Realization and Measurements of Industrial Wireless Sensor and Actuator Networks

Kan Yu[1] and Johan Åkerberg[2] and Mikael Gidlund[3] and Mats Björkman[1]

Abstract— **Industrial automation can benefit from applying wireless sensor and actuator networks (WSAN) on cost reduction, mobility and flexibility. However, wireless solutions are more prone to interferences compared to wired ones. In order to avoid production losses and to keep the revenues at an anticipated level, it is of utmost importance for WSANs to meet the stringent requirements from industrial automation, such as high reliability and real-time performance. A great number of research efforts were taken in this field based on simulations, but simulation results may not show sufficient confidence. Existing implementations and products compatible with the standards may still fail to provide reliable and real-time communication. Therefore, in this paper we built a prototype of industrial wireless sensor and actuator networks (IWSAN) and implemented a protocol stack, aiming for providing reliable and real-time communication for mission-critical industrial applications. Afterwards, we deployed our prototype and conducted measurements in real industrial environments. Our measurement results exhibited possibility of applying IWSANs for industrial applications and brought more evidence to our industry.**

I. Introduction

Nowadays wireless technologies for usage in industrial automation has been an important trend. It is known that wired industrial systems not only demand expensive cables, but also involve costly device installation and maintenance. As wireless technologies evolve, wireless systems have been applied in the industrial domain at the form of industrial wireless sensor and actuator networks (IWSAN). IWSANs have been exhibiting its advantages over traditional wired counterparts in cost saving and access to mobility.

Although IWSANs can bring numerous benefits to industrial automation, their applicability in industrial environments is still in the development stage. The majority of traditional wireless sensor networks (WSN) still focus on non-deterministic communication and low power consumption. However, for many mission-critical industrial applications, requirements are quite different and more stringent [1], since failures in mission-critical applications might cause catastrophic consequences, such as severe financial losses and safety accidents. Thus, among those requirements, high reliability and real-time performance are of most importance. Moreover, IWSANs are foreseen to be deployed in harsh industrial environments, which are considered to be dusty, humid, full of metallic equipment with high temperature and vibrations. To deal with these interferences, appropriate strategies should be developed, such as time division multiple access (TDMA) schemes and multipath transmission methods. Therefore, designing and realizing such IWSANs pose numerous challenges.

In order to verify IWSAN solutions, simulation is insufficient for several reasons. Firstly, some solutions are designed under certain assumptions which are not realistic. Secondly, wireless channel settings in simulators are very different from the real world. Thirdly, an IWSAN solution may be overcomplicated to be implemented in practical resource constrained devices. Nowadays, a great number of research efforts based on IEEE 802.15.4 have been taken in this area, but only a few of them are realized and verified in practice [2][3][4][5]. However, these implementations and realizations still either fail to concentrate on reliable and real time communication or fail to meet industrial setups. Moreover, several IWSAN standards were also published, such as WirelessHART [6], ISA 100.11a [7] and WIA-PA [8]. Up to now, there are a number of standards-compliant products available in the market. For instance, Emerson and Siemens released several products based on WirelessHART; the companies like Nivis also released their products using ISA 100.11a. However, these existing products may still fail to meet the strict requirements from some mission-critical industrial applications. For instance, authors in [9] deployed commercial implementations of WirelessHART in industrial environments and revealed that round trip time in some locations was too long, especially for downlink transmission.

To meet the stringent requirements from industrial automation and show the possibility of applying IWSANs for mission-critical industrial applications, we built an IWSAN prototype including the protocol stack for reliable and real time communication. Majority of the prototype is built on a resource limited embedded platform, which leads to challenges to implement the complete protocol stack. TDMA mechanism is applied for deterministic packet delivery, and a flooding-based routing protocol is implemented to improve reliability and real-time performance. Then we deployed our prototype in a real industrial environment for testing. The measurement results showed the confidence on providing both reliable wireless sensing and actuating for industrial automation systems.

The rest of this paper is organized as follows: Section II we give an overview of IWSAN. The prototype implementation is introduced in Section III in detail. In Section Iv, the initial measurement in a real industrial environment is described. Finally, the paper is concluded in Section V.

[1]Kan Yu and Mats Björkman are with the School of Innovation, Design and Engineering Malardalen University, Sweden

[2]Johan Åkerberg is with ABB AB Corporate Research, Sweden

[3]Mikael Gidlund is with the Department of Electrical Engineering, Wright State University, Sweden

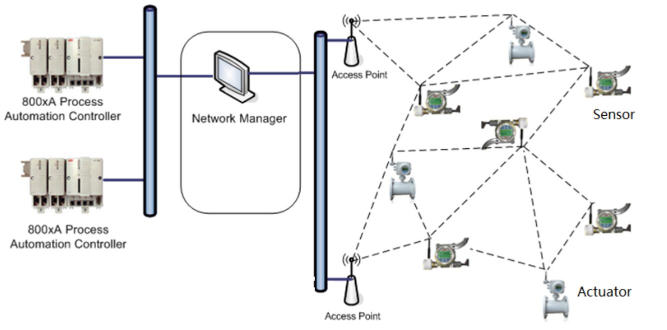
800xA Process Automation Controller
800xA Process Automation Controller
Network Manager
Access Point
Access Point
Sensor
Actuator

Fig. 1. Example of an IWSN structure

II. Overview of Industrial Wireless Sensor and Actuator Networks

IWSANs emerge as a new generation of WSNs to serve for both monitoring and control for industrial purposes. In this section, the architecture of IWSAN is described and the requirements from industries is introduced.

A. IWSAN Architecture

Centralized management is often applied in IWSANs instead of self-organization, since operators in the central control room must have full knowledge of the status of all devices in the network. A general structure of an IWSAN is shown in Figure 1. Various types of devices are defined in different standards. As shown in Figure 1, four types of devices are involved: 1) *Network manager*: responsible for managing the activities of the network, including scheduling, routing and time synchronization management; 2) *Access point*: to bridge the wireless part to the control system; 3) *Sensor node*: responsible for monitoring various types of status; 4) *Actuator node*: attached to the process plant and perform basic functions of actuating.

In IWSANs, sensor nodes periodically send data to the control system via access points. After the collected information being processed, the corresponding commands are delivered to actuator nodes for controlling. Therefore, both uplink and downlink transmissions are equally important in IWSANs.

B. IWSAN Requirements

Unlike traditional WSNs, industrial applications served by IWSANs can be grouped into three categories : 1) monitoring: different types sensors provide diagnostics and supervision, which are updated periodically; 2) closed loop control: industrial processes are stabilized by the controlling the actuators based on the sensor readings; 3) interlocking and control: many industrial control applications require discrete signaling with different interlocks. Authors in [1] summarized a number of industrial applications in each categories. According to their summary, several stringent requirements from industrial automation can be abstracted, such as energy consumption and security. In this work, we focus on the following requirements:

1) Reliability: In IWSANs, packets should be reliably transferred to their destinations to assure proper functioning. Since packets are transmitted over time-varying and error-prone wireless medium, it is critical to guarantee to packet losses within a tolerant range to avoid application failures.

2) Real-time: Many mission-critical industrial applications have hard deadlines. Successive outdated packet delivery may also lead to production outages. Moreover, it is unnecessary to guarantee delivery of all transmissions if a packet is outdated.

3) Network Size: According to [1], many industrial applications have very fast refresh rate, in the order of seconds or milliseconds. Therefore, large network size of an IWSAN can hardly be supported; otherwise, network resources are not sufficient for all transmissions.

Therefore, in this paper, we intend to build a prototype to fulfill those requirements mentioned above and provide more confidence to apply IWSANs for industrial automation.

III. Prototype Implementation

In this section, the detailed prototype implementation is described. Our prototype implementation is composed of a hardware part and a software part. In the hardware part the hardware platforms will be stated, while in the software part the protocol stack implementation method will be presented.

A. Hardware Platform

In our prototype, two types of hardware platforms are used to build different devices. For simplicity, we merge the network manager and access point as one part. Since a sensor node and an actuator node are similar regarding to wireless communication, one hardware platform is used for both of them. In order to monitor wireless communication behaviors, we also built a sniffer based on the same hardware as a sensor/actuator node. Therefore, the first platform, named the radio board, is used to build sensors, actuators and the sniffer. Since this hardware platform includes radio interfaces, we also used this platform to build the radio part of the network manager. However, due to the constrained memory resource in the first platform, we added the second hardware platform, named the management board, to build the upper layers of the network manager for the complicated operation, such as scheduling and routing calculation. The detailed information about the hardware platforms is introduced as follows.

B. Radio board

The first platform is STM32W-SK from ST Microelectronics shown in Figure 2. It integrates a 32-bit ARM Cortex M3 microprocessor STM32W108CC. This processor has the following important features:

- 2.4 GHz IEEE 802.15.4 transceiver and lower MAC
- 256-Kbyte Flash and 16-Kbyte RAM memory
- Normal mode link budget up to 102 dB; configurable up to 107 dB
- -99 dBm normal RX sensitivity; configurable to -100 dBm

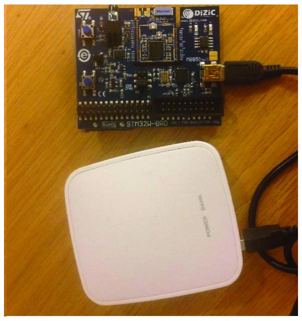
Fig. 2. The platform STM32W108CC

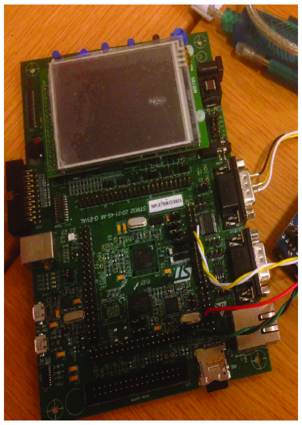
Fig. 3. The platform STM3240G

- +3 dB normal mode output power; configurable up to +8 dBm

In our prototype, we use the external crystal and configure the processor running at 24 MHz frequency for faster processing speed and better timer accuracy. The link budget and RX sensitivity are set to the default values. Since there is no interface for external antenna, we have to use the on-board ceramic antenna. In order to achieve the maximum transmission range, we set the RF output power to be +8 dBm.

1) Management board: We noticed that STM32W108CC only has 16KByte of SRAM. Different from a sensor or actuator node, more functions should be implemented in the network manager, such as network management, TDMA scheduler, etc., so we need to add an additional platform for those advanced functions. Therefore, the second hardware platform is STM3240G also from ST Microelectronics shown in Figure 2, which has the following important features:

- contains a STM32F407 high-performance ARM Cortex-M4F 32-bit microcontroller
- 16 Mbit SRAM and extensible interfaces
- RS-232 communication model

C. Interconnection of two boards

In our network manager implementation, the physical layer and the datalink layer are implemented in the radio board. The remaining layers of the stack including the application

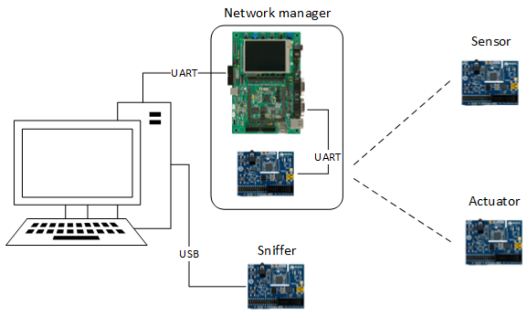

Fig. 4. The prototype connection

layer and the network layer are allocated in STM3240G. Since both boards support serial communication, we used UART interface to interconnect two platforms. In order to reduced the CPU load, we applied the direct memory access (DMA) scheme for the UART communication. Due to the limited memory size for buffering packets on the board STM3240G and real-time communication requirement, we set the baud rate of UART to be 921600 Bps. After testing, the external communication across two boards has no obvious interference to the overall performance.

1) Power supply: The platform STM32W108CC has two methods for the power supply. One is battery powering. Since the board has a battery holder, it can be powered by two normal AAA batteries. The other is via USB interface. When the radio board is used as a sensor or actuator node, in order to achieve a longer operating time, we use an external battery pack as the power supply and connect it with the board via a USB cable, which is also shown in Figure 2. When this board is used as the sniffer or the radio part of the network manager, we used a jumper to obtain power from outside. Different from the power supply for the board STM32W108CC, we use an external power supply for the board STM3240G due to the much more intensive running tasks.

2) Data collection: In order to observe the current network status, we need to visualize all information from sensors, actuators and the network manager. Due to the centralized architecture, all information can be obtained from the network manager. Therefore, we connect a PC to the network manager via the UART interface, and let the network manager forward all necessary information to the PC. Since we use the sniffer to observe the wireless communication, we also use the USB cable to connect the sniffer to the PC. Finally, the prototype connection is summarized in Figure 4.

D. Protocol Stack Implementation

We use a real time operation system (RTOS)-based architecture to implement our protocol stack. This architecture offers significant benefits such as platform independent, system integration and performance scalability. This architecture consists of two major components: 1) the stack core; 2) the platform abstraction layer (PAL), which is shown in Figure 5. The stack core is the main body of the protocol which is platform-independent. It also comprises two parts: data engine and protocol engine. The data engine copes with data

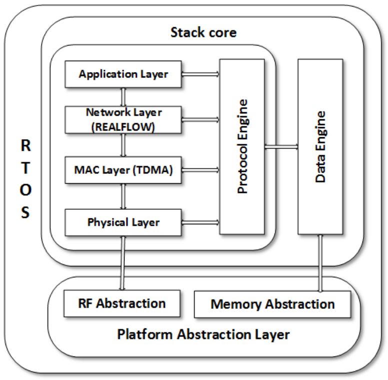

Fig. 5. The RTOS-based architecture for the protocol stack

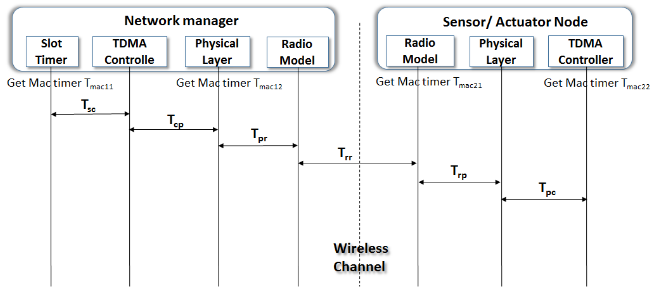

Fig. 6. Time synchronization process

management, while the protocol engine takes the responsibility of both layer management and inter-layer communication. The PAL is an abstraction of the implementation platform including the memory management and radio transceiver. By separating the PAL from the stack core, the stack core is isolated from the implementation platform. Therefore, it is efficient to port a protocol stack from one platform to another without re-implementing the whole stack.

In our stack architecture, each layer in the stack core has at least one dedicated thread and one dedicated mailbox. All inter-layer interactions are through this mailbox and is managed by the RTOS. The upper or lower layer is distinguished by the thread priority from the RTOS. Higher priority is assigned to lower layers. To support flexible partitioning of different layers, global variables are avoided. The only method for cross-layer communication is through mailboxes from the RTOS. Four basic layers are implemented in our prototype including the physical layer, datalink layer, network layer and application layer. Since both two boards are used in designing the network manager, another serialization layer is added in the network manager. The detailed information of the protocol stack implementation is introduced as follows.

1) Operating system: The RTOS we used in both platforms is PowerPac from IAR, which is designed as an embedded OS for the development of real-time applications. The footprint of the kernel in our prototype is approximately 3000 bytes used in ROM and 52 bytes used in RAM. We used semaphores, mailboxes, and events for different tasks or threads to communicate with each other with the delay less than microsecond. We implement the PAL by re-packaging the APIs of the PowerPac libraries according to the specification.

2) Radio interface: To transmit and receive packet from the radio interface, the physical layer uses the APIs of radio library provided by ST Microelectronics. The callback functions from the library will announce the physical layer about the success of sending and receiving packets. To distinguish our own packets from those of other wireless systems, we re-defined our own physical preamble. To avoid corrupted packets, we enabled the automatic CRC inserting and checking from the RF library.

3) Time synchronization: TDMA mechanism is applied on the datalink layer to provide deterministic communication. As we know that IEEE 1588 has been widely used for time synchronization in industrial Ethernets. Although the precision of our synchronization method is less than IEEE 1588, the complexity of our method is also much less. In our approach, to precisely maintain the local timer for the time synchronization, two threads are used on the datalink layer: one is for exchanging information with other layers; the other is for controlling the time. The duration of one timeslot is configured to be 10 ms, which is the same as the WirelessHART standard.

We set the network manager as the source of the slot timer, so every node in the network tries to synchronize with the network manager. The time synchronization process is summarized in Figure 6. According to the figure, once an interrupt of the slot timer in the network manager happens, the RTOS issues a signal to the TDMA controller to activate this thread from a waiting state. Then the TDMA controller looks up the local scheduling decision to check if the current slot is for transmitting. If it is, the TDMA controller thread asks for data from the datalink layer main thread and forwards it to the physical layer. Then the physical layer inserts the physical preamble and sends to the radio interface for transmission. When a node receive this packet from the gateway, the packet is processed in a similar way. All time differences between each model are defined in Figure 6 as well. Therefore, the total time difference T_{total} between the network manager and the node is:

$$T_{total} = T_{sc} + T_{cp} + T_{pr} + T_{rr} + T_{rp} + T_{pc} \qquad (1)$$

In order to obtain total time difference for accurate time synchronization, we used the MAC timer from the STM32W108CC platform. We utilized the timer values at four points during time synchronization. When the slot timer in the network manager triggers, we obtain T_{mac11} from the MAC time indicating the starting point of a time slot in the gateway. Before this packet being sent to the radio interface, we obtain T_{mac12} from the MAC timer. Then we can compute:

$$T_{mac12} - T_{mac11} = T_{sc} + T_{cp} \qquad (2)$$

According to Figure 6, T_{rr} is the packet transmission time. We assure the packet lenght is L bits. Since the data rate of IEEE 802.15.4 is 250 Kb/s, we can obtain:

$$T_{rr} = L/250 * 1000 \tag{3}$$

Once the packet is received by a sensor/actuator node, we are able to obtain the timer value T_{mac21} from the radio model indicating the exact packet receiving time. When the TDMA controller thread obtains this packet, the last timer value T_{mac22} is needed. Then we can calculate that:

$$T_{mac22} - T_{mac21} = T_{rp} + T_{pc} \tag{4}$$

It is obvious that the time difference T_{pr} cannot be obtained. Since the RTOS is used and T_{pr} equals function calling time, we skip this value during the time synchronization. The value $T_{mac12} - T_{mac11}$ is inserted into the packet before sending out. Finally, the total time difference T_{total} is:

$$T_{total}{=}T_{mac12}{-}T_{mac11}{+}L/250{*}1000{+}T_{mac22}{-}T_{mac21} \tag{5}$$

When the receiver obtains this value, it can adjust its own local timer to synchronize with the network manager. Furthermore, to avoid timeslot instability, any node should only synchronize with the parent node who is the closest one to the network manager.

4) REALFLOW Implementation: To provide reliable and real-time communication, we applied REALFLOW routing protocol proposed in [10]. In order to make the REALFLOW protocol fit for real IWSANs, we adapted this protocol with several updates.

In the REALFLOW routing protocol, in order to explore the network topology and establish packet forwarding paths, list-updated messages are sent out periodically. In our implementation, we added an additional parameter s_{gw} in a list-updated message as the route updating sequence number. Every time when a new list-updated message is generated by the network manager, this value will be increased by one, which can assist nodes with identifying a new round of network maintenance.

After list-updated messages being propagated through the whole network, all sensor and actuator nodes send back list-response messages to the network manager. Before a sensor or actuator node sends out this message, it needs to choose its parent nodes $\mathbb{N}_{parent}$. The maximum allowed number of parent nodes is defined as K_{max} ($K_{max} \geq 1$). A larger K_{max} indicates more transmission paths to the destination, but at the cost of network resources. Therefore, there is a trade-off between reliability and network efficiency when choosing K_{max} values.

Several important parameters are included in a list-response message. These parameters are: 1) the selected parent nodes $\mathbb{N}_{src}$; 2) the previous forwarding node address A_{fwd}; 3) the next hop node addresses $\mathbb{N}_{fwd}$. The purpose of list-response messages is not only for network management, but also used for routing tables generation. Routing tables in REALFLOW are named *related node lists* $\mathbb{L}$. With the help of $\mathbb{L}$, data can be directionally flooded from the source to its destination in an efficient way.

During our implementation, we noticed that it is better to deliver management packet based on unicasting, rather than flooding, since the datalink layer acknowledgement can be applied for retransmission. Thus, additional routing information is needed to forward unicasting packets. In order to make the least changes, related node lists are used for helping management packet delivery. For uplink unicasting delivery, when an intermediate node receives a management packet, if the source address of this packet is seen in its related node list, this intermediate node can just use one of its parent node address as the next hop address. However, the previous REALFLOW protocol lacks the support for the downlink management packet delivery. Therefore, we added *downlink node lists* $\mathbb{N}_{down}$ in each node. We assume that there are i number of nodes in a related node list. $\mathbb{N}_{down}$ is defined as $\{\{A_{scr1}, A_{fwd1}\}, \{A_{scr2}, A_{fwd2}\}, ..., \{A_{scri}, A_{fwdi}\}\}$, where A_{scri} is the source addresses of i-th related node. So when an intermediate node receives a packet from the node i, it records its source address A_{scri} and A_{fwdi} into $\mathbb{N}_{down}$. Finally, the command forwarding procedure is summarized in Algorithm 1, where A_{dst} is the destination address of a packet and A_{nmg} is the network manager address.

Algorithm 1 Command Forwarding Procedure

```
 1: if $A_{dst} = A_{nmg}$ then
 2:     if $A_{src} \in \mathbb{L}$ then
 3:         pick a address from $\mathbb{N}_{parent}$ as the next hop address
 4:     else
 5:         drop this command packet
 6:     end if
 7: else
 8:     if $A_{dst} \in \mathbb{L}$ then
 9:         pick a address from $\mathbb{N}_{down}$ as the next hop address
10:     else
11:         drop this command packet
12:     end if
13: end if
```

After related node lists $\mathbb{L}$ and downlink node lists $\mathbb{N}_{down}$ being respectively generated in each intermediate node, REALFLOW is able to forward both data packets and management packets. The rest of REALFLOW implementation strictly follows the description in [10], so more details of REALFLOW can be found in this previous work.

5) Timeslot Management: Since the TDMA scheme is applied on the datalink layer, timeslots need to be scheduled for all nodes for conflict-free communication. The TDMA scheduling algorithm used in our prototype is totally based on the previous work [11]. Once the scheduling decision is made, the network manager needs to distribute the new TDMA scheduling decisions to all nodes. Different from simulation, it is a great challenge for real implementation to deliver the scheduling decisions to all nodes in a distributed manner and make all nodes agree on the new scheduling decisions at the same time. Finally, we chose to apply Two-Phase Commit (2PC) method for the scheduling decision distribution in our prototype.

There are two phases in 2PC, commit request phase and completion phase respectively. During the first phase, the network manager sends scheduling request messages to all nodes. Each scheduling request message contains the latest scheduling decision, as well as the scheduling ID to indicate the scheduling decision version. When a node receives a scheduling request message, the latest TDMA scheduling decision is obtained, but not activated. The node will reply a scheduling response message to the network manager to inform of the acceptance of the scheduling decision. If the network manager fails to get responses from any node, 2PC is considered to fail and the whole process will restart. Once the network manager receives all responses, it will send scheduling confirming messages to all nodes. The scheduling confirming messages contain the time to indicate when new scheduling being applied. After receiving the scheduling confirming messages, all nodes will activate the latest scheduling at the same time. In order to reduce the complexity of 2PC, all nodes do not need to send back a final acknowledgment to the network manager. Because all nodes will report their scheduling ID in the list-response messages, the network manager is able to check the correctness of the scheduling at this point. If any node reports an outdated scheduling version, the network manager will restart the whole route maintenance.

IV. Measurement in Industry

After the prototype was built, we deployed our prototype in a real industrial environment for initial measurements. In this section, the measurement setups are described and the measurement results are analyzed.

A. Measurement Setup

Our measurement was conducted in a low voltage production manufacturing workshop during working hours. This industrial environment was full of concrete obstacles and metallic equipment. Since our measurement was taken during the working hours, there were a number of forklifts and workers moving around. Moreover, there are also other wireless systems using the same frequency band, such as WLAN. Before our measurement, we used Commview for WiFi to measure the existing WiFi networks. The measurement result is shown in Figure 7. According to the figure, we can see a great number of industrial WiFi networks running during that period. Our measurement used the frequency 2.47 GHz. According to the figure, there existed a lot of interferences from those WiFi networks.

We deployed the network manager at the center and randomly placed 4 sensor nodes and 4 actuator nodes around it. Each sensor node periodically sends data packets with the length of 35 bytes to the network manager and the network manager periodically sends data packets to all actuator node with the same length. The refresh rate is one packet per second. Furthermore, during our measurement, we set K_{max}

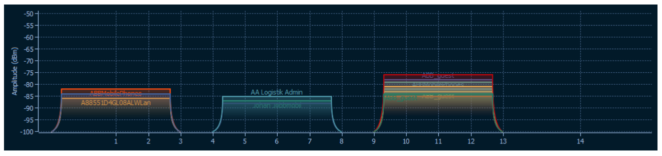

Fig. 7. The existing WiFi networks during the measurement

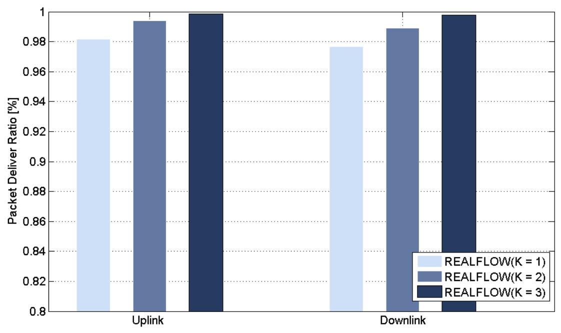

Fig. 8. Packet delivery ratios with deadline

to be 1, 2 and 3 and measured for more than one hour each time. Thus, more than 4000 packets are sent for each K_{max}. Due to the limited available timeslots, we set the maximum value of K_{max} as 3, otherwise, the network will become unschedulable.

B. Measurement Results and Analysis

We measured the average packet delivery ratio (PDR) with the deadline constraint to investigate the reliability and real-time performance of our prototype. Since hard deadline is required by mission-critical industrial applications, outdated transmissions are deemed as communication failure. So only successful uplink and downlink transmissions within the deadline are considered. The PDR measurement for both uplink and downlink is shown in Figure 8. According to the measurement results, because of the TDMA and flooding mechanism, PDRs from different configurations are higher than 97.5%. Moreover, when the maximum allowed parent number K_{max} increases, the average PDR for both uplink and downlink increases as well. The reason for this is straightforward. Because when the value K_{max} increases, each node is able to more parent nodes, a packet can be delivered to the next hop via more paths. Due to the multipath diversity, the overall reliability can be effectively increased. It is notable that when three parent nodes are selected, the average PDRs for both uplink and downlink are almost 100%.

To exhibit the reason behind, we explored the types of transmission paths from the source to the destination, shown in Figure 9. In this figure, the blue bars represent the percentage of the communication via the main path, whereas the red bars indicate the percentage of packet delivery through alternative paths. It is notable that even when K_{max} equals one, the alternative paths still exist. Since the destination address on the datalink layer is a broadcasting address,

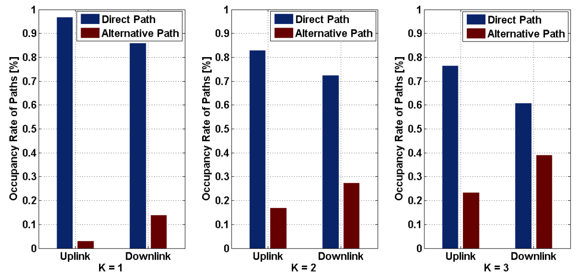

Fig. 9. Statistic of transmission paths

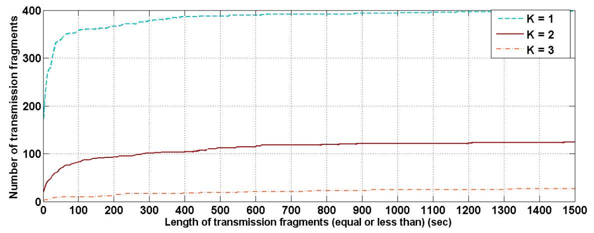

Fig. 10. Time length between two erroneous transmission

packets do not need to strictly followed specific paths for delivery. Intermediate nodes can be bypassed depending on the channel conditions, which reveals the inherent advantage of the flooding mechanism. As K_{max} increases, more alternative paths are involved, which can explain the previous results. However, the larger K_{max} means more network resources required. Therefore, there is a trade-off to choose the K_{max} value.

In order to further evaluate our prototype and analyze the measurement results, we investigated the transmission fragments regarding different K_{max} values. We define a transmission fragment to be the time duration between two communication errors. More transmission fragments mean that communication is more often disturbed by interferences. For each fragment length, the accumulative number of fragments is investigated. More specifically, if a fragment length is L s, all fragments whose lengths are not longer than L are counted. Then the statistics is shown in Figure 10. The length 0 means no consecutive error. Only uplink is involved in this investigation due to the lack of the error position information for the downlink. In Figure 10, the curve $K_{max} = 1$ starts from the highest position, around 150, which indicates that there are a great number of consecutive communication failures when only one parent is allowed to be chosen. When $K_{max} = 3$, there are almost no burst errors. Since consecutive errors may cause application failures, the result indicates that by increasing K_{max} value the availability of industrial applications can be significantly improved. According to the figure, the number of transmission fragments with $K_{max} = 1$ soars between the time length 0 and 100. It reveals that communication errors frequently occurs and cause a great number of short communication fragments. When K_{max} increases, the changes of the curve becomes less drastic, which shows that the industrial applications are more prone to be stabilized by sacrificing the networking efficiency.

V. Conclusions and Future works

In this paper, we built an IWSAN prototype aiming for meeting the stringent requirements from industrial automation systems, because there is still lack of full implementations of IWSAN to provide both reliable and real-time communications for uplink and downlink. Two hardware platforms were used in the prototype implementation. The protocol stack, from the physical layer to the application layer, is also implemented in the prototype based on a RTOS-based architecture, including time synchronization, REALFLOW realization and timeslot management. Finally, we conducted an initial measurement using the prototype in a real industrial environment. According to the measurement results, 99% PDR can be achieved and no consecutive errors were seen with deadline constraints by configuring appropriate parameters. Although a few failures still exist, oversampling is usually applied in many industrial applications so that industrial systems can be tolerant of a few communication errors. Therefore, our measurement results show great confidence in apply IWSANs for industrial mission critical applications. In order to further test the prototype and investigate our previous research works, more measurements and analysis are required in the future, but we should bear in mind that to get access to real industrial environments is very hard due to production in restricted areas.

References

[1] J. Åkerberg, M. Gidlund, and M. Björkman, Future research challenges in wireless sensor and actuator networks targeting industrial automation, in IEEE 9th International Conference on Industrial Informatics (INDIN11), July 2011.
[2] A. Gonzalez, N. Leone, M. Murdoch, P. Mazzara, and J. Oreggioni, A wireless sensor network implementation for an industrial environment, in Argentine School of Micro-Nanoelectronics Technology and Applications (EAMTA), 2010, Oct 2010, pp. 82-86.
[3] B. Lu, T. Habetler, R. Harley, and J. Gutierrez, Applying wireless sensor networks in industrial plant energy management systems. part ii. design of sensor devices, in Sensors, 2005 IEEE, Oct 2005, pp. 6.
[4] F. Salvadori, M. de Campos, R. de Figueiredo, C. Gehrke, C. Rech, P. Sausen, M. Spohn, and A. Oliveira, Monitoring and diagnosis in industrial systems using wireless sensor networks, in Intelligent Signal Processing, 2007. IEEE International Symposium on, Oct 2007, pp. 1-6.
[5] H. Wang, L. Li, J. Fu, W. Bao, and T. Wang, The design and implementation of dual-mode wireless sensor network for remote machinery condition monitoring, in Control and Decision Conference (CCDC), 2013 25th Chinese, May 2013, pp. 2765-2769.
[6] (2010) Hart 7 specification, http://www.hartcomm.org/.
[7] Industrial society of automation, http://www.isa.org/.
[8] Shenyang institute of automation, http://www.industrialwireless.cn/.
[9] J. Åkerberg, F. Reichenbach, M. Gidlund, and M. Björkman, Measurements on an industrial wireless hart network supporting profisafe: A case study, in Emerging Technologies Factory Automation (ETFA), 2011 IEEE 16th Conference on, Sept 2011, pp. 1-8.
[10] K. Yu, Z. Pang, M. Gidlund, J. Åkerberg, and M. Björkman, Realflow: Reliable real-time flooding-based routing protocol for industrial wireless sensor networks, International Journal of Distributed Sensor Networks, vol. 2014, p. 17, 2014.
[11] K. Yu, M. Gidlund, J. Åkerberg, and M. Björkman, Low jitter scheduling for industrial wireless sensor and actuator networks, in Industrial Electronics Society, IECON 2013 - 39th Annual Conference of the IEEE, Nov 2013, pp. 5594-5599.