Fault Detection Variants of the CloudBus Protocol for IoT Distributed Embedded Systems

Alexander BARKALOV[1], Larysa TITARENKO[1], Grzegorz ANDRZEJEWSKI[1], Kazimierz KRZYWICKI[1], Malgorzata KOLOPIENCZYK[2]
[1]*Institute of Metrology, Electronics and Computer Science*
[2]*Institute of Control and Computation Engineering*
Faculty of Computer, Electrical and Control Engineering, University of Zielona Gora
ul. Prof. Z. Szafrana 2, 65-516 Zielona Gora, Poland
k.krzywicki@wiea.uz.zgora.pl

***Abstract*—Distributed embedded systems have become larger, more complex and complicated. More often, such systems operate accordingly to the IoT or Industry 4.0 concept. However, large number of end modules operating in the system leads to a significant load and consequently, to an overload of the communication interfaces. The CloudBus protocol is one of the methods which is used for data exchange and concurrent process synchronization in the distributed systems. It allows the significant savings in the amount of transmitted data between end modules, especially when compared with the other protocols used in the industry. Nevertheless, basic version of the protocol does not protect against the system failure in the event of failure of one of the nodes. This paper proposes four novel variants of the CloudBus protocol, which allow the fault detection. The comparison and performance analysis was executed for all proposed CloudBus variants. The verification and behavior analysis of the distributed systems were performed on SoC hardware research platform. Furthermore, a simple test application was proposed.**

***Index Terms*—decentralized control, industrial communication, fault diagnosis, internet of things, machine-to-machine communications.**

I. INTRODUCTION

Data exchange between the nodes/end-modules (EM) in the distributed embedded systems (DES) is one of the foundations [1-7]. None of the distributed system could operate without information from the other EMs. The rapid development and low costs of DESs cause a significant increase in the number of modules operating in such systems. Furthermore, such systems more often operate accordingly to the IoT or Industry 4.0 concept [1],[4-7]. This reflects directly into significant load of communication interfaces [9]. This is particularly important in the case of wireless transmission. Due to the limited number of bands and the interferences, this problem is especially noticeable in developed countries, where there is a high density of wireless transmitters and end-point devices. Because of, the researches concentrate on reducing the amount of interferences and the amount of the data transferred between nodes.

Usually to face with interferences we need to have knowledge about global channels/bands. In [10] the authors provide examples of iterative algorithms that utilize the reciprocity of wireless networks to achieve interference alignment with only local channel knowledge at each node. This research was performed for small, specific networks and need to be investigated for larger networks. The authors of [11] propose cognitive radio (CR) methods, which offer a potential solution to improve interference resistance for industrial wireless sensor networks by integrating into the lower layers which may enable devices to detect and avoid interferences. Furthermore, researches focus over the channel control [8] and interference cancellation in 5G networks [12]. The existing interference management schemes will not be able to address the interference management problem in prioritized 5G networks [12-13].

Another approach is connected with methods of reducing the interferences based on diminishing the amount of transferred data. In [14] the authors describe progressive optimization approach on multirate features of wireless communications. Furthermore, for existing protocols researches are conducted on the relevant aggregation and distribution of data. In [15] the authors propose a taxonomy and classification of existing data aggregation scheduling solutions. Previous researches [16-17] in the field of data exchange methods for DESs have shown that the utilization of communication protocols commonly used in the industry (Modbus, Profibus, DeviceNet [19-21]) has to be carefully matched to the application. It is caused by the amount of the transmitted data, which increases significantly with increasing the amount of modules. This is particularly important in the case of Modbus protocol, where the difference in the amount of transferred data when compared to CloudBus ranges tens of times. Profibus-DP and DeviceNet give better results, which are the closest to CloudBus protocol. End modules which communicate using these protocols in the case of failure of one of the nodes, may report a failure immediately after a response timeout when it does not receive any information from another end module. In the case of CloudBus protocol, it is a problem, because characteristic of the communication and synchronization model, will result in the suspension of the whole system. It leads to situation when the end module will wait indefinitely for a response. Depending on the executed task by the system, it can cause unstable behavior, damage of controlled object or other operator injuries.

