simulations (WH-HIL) [7,8]. In these cases, not only are the simulated units very few, but the units being simulated are not the motes or GW but rather the plant controllers. In [9], a HIL-based large scale wireless network simulator was presented, but the network is entirely simulated in software with just one instance of the wireless node connected to an emulated automotive control system. In contrast, our platform exercises the full WirelessHART protocol over RF signals with an actual GW, as if a large number of real-world motes are communicating with it. With this virtual network of motes thus validated, it becomes easier and cost effective to perform accurate evaluations of our own WirelessHART-related GW algorithms [10-11].

III. SYSTEM DESIGN

A. Overall System Architecture

The overall system comprises three entities: the OMNET simulator, the CC2652 HIL transceiver, and the WirelessHART

GW (Fig. 1). The interface between the simulator and the HIL is a USB UART cable running at 961 Kbps, while the HIL and the GW exchange IEEE 802.15.4 PHY-compliant radio frames via an RF cable. Our work was limited to the simulator and the HIL transceiver, designing, implementing and integrating the components that make up their final software architecture. Of these, the key components are the field device protocol stack (Stack), the virtual PHY interface (PhyIf) and the HIL-synchronized future-event-set (FES) scheduler (RtSched) in the simulator, and the lightweight MAC (LiteMac) on the HIL transceiver.

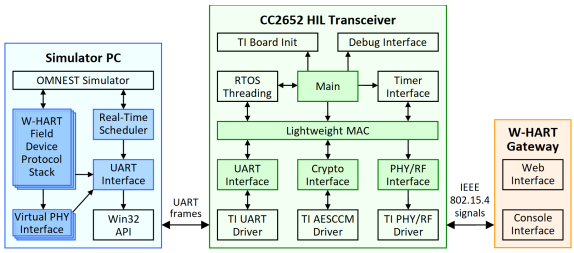


Fig. 1. Overall system architecture.

The Stack component is an OMNEST/C++ port of the Softing WirelessHART field device protocol stack software [3] originally targeted to the CC2652 radio module. It comprises the standard WirelessHART Application, Network, Data Link and MAC layers as well as a Field Device entity which provides device information and process variables for burst transmissions. Each instance of this stack emulates a WirelessHART mote as it scans for Advertisements, establishes slot synchronization with the GW, exchange MAC Protocol Data Units (MPDUs) with it while joining the network, sending Advertisements and KeepAlives, receiving and responding to the GW's HART commands, etc. Based on its link schedules, the MAC layer of each Stack instance sends a TXREQ or RXREQ command to the singleton PhyIf component to transmit or receive an MPDU in a given slot and channel. Within PhyIf, decisions are made whether to discard, send the MPDU virtually to the MAC layers of other virtual motes, and/or convey the request to the HIL's LiteMac layer, based on mote-to-mote received signal strength (RSS) and path stability matrices as well as network-wide link scheduling constraints.

The RtSched component on the other hand is a FES scheduler that is time-synchronized to the HIL's time base. Drift compensation was necessary to ensure that the timeslot synchronization maintained by the Stack instances remain as close as possible to the HIL's actual timeslots, especially over long periods when no transmission or reception takes place. RtSched is also where RXIND messages bearing received MPDUs from the HIL are time-advanced before being added to the OMNEST's "Future Event Set" (FES) event queue to be dispatched to the MAC layers of the Stack instances via the OMNEST event dispatching framework. This is to ensure that the TXREQ/RXREQ commands sent by the Stack instances arrive at the HIL in time for the actual deadlines within the slot (Fig. 2).

LiteMac performs slot and Absolute Slot Number (ASN) synchronization with the GW, and conveys transmitted/ received MPDUs between the GW and the virtual motes in the simulator. For MPDU reception, due to the interframe timing constraint, LiteMac sends the acknowledgement (ACK) frame to the GW on behalf of the recipient mote. LiteMac also performs scanning, reception and sending of Advertisement frames to the simulator.

The rest of this section discusses the protocol analyses that underly the design of these key components.

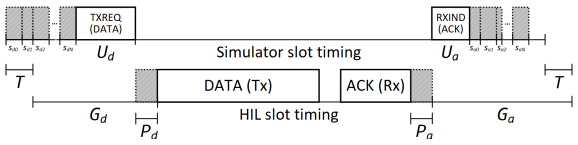


Fig. 2. Simulator-HIL slot timing constraints.

B. Slot Timing Considerations

To ensure that DATA frames from the virtual motes arrive before their actual transmission deadline on the HIL transceiver, their timeslots have to start earlier relative to the HIL's. This timing-advance, along with the maximum number of virtual motes that can be supported for a given simulator-HIL interface speed, can be derived by analysing the simulator and HIL processing and communication delays relative to the MAC protocol's transmission timing constraints (Fig. 2). Although the analysis here is specific to the WirelessHART's TDMA MAC protocol which has the DATA-ACK transaction within the slot, it should be possible to carry out similar analyses for other MAC schemes.

From the figure, the timing constraints before the DATA and after the ACK transmission are:

$$\max\left(\sum_{i}^{N} s_{d_i}\right) + U_d \leq G_d - P_d + T \tag{1}$$

$$\max\left(\sum_{i}^{N} s_{a_i}\right) + U_a \leq G_a - P_a + T \tag{2}$$

where N is the number of virtual motes, s_{d_i} and s_{a_i} the simulator processing durations for the i-th virtual mote before the DATA and after the ACK frame, U_d and U_a the simulator-HIL interface transmission duration for the maximum-length DATA and ACK frames, G_d and G_a the minimum gaps before and after the over-the-air transmission of the DATA and ACK frames, P_d and P_a the maximum PHY processing delays prior to transmitting the DATA and after receiving the ACK frame, and T the timing-advance to be applied. Combining Eqs.(1,2) and approximating U_d and U_a to be directly proportional to the bit lengths L_d and L_a of their corresponding frames we get

$$\frac{L_d + L_a}{B} \leq (G_d + G_a) - (P_d + P_a) - \left(\max\left(\sum_{i}^{N} s_{d_i}\right) + \max\left(\sum_{i}^{N} s_{a_i}\right)\right). \tag{3}$$

The values for s_{d_i} and s_{a_i} are bounded within a finite range, so the means (μ_{s_d} and μ_{s_a}) and variances ($\sigma^2_{s_d}$ and $\sigma^2_{s_a}$) of their distributions are likely to be finite. As the motes are

identical, s_{d_i} and s_{a_i} are also each likely to be identically distributed. As a simplification, we will further assume that they are also independently distributed. This will allow us to apply the central limit theorem to approximate the maximums of their sums to the values at a desired tail-end percentile α of the corresponding Gaussian distributions:

$$\frac{L_d + L_a}{B} \leq (G_d + G_a) - (P_d + P_a) - \left(N(\mu_{s_d} + \mu_{s_a}) + Z(\alpha)\sqrt{N}(\mu_{s_d} + \mu_{s_a})\right). \quad (4)$$

Here, $Z(\alpha)$ is the Z-score corresponding to the α percentile. With this equation, we can now work out N and T for our platform: Over the UART interface, the maximum TXREQ(DATA) and RXIND(ACK) frame lengths inclusive of the UART headers are 134 and 53 bytes, respectively. The UART speed itself is limited to 961Kbps. From the WirelessHART slot timing specifications, $G_d = 1.960ms$ and $G_a = 1.784ms$. On the CC2652, the PHY processing delays never exceed $0.3ms$. In the simulator, processing durations for a single mote before the DATA and after the ACK frame never exceed $10us$ and $2us$, respectively. Assuming $s_{d,i}$ and s_{a_i} to be uniformly distributed, then $\mu_{s_{d/a}} = \max(s_{d/a})/2$ and $\sigma_{s_{d/a}} = \max(s_{d/a})/\sqrt{12}$. Taking the 99.7-percentiles ($Z(0.997) = 2.75$) as the maximums for their sums, Eq.(4) yields $N = 239$ while Eqs.(1-2) give us $773us \leq T \leq 779us$.

C. Link Scheduling Considerations

At any one time, only one TXREQ or RXREQ should ever be forwarded to the HIL. However, multiple motes may send TX/RXREQs for DATA and ACK frames concurrently in the simulator. For ACK frames, the HIL's LiteMac already handles them automatically, leaving us with only the task of identifying which DATA TX/RXREQs should be forwarded. To figure this out, we had to identify each scenario where these contentions occur and formulate the appropriate remedy.

The first of these occurs in the unslotted mode of operation in the "searching" state which a mote enters after booting up or resetting due to an irretrievable error. In this mode, motes cyclically send an RXREQ on each available channel independent of slot timings, waiting to receive an Advertisement from the GW or potential parent nodes. Since boot/reset times vary, multiple motes may send RXREQs for different channels in a time-overlapped manner, making it physically impossible for the single HIL transceiver to conform while also handling TXREQ/RXREQs from motes in the slotted mode. To cater to this scenario, a background process is added to the HIL's LiteMac to synchronize it with the GW's timeslot and search for channels and timeslots, in which the GW cyclically sends Advertisement messages. With this Advertisement link schedule determined, LiteMac automatically activates the HIL's receiver on each of the scheduled slots to receive an Advertisement message and send it to the simulator via the Simulator-HIL RXIND message, which also specifies the channel and reception start/end times over which it was received. In the simulator, RtSched duplicates and dispatches the RXIND to each mote's PhyIf, which then forwards the Advertisement message it bears to its MAC layer if it previously received an RXREQ with matching channel and reception start/end times. Thus, RXREQs from motes in the unslotted mode are never forwarded to the HIL.

The second scenario occurs in the slotted mode of operation, which a mote enters after synchronising its timeslots to the GW's based on the reception start time of the Advertisements it received. In this mode, multiple nodes may send simultaneous TXREQs that clash, when they send their first message over shared upstream links using random backoff in the "negotiating" state. For this scenario, an inter-node RSS matrix and a message-in-message (MIM) radio capture simulation is used to resolve which mote's TXREQ (if any) is forwarded to the HIL.

The third and final contention scenario we identified typically occurs in the "quarantined" or "operational" states, where a mote within range of the GW issues a TX/RXREQ to send/receive a packet to/from the GW on one channel, while another mote also within range of the GW issues a TX/RXREQ to send/receive a packet to/from a child node on a different channel. To resolve this case, we need to distinguish which of the corresponding links are for the motes' upstream or downstream traffic. For the SmartMesh WirelessHART system [12], this is possible given that for each bandwidth profile, different superframe lengths are assigned to different traffic directions. Since the GW only listens over the motes' upstream links and transmits over their downstream links, we forward only TXREQs for upstream links and RXREQs for downstream links to the HIL.

Taking all these scenarios into account, a TX/RXREQ arbitration process (Algorithm 1) is designed and integrated into the PhyIf module in the simulator. It comprises four procedures: init() which is called when the mote's Stack initializes its PhyIf, txReq()/rxReq() when it instructs its PhyIf to start receiving/transmitting, and txReqCapTout() which is called when the capture timer set in txReq() expires.

IV. Setup and Evaluation Results

Fig. 3 shows the physical setup of the platform in our lab. As mentioned earlier, the 3 entities are the PC, the CC2652 and the GW. To run this platform, a user first edits the OMNEST simulator configuration file (omnetpp.ini) to configure the simulation duration and network to be simulated. The latter is done either by specifying a JavaScript Object Notation (JSON) file containing path stability and RSS data collected from a real-world deployment, or specifying parameters such as the number of first hop motes, the fan-out factor, and the maximum number of hops to a flexible tree-topology generator which can generate variations of synthetic star, linear, single-hop, and multi-hop networks to evaluate the GW performance in well-defined topologies. Additionally, the data burst rate and few protocol parameters such as the Advertisement wait and join timeout can also be configured.

As the simulation runs, the OMNEST IDE's console window is updated with trace logs of the change in the emulated motes' join states, whereas on the GW's web browser interface and nwconsole command window, more information regarding the motes' states and activities can be observed (Fig. 4).

Algorithm 1 Simulator-HIL TX/RXREQ Arbitration

Input: Inter-node RSS and frame success rate (FSR) matrices, GW's upstream and downstream superframe lengths, GW's RX sensitivity, capture threshold and capture timeout, multiple TX/RXREQs submitted via tx/rxReq() procedures.
Output: One TXREQ and/or RXREQ sent to the HIL.

1: **procedure** INIT():
2: Init ASNs for previous completed TX/RXREQ and current TXREQ arbitrations to 0.
3: Init GW's captured transmission's rx power to 0, aggregate noise (uncaptured signals) power to sensitivity level and captured transmission's TXREQ payload to none.
4: **procedure** RXREQ(): ▷ Input: Node's index, slotted/unslotted mode, intraslot DATA/ACK phase and link's superframe length, node's RXREQ's absolute slot number and message.
5: Exit if RXREQ arbitration is already done for the current ASN, or node is in unslotted mode, or in ACK phase, or not using a downstream link, or not possible at all to receive a transmission from the GW based on the FSR matrix.
6: Otherwise, forward this RXREQ to the HIL and set the ASN for the previously completed RXREQ arbitration to this RXREQ's.
7: **procedure** TXREQ(): ▷ Input: Node's index, current intraslot DATA/ACK phase and link's superframe length, node's TXREQ's absolute slot number and message.
8: Exit if TXREQ arbitration is already done for the current ASN, or node is in ACK phase, or not using an upstream link, or not possible for GW to receive a transmission from this mote based on the FSR matrix.
9: Set ASN for current TXREQ arbitration to this RXREQ's.
10: Set timer to call txReqCapTout() after capture timeout if not already running.
11: Calculate GW's received power for this node's transmission based on the RSS matrix.
12: If this transmission is strong enough to be captured by the GW, add its previously captured transmission's receive power to its noise power, add its captured transmission's received power to this transmission's and set its captured transmission's TXREQ message to this transmission's.
13: Else if the previously captured signal is strong enough to be retained, add this transmission's received power to the GW's noise power.
14: Else if neither this nor the previously captured transmission is strong enough for capture/retention, add both transmissions' received powers to the GW's noise power, Set GW's captured transmission's received power to 0 and Set GW's captured transmission's TXREQ to none.
15: **procedure** TXREQCAPTOUT():
16: If there is a captured transmission, forward the captured transmission's TXREQ to the HIL.
17: Set ASN for the previous completed TXREQ's arbitration to the current arbitration's, set the GW's captured transmission's received power to 0, set its aggregate noise (uncaptured signals) power to sensitivity level and its captured transmission's TXREQ message to none.

After the simulation ends, various statistical data which were recorded during the simulation within the OMNEST Stack module can be used within the OMNEST IDE's built-in Analysis Editor to plot and examine network performance related charts. Due to space constraint, we present only a select few of these based on 4 network topologies we emulated: 1) a 200-mote 1-hop topology where all motes and GW hear each other (1Hop200); 2) a 250-mote 2-hop tree topology with 50 first hop motes and fan-out factor of 4, where child motes

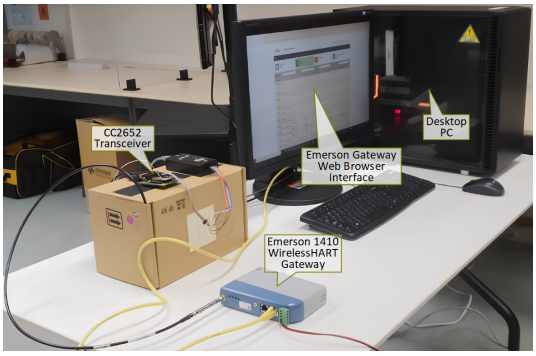


Fig. 3. Physical setup of the WH-HIL platform.

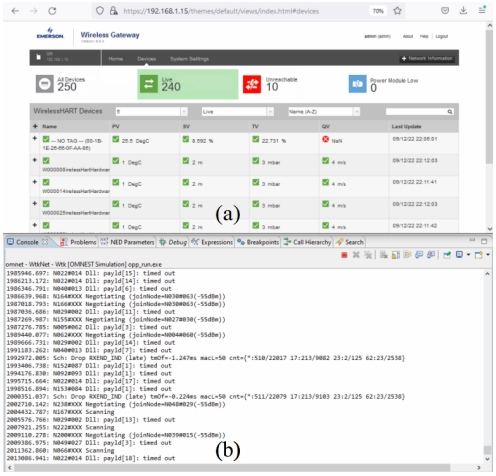


Fig. 4. Monitoring the emulated motes' states and activities via the GW web interface (a) and OMNEST console (b).

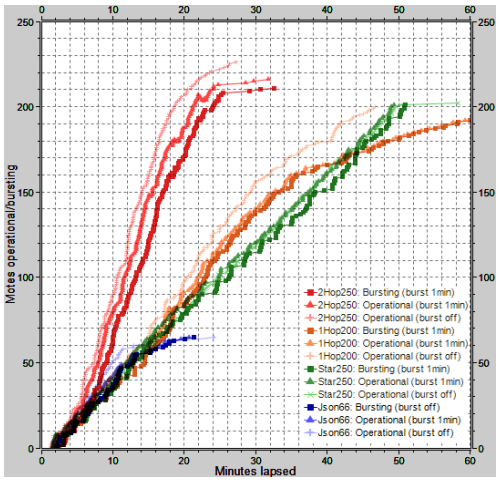


Fig. 5. Example motes operational and bursting time-series chart.

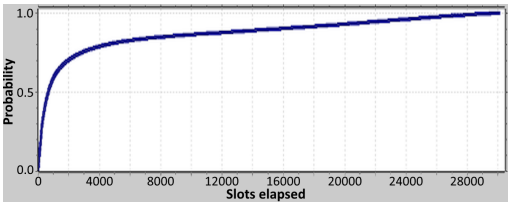


Fig. 6. Example burst latency CDF (Json66).

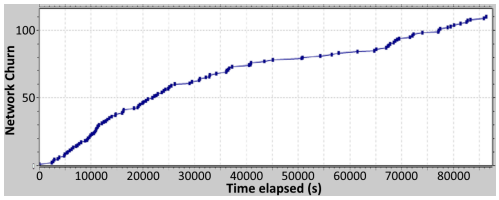


Fig. 7. Network churn time-series (Json66).

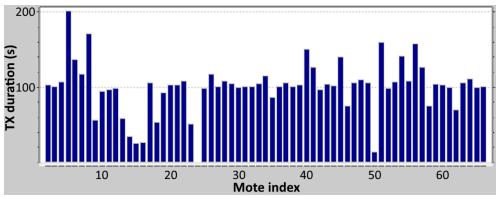


Fig. 8. TX durations histogram (Json66).

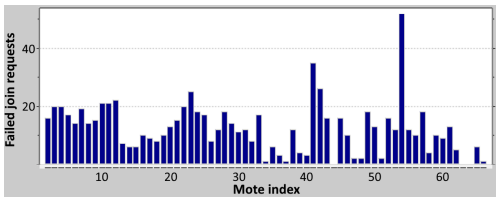


Fig. 9. Join failures histogram (Json66).

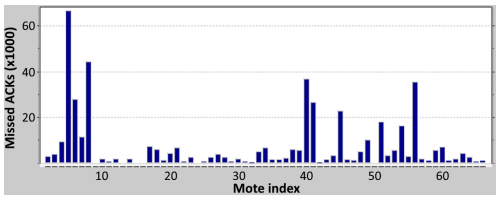


Fig. 10. Missed ACKs histogram (Json66).

and their parent mote are able to hear each other but child motes themselves cannot hear each other (2Hop250); 3) a star topology where motes and GW hear each other, but motes themselves cannot hear each other (Star250); and finally, a 66-mote topology based on real-world RSS & path stability data contained in a Json file (Json66).

An example chart among these is the speed at which motes achieve operational and bursting state (Fig. 5). Bursting at 1 packet/min, we see that the 2Hop250 topology is fastest at this, with over 200 done within 30 minutes. The 1Hop200 and Star250 topology achieves a far second and third best performance, with about 140 and 115 motes done within 30 minutes, respectively. One reason for this discrepancy could be that for the 2Hop250 topology, the first hop motes are able to act as proxies for the join procedure thus speeding up the joining process. As for the Json66 topology, all 65 motes are operational and bursting within 22 minutes, which is just at slightly slower rate than the 1Hop200 and Star250 topologies. Without bursting, the speed at becoming operational is slightly faster for all topologies as expected, since there are less packets contending for air-time.

Other example charts that are available for analysis are the burst latency cumulative density function (CDF) (Fig. 6), network churn time-series (Fig. 7), total transmit/receive duration histogram which is useful for evaluating power usage (Fig. 8), the join failures histogram (Fig. 9) and neighbour missed ACKs histogram (Fig. 10). With the APIs provided by the OMNEST framework, more such statistics can be added with ease as needed.

V. Conclusion and Future Work

We have presented our WH-HIL platform in this paper. Taking advantage of underlying protocol constraints, its ability to emulate a large number WirelessHART motes communicating in real-time with a physical GW with just a single transceiver marks a significant advancement in the field. With this platform, various performance metrics can be produced to gain insights into the real-time behavior of large-scale WirelessHART networks, making it a useful tool for researching into better algorithms for mote and GW as well as a potentially important component for digital twinning. Currently it is limited to 240 virtual motes mainly due to its OMNEST components running on single thread. With multithreading, larger number of motes is likely supportable. Adding radio emulation to the current path stability and RSSI simulation would also greatly improve its emulation accuracy.

Acknowledgment

This work was sponsored by Emerson Asia Pacific Pte Ltd with additional technical support from Emerson Automation Solutions.

References

[1] A. Varga, “The OMNeT++ discrete event simulation system”, in Proc. European Simulation Multiconf. (ESM), pp.1-7, Jun 2001.

[2] Texas Instruments, “CC2652R SimpleLink™ Multiprotocol 2.4 GHz Wireless MCU”, https://www.ti.com/lit/ds/symlink/cc2652r.pdf, last accessed Jan 2024.

[3] Softing Inc., “WirelessHART: Communication stack opens new possibilities”, https://industrial.softing.com/uploads/$softing_downloads$ /$IN_HTZ_Witeck_Stack_etz$_2012_06_EN.pdf, last accessed Jan 2024.

[4] K. Shah, T. Seceleanu, M. Gidlund, “Design and implementation of a WirelessHART simulator for process control”, in Proc. Intl. Symp. on Industrial Embedded System (SIES), pp.221–24, Jul 2010.

[5] G. Gao, H. Zhang, L. Li, “An OPNET-based simulation approach for the deployment of WirelessHART”, In Proc. 9th IEEE Int. Conf. on Fuzzy Systems and Knowledge Discovery, pp.2120-4, May 2012.

[6] P. Zand, E. Mathews, P. Havinga, S. Stojanovski, E. Sisinni, P. Ferrari, “Implementation of WirelessHART in the ns-2 simulator and validation of its correctness”, Sensors, 14(5), pp.8633-68, May 2014.

[7] S.M. Hassan, R. Ibrahim, N. Saad, V.S. Asirvadam, K. Bingi, “Adopting setpoint weighting strategy for WirelessHART networked control systems characterised by stochastic delay. IEEE Access, 5, pp.25885–96, Nov 2017.

[8] C.D. Tran, R. Ibrahim, V.S. Asirvadam, N. Saad, H.S. Miya, “Internal model control for industrial wireless plant using WirelessHART hardware-in-the-loop simulator”, ISA Trans., 75, pp.236–46. Apr 2018.

[9] D.S. Buse, M. Schettler, N. Kothe, P. Reinold, C. Sommer, F. Dressler, “Bridging worlds: Integrating hardware-in-the-loop testing with large-scale VANET simulation”, Proc. 14th Annual Conf. on Wireless On-demand Network Systems and Services (WONS), pp.33-6, Feb 2018.

[10] Q. Chen, X.J. Zhang, W.L. Lim, Y.S. Kwok, S. Sun, “High reliability, low latency and cost effective network planning for industrial wireless mesh networks”, IEEE/ACM Trans. Networking, 27(6), pp.2354-62, Oct 2019.

[11] Q. Chen, X.J. Zhang, W.L. Lim, Y.S. Kwok, S. Sun, “Automated network planning for industrial wireless mesh networks”, Proc. IEEE Global Communications Conf. (GLOBECOM). pp.1-6, Dec 2018.

[12] Linear Technology, “SmartMesh WirelessHART Manager API Guide”, Retrieved: https://www.analog.com/media/en/reference-design-documentation/design-notes/$smartmesh_wirelesshart_manager_api_guide$.pdf, Mar 2024.