This paper proposes additional novel variants of the CloudBus protocol, which allow the fault detection in safety-critical systems. This allows taking certain steps to diagnose the fault, halt the system or enable other redundant end modules.

The paper is organized as follows: Section II describes the general principle of the CloudBus protocol, Section III

Digital Object Identifier 10.4316/AECE.2017.02001

proposes novel variants of the CloudBus protocol, Section IV provides the research results with comparison and performance analysis, Section V proposes test application, Section VI provides description and comparison to related work. Finally, Section VII concludes the paper.

II. CloudBus Protocol

The CloudBus protocol is one of the methods of the data exchange and concurrent process synchronization in the distributed embedded systems. It allows significant savings in the amount of transmitted data among end modules, when compared with the other protocols used in the industry [16-17]. It is located in the second (Data Link) layer of OSI model. In the CloudBus communication model, all modules or embedded systems (ES) are equal and operate on equal rights. This model allows using it as internal network or through Internet connection, accordingly to the IoT or Industry 4.0 concept. Figure 1 presents general CloudBus topology.

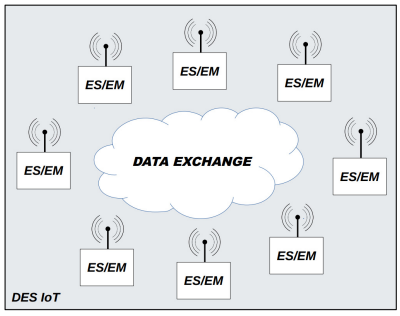

Figure 1. General network topology of the CloudBus protocol

Each end module is self-independent and implements its own part of the control algorithm. The CloudBus protocol data exchange method is based on the rule that data transfer between end modules is executed only when one of the end modules requires information from outside its own, native resource variables. General communication algorithm is presented in Fig. 2. The end module (*Mx*) broadcasts a request to the system (other end modules) about the state of the specified variable, e.g. *if Xn = h?*, where *h* is the variable state. Afterwards the module which manages the variable (*My*), responds to the system. The response is sent only if a variable matches the value sent in a request.

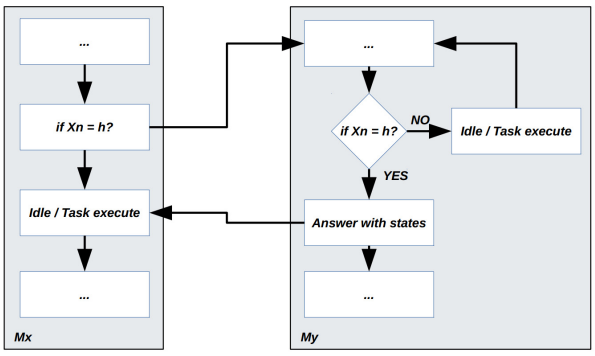

Figure 2. General communication model of the CloudBus

Otherwise, the communication between modules either does not occur or occurs only to verify the presence of the modules. This model allows the significant savings in both the amount of transmitted data between end modules and communication frequency. Moreover, it provides concurrent process synchronization in distributed systems. However, it can also lead to a situation in which some module will wait indefinitely for information about the state of a specific variable. It does not matter, in the case of systems, which are not safety-critical. Nevertheless, erroneous industrial process control can have an impact to the production process, the safety of staff or the controlled equipment. The general structure of the CloudBus data frame is presented in Figure 3. This frame is universal and multifunctional. It allows sending the different commands between end modules (e.g.: request, response, module presence etc.).

CNT	FUNC	VARS	DATA	CRC

Figure 3. General data frame structure of the CloudBus protocol

The fields in Fig. 3 mean the following:
CNT – frame length; *FUNC* – function code; *VARS* and *DATA* – represents binary data array of variables and their values; *CRC* – checksum.

III. Proposed Protocol Variants

CloudBus-BS

This is the basic version of the CloudBus protocol (described in Section II). For easier identification of CloudBus variant names, it will be called CloudBus-BS. This version does not provide any fault detection algorithms for nodes. The maximum reaction time for each cycle per module is defined by the following formula:

$$\Delta\tau_{BS\max} = \sum_{n=1}^{m}(\Delta\tau_{MED\,n} + \Delta\tau_{TRANS\,n} + \Delta\tau_{CPU\,n}) \qquad (1)$$

In (1), $\Delta\tau_{BS\max}$ is a maximum reaction time for CloudBus-BS protocol per cycle, per node, $\Delta\tau_{MED\,n}$ is a delay of transmission medium, $\Delta\tau_{TRANS\,n}$ is a real transmission time and $\Delta\tau_{CPU\,n}$ is a CPU processing time.

It should be noted that most of the variables will be equal to zero, because data exchange between nodes is processed only in specific synchronization points. In the other case, system where all variables are shared can be regarded as single-unit system, because distribution of nodes is no longer needed.

CloudBus-PC

The simplest way to provide fault detection for the ES is the presence verification of the EMs. This allows for constant monitoring of the system status. Figure 4 presents general communication model of the CloudBus-PC variant. Each end module (*Mx*) in each cycle, sends to the system (other EMs – *My*) heartbeat frame (additional command 0x0B in *FUNC* – Fig. 3). This frame (*My – Received heartbeat frame*) informs other end modules that specified shared variable still exists in the system. All nodes store information about it and they monitor maximum waiting time for receiving next heartbeat frame. If it does not arrive (*My – All vars?*) and maximum waiting time ($My - \tau_{RESP\,n} > \tau_{RESP\,max}$) is exceeded current node will report an error (*My – Fault*). In the other case, current node returns to task execution (*My – Idle/Task execute*). It should be noted that CloudBus-PC allows to check whether the end module is alive (the variable still exists in the system), but it cannot check a proper operation of specific node.

The maximum reaction time for each cycle per module is defined by the following formula:

$$\Delta\tau_{PC\max} = \sum_{n=1}^{m-1}(\Delta\tau_{BS\max n}) + \tau_{RESP\,n} \qquad (2)$$

In (2), $\Delta\tau_{PC\max}$ is a maximum reaction time for CloudBus-PC protocol per cycle, per node, $\Delta\tau_{BS\max n}$ is a maximum reaction time for node per cycle and $\tau_{RESP\,n}$ is a maximum response waiting time. Additional sum for $\Delta\tau_{BS\max n}$ is the result of sending presence checking frames in each cycle per module. The frame length and the amount of transmitted data depend directly on the number of modules with shared variables. When the number of modules increases, then the number of frames that need to be transmitted will also increase.

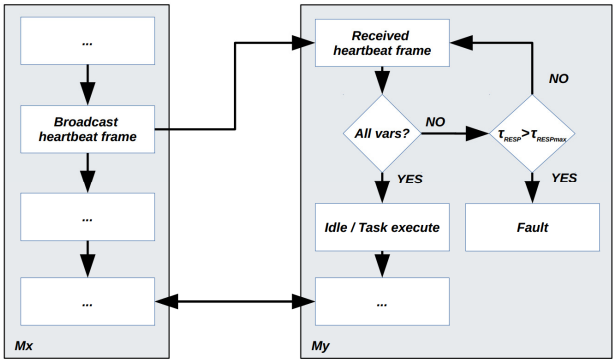

Figure 4. General communication model of the CloudBus-PC

CloudBus-DPC

The CloudBus-DPC is similar to CloudBus-PC, because it is based on the same rule i.e. presence checking. However, in this variant presence checking is executed only between EMs which share synchronization variables. Figure 5 presents general communication model of the CloudBus-DPC protocol. In this variant, direct presence checking is performed by the node (*My*) which uses some shared variable from the other node (*Mx*). It sends direct command (0x0C in *FUNC* and polled node address in *DATA* – Fig.3, *Check if var exists?* – Fig.5) to the node which is responsible for specified variable (*Mx*). If it receives the response (*My – Response*), it means that shared variable still exists in the system. In the other case, if maximum waiting time exceeds ($My - \tau_{RESP\,n} > \tau_{RESP\,max}$), it will report system fault (*My – Fault*).

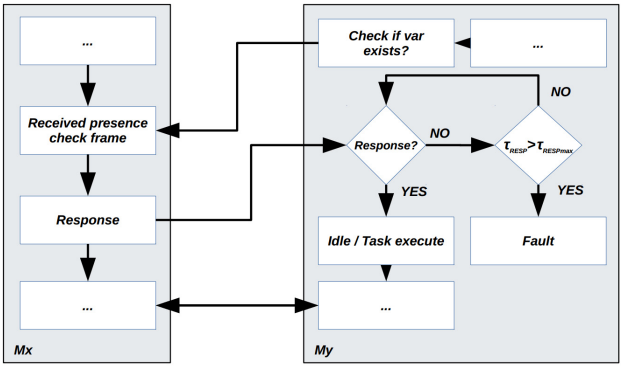

Figure 5. General communication model of the CloudBus-DPC

The maximum reaction time for each cycle per module is defined by following formula:

$$\Delta\tau_{DPC\max} = \Delta\tau_{BS\max n} + \Delta\tau_{BS\max m} + \tau_{RESP\,n} \qquad (3)$$

In (3), $\Delta\tau_{DPC\max}$ is a maximum reaction time for CloudBus-DPC protocol per cycle, per node, $\Delta\tau_{BS\max n}$ is a maximum reaction time for node (which uses shared variable) per cycle, $\Delta\tau_{BS\max m}$ is a maximum reaction time for node (which holds the shared variable) and $\tau_{RESP\,n}$ is a maximum response waiting time from *m* node. The frame length and the amount of transmitted data depend directly on the number of modules with shared variables. Nevertheless, in reference to CloudBus-PC, not all of the others modules will check its presence in each cycle, but only that which will use shared variable in current cycle.

CloudBus-TL

The CloudBus-TL does not utilize any additional data frames for fault detection, so the protocol data structure frame is the same as in CloudBus-BS. In reference to CloudBus-PC or CloudBus-DPC, this variant allows detecting failure in system behavior. It is based on maximum waiting time in synchronization points. This allows precise detecting in which point of controlled process the fault was noted. Figure 6 presents general communication model of the CloudBus-TL protocol. Each node in its synchronization point broadcasts the question about the state of the specified variable (*My – if Xn = h?*) and waits finite time for receiving its state. If the shared variable arrives (*My – Response?*) before maximum waiting time exceeds ($My - \tau_{SYNC\,n} > \tau_{SYNC\,max}$), then the system will operate normally (*My – Idle/Task execute*). In the other case, the failure will be returned (*My – Fault*).

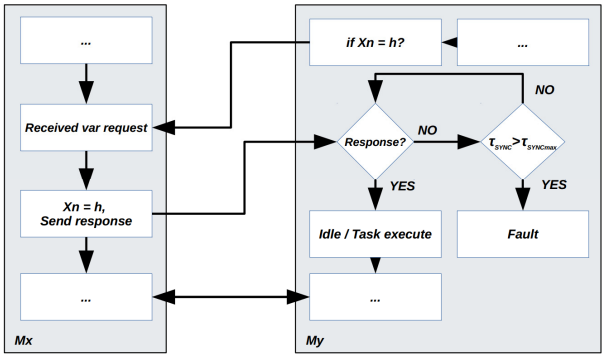

Figure 6. General communication model of the CloudBus-TL

The maximum reaction time for each cycle per module is defined by the following formula:

$$\Delta\tau_{TL\max} = \Delta\tau_{BS\max n} + \tau_{SYNCn} \qquad (4)$$

In (4), $\Delta\tau_{TL\max}$ is a maximum reaction time for CloudBus-TL protocol per cycle, per node, $\Delta\tau_{BS\max n}$ is a maximum reaction time for node (which uses shared variable) per cycle and $\tau_{SYNC\,n}$ is a maximum waiting time for shared variable, before fault will be reported.

This variant does not provide any additional data transmission, but maximum timings ($\tau_{SYNC\,max}$) for each synchronization point have to be chosen very carefully. Usually, specific constant time have to be adjusted directly on controlled object.

CloudBus-PCTL

The CloudBus-PCTL is a hybrid method where TL and PC variants are combined together. This architecture provides two different levels of fault detection. First is, the nodes presence check (PC) where the system is able to determine whether the node still presents in the system or not. The second - time limit (TL), where system recognizes

proper operation by checking the maximum time limit in synchronization point defined in control algorithm. It means that each node collects information about working nodes in the system and also watches the maximum time in synchronization points. Figure 7 presents general communication model of the CloudBus-PCTL protocol. In each cycle, each end module (*Mx*), sends to the system (other EMs – *My*) heartbeat frame. This frame (*My – Received heartbeat frame*) informs other end modules that specified shared variable still exists in the system. Afterwards, each node in its synchronization point broadcasts the question about the state of the specified variable (*My – if Xn = h?*) and waits finite time for receiving the state. If the shared variable arrives (*My – Response?*) before maximum waiting time exceeds ($My - \tau_{SYNC\,n} > \tau_{SYNC\,max}$), then the system will operate normally (*My – Idle/Task execute*). Otherwise, if the maximum waiting time exceeds ($My - \tau_{RESP\,n} > \tau_{RESP\,max}$ or $\tau_{SYNC\,n} > \tau_{SYNC\,max}$), it will report a failure (*My – Fault*).

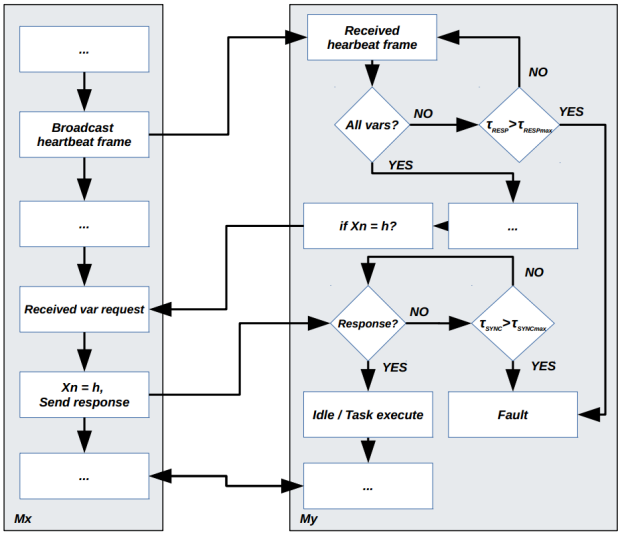

Figure 7. General communication model of the CloudBus-PCTL

The maximum reaction time for each cycle per module is defined by the following formula:

$$\Delta\tau_{PCTL\max} = \sum_{n=1}^{m-1}(\Delta\tau_{BS\max n}) + \tau_{RESP\,n} + \tau_{SYNCn} \quad (5)$$

In (5), $\Delta\tau_{PCTL\max}$ is a maximum reaction time for CloudBus-PCTL protocol per cycle, per node, $\Delta\tau_{BS\max n}$ is a maximum reaction time for node per cycle, $\tau_{RESP\,n}$ is a maximum response waiting time and $\tau_{SYNC\,n}$ is a maximum waiting time for shared variable, before fault will be reported. Additional sum for $\Delta\tau_{BS\max n}$ is the result of sending presence checking frames in each cycle per module. This variant provides highest reaction time and amount of transferred data, but also gives the highest probability to detect failure in the system.

IV. Research Results and Performance Analysis

To ensure compatibility with previous studies [16-17], the following assumptions were made for performance analysis of CloudBus protocol variants: sixteen operating EMs (nodes); 8 digital inputs/outputs per node; 4 analog inputs/outputs per node (each variable has 16 bits); the maximum waiting time ($\tau_{RESP\,n}$) for response was set to 15 ms (CloudBus-PC/DPC/PCTL); the maximum waiting time ($\tau_{SYNC\,n}$) for synchronization was set to 500 ms (CloudBus-TL/PCTL); the CPU processing time ($\Delta\tau_{CPU\,n}$) was rounded to 1 ms per each cycle; $\Delta\tau_{MED\,n}$ was rounded to 0 ms (medium is selected for specific application); the transmission speed was set to 100 Kibit/s.

The end module which requires information from outside its own resource variables broadcasts a question (*Request* – total: 161 bytes) to the system (other 15 modules) regarding certain states of their inputs/outputs. The response is sent by each node (*Response* – 14 bytes per module, total: 210 bytes). Due to the architecture of the CloudBus protocol, the output state is set directly by each node. Data exchange between modules is limited to separate sub-process synchronization points. Otherwise, the communication between modules occurs only to verify the presence (*Presence* – total: 5 (CloudBus-PC/PCTL) or 10 bytes (CloudBus-DPC) per node) of the modules. In CloudBus-TL each end module waits for a limited time and then reports a failure if it does not receive any information about the state of the variable.

Scenario I

We assumed an extremely pessimistic scenario, where each node uses shared variables from all other EMs. In the real distributed embedded system such a situation would be regarded as a mistake at the design phase. Such a system (in which all variables are shared) is usually compressed to a single-unit embedded system.

TABLE I. Traffic Amount Comparison for Different Variants of the CloudBus Protocols under Scenario I Assumptions

Protocol	Traffic amount [Bytes]			
	Request	Response	Presence	TOTAL
CloudBus-BS	161	210	0	371
CloudBus-PC	161	210	75	446
CloudBus-DPC	161	210	150	521
CloudBus-TL	161	210	0	371*
CloudBus-PCTL	161	210	75	446*

In Table I a traffic amount comparison for different variants of the CloudBus protocols under *Scenario I* assumptions is presented. The lowest traffic amount was noted for CloudBus-BS and TL variants. This follows directly from the model characteristics. For both cases, *Presence* values are equal to 0. CloudBus-BS does not contain any error detection algorithms, and therefore does not check the presence of other modules in the system. Similarly, the CloudBus-TL fault detection is based only on timing monitoring at the synchronization points, so it also does not check the node presence. The *Scenario I* assumes that all variables are shared, so the EM in each cycle polls all other nodes about their states. This is particularly noticeable for the CloudBus-DPC where the highest traffic amount was noted. In comparison with another presence checking variants (CloudBus-PC/PCTL), the transmitted data amount is twice more for CloudBus-DPC. The difference between them follows from the fact that each node in CloudBus-PC/PCTL variant collects presence checking frames from other end modules and does not have to poll each one individually. In CloudBus-DPC each node have to poll all other nodes and after that collects the responses. This causes that amount of transmitted data for *Presence* is twice more than for DPC variant.

Symbol (*) in the tables means that the variant provides additional timings.

Table II presents the relation between the number of nodes and different variants of the CloudBus protocols under *Scenario I* assumptions. Comparing the obtained results for presence checking variants (CloudBus-PC/DPC/PCTL) with variants that do not provide additional data transmission (CloudBus-BS/TL) brings the following conclusions: CloudBus-PC/PCTL gives approximately 29% more data amount that need to be transferred and it slowly grows with the growth for number of nodes; the CloudBus-DPC compared to CloudBus-BS/TL provides approximately 58% more data amount. However, if we compare these results to Table I, we can see that growth of the total data amount is caused by the direct presence checking.

TABLE II. TRAFFIC AMOUNT COMPARISON FOR DIFFERENT VARIANTS OF THE CLOUDBUS PROTOCOLS IN THE TERMS OF MODULES QUANTITY UNDER SCENARIO I ASSUMPTIONS

Modules Qty.	CloudBus Protocol Variant/ Traffic amount [Bytes]				
	BS	PC	DPC	TL	PCTL
8	265	300	335	265*	300*
16	377	452	527	377*	452*
32	601	756	911	601*	756*
64	1049	1364	1679	1049*	1364*
128	1945	2580	3215	1945*	2580*

Table III presents comparison of maximum reaction time for different variants of the CloudBus protocols in the terms of modules quantity under *Scenario I* assumptions. The lowest maximum reaction time for fault detection variants was noted for CloudBus-PC due to its low response time ($\tau_{RESP\,n} = 15\ ms$). The highest reaction time was noted for DPC variant, because of cycle polling each node and waiting each time for a response. The TL noted constant additional reaction time due to $\tau_{SYNC\,n} = 500\ ms$ and PCTL the sum of $\tau_{RESP\,n}$ and $\tau_{SYNC\,n}$. Despite better results than achieved in comparison with other protocols, the exchange of data between all modules in each cycle results in a significant increase in latency, especially in fault detection variants. Moreover, it should be noted that different DES architectures may require different values of maximum waiting/synchronization time.

TABLE III. MAXIMUM REACTION TIME COMPARISON FOR DIFFERENT VARIANTS OF THE CLOUDBUS PROTOCOLS IN THE TERMS OF MODULES QUANTITY UNDER SCENARIO I ASSUMPTIONS

Modules Qty.	CloudBus Protocol Variant/ Time [ms]				
	BS	PC	DPC	TL	PCTL
8	21.7	24.4+39.4	27.1+120	21.7+500	24.4+539.4
16	30.4	36.5+51.5	42.1+240	30.4+500	36.5+551.5
32	47.9	60.1+75.1	72.2+480	47.9+500	60.1+575.1
64	82.9	107.5+122.5	132.2+960	82.9+500	107.5+622.5
128	152.9	202.5+217.5	252.1+1920	152.9+500	202.5+717.5

The obtained results shown that even in the worst possible scenario the amount of transmitted data and reaction time is at the acceptable level for most common applications. When the large number of end modules is used, the data amount that need to be transmitted is around 3kB and the maximum reaction time is lower than 2200 ms for the CloudBus-DPC variant which obtained the worst result of all.

Scenario II

Second scenario assumes threshold number of shared variables between modules at the level of 10% and 10% of the threshold of the number of all the synchronization points in the system. This scenario simulates a common embedded distributed system where not all shared variables are synchronized in each cycle.

TABLE IV. TRAFFIC AMOUNT COMPARISON FOR DIFFERENT VARIANTS OF THE CLOUDBUS PROTOCOLS IN THE TERMS OF MODULES QUANTITY UNDER SCENARIO II ASSUMPTIONS

Modules Qty.	CloudBus Protocol Variant/ Traffic amount [Bytes]				
	BS	PC	DPC	TL	PCTL
8	28	33	38	28*	33*
16	52	62	72	52*	62*
32	100	120	140	100*	120*
64	183	218	253	183*	218*
128	349	414	479	349*	414*

The results presented in Tables IV and V show that imposing the restrictions on the amount of shared variables and synchronization points reduced the reaction time and the amount of transferred data by several times. This is especially visible for presence checking variants (PC/DPC/PCTL). The difference in the transmitted data amount between the best and the worst result is equal to only 130 bytes. Furthermore, maximum difference of reaction time equals to 500ms.

TABLE V. MAXIMUM REACTION TIME COMPARISON FOR DIFFERENT VARIANTS OF THE CLOUDBUS PROTOCOLS IN THE TERMS OF MODULES QUANTITY UNDER SCENARIO II ASSUMPTIONS

Modules Qty.	CloudBus Protocol Variant/ Time [ms]				
	BS	PC	DPC	TL	PCTL
8	3.1	3.5+18.5	3.9+15.0	3.1+500.0	3.5+518.5
16	5.0	5.8+20.8	6.6+30.0	5.0+500.0	5.8+520.8
32	8.8	10.3+25.3	11.9+60.0	8.8+500.0	10.3+525.3
64	15.2	18.0+33.0	20.7+105.0	15.2+500.0	18.0+533.0
128	28.2	33.0+48.0	38.4+195.0	28.2+500.0	33.0+548.0

Comparing these results with the *Scenario I* shows that difference between presence checking variants (CloudBus-PC/DPC/PCTL) and CloudBus-BS/TL decreased to 19% for PC/PCTL and down to 39% for the DPC variant.

An important aspect that should be also considered is the analysis of the increase for the amount of data transferred in the terms of the transmission speed. However, even if we consider a very slow connection, such as one of the wireless sensor network and additional encapsulation of the CloudBus frame, the amount of transferred data in both scenarios is very small. Generally, the maximum speed for each of the CloudBus protocol variant is limited by the used transmission medium and processing unit.

Verification

Hardware verification was performed using the SoC research and development platform for distributed embedded systems [18]. This platform is based on three Xilinx Zynq-7000 SoC devices. The platform architecture was developed for fast physical verification and behavior analysis of the distributed embedded systems, including a very large number of end modules. General platform architecture is presented in Figure 8.

In order to verify the failure detection by a particular protocol, one of the end modules was randomly disabled. This also forced modification of research platform architecture, because it was necessary to add supplement signal EN (enable) per each node. The failure report was send to PC via UART port. The report contained

information about which node was detached and which EM triggered a failure.

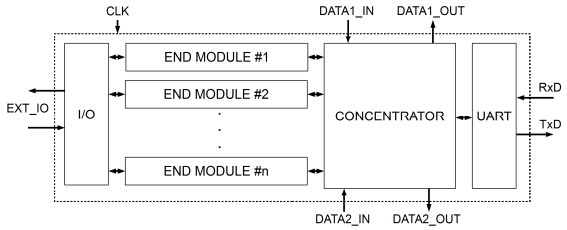

Figure 8. General architecture of the research and development platform

V. Application for IoT

The IoT smart garden was chosen as a sample application. The small vivarium (10 liters) with a few different plants inside was used as a controlled object. The main purpose of the distributed system was to monitor and control conditions of the plants. Furthermore, the scenario assumed random failure of one of the end modules for each presented protocols. The entire system was decomposed into six independent end modules (embedded systems). The general implementation diagram is presented in Figure 9.

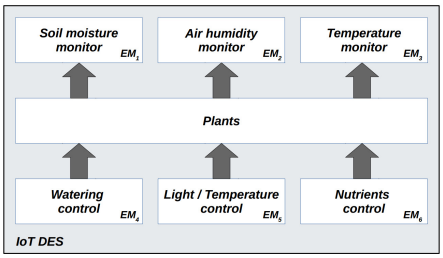

Figure 9. General architecture of implemented smart garden system

Each module was responsible for different part of the controlled process: EM_1 – monitoring the soil moisture; two analog sensors (FC-28) placed in the vivarium; EM_2 – monitoring the air humidity; two digital sensors (HTU21D and DHT22); EM_3 – monitoring the temperature inside the vivarium; one analog sensor (PT100) and one digital sensor (DS18B20); EM_4 – controls the watering; two digital (relay) outputs for direct watering from the water pump and sprinklers; EM_5 – controls the heating and UV light; two digital (relay) outputs for heating and UV bulbs; EM_6 – controls the nutrients dozing; two digital (relay) outputs.

All EMs were based on STM32F103CBT6 microcontroller with additional two digital alarm outputs (LED and buzzer) and ESP8266 (802.11b/g/n, 2.4GHz WiFi module) as the Internet gateway. The transmission between EMs of the CloudBus frames was realized by TCP protocol. Each EM stored the IP address and port of all other end modules. It was necessary for broadcast listening and direct presence checking.

TABLE VI. Fault Detection Event Count for Different Variants of the CloudBus Protocol

CloudBus protocol	Event count			
	Not recognized	Falsely recognized	Correctly recognized	TOTAL
BS	10	0	0	10
PC	0	3	52	55
DPC	0	2	53	55
TL	0	1	54	55
PCTL	0	4	51	55

For each CloudBus protocol variant, the testing scenario assumed failure of one of the nodes. The following maximum timings were assumed: $\tau_{MED\ max}$ = 100ms, $\tau_{RESP\ max}$ = 1000ms, $\tau_{SYNC\ max}$ = 500ms. Afterwards, the system reaction was monitored to detect and catch first fault. The faults were indicated by the blinking LED and buzzer from the EM, which recognized the failure. Afterwards, for each verified protocol the results of fault detection was noted. Table VI presents fault detection event count for different variants of the CloudBus protocol. The results were divided into three categories: first, *not recognized*, which correspond to fault which occurred, but was not recognized; second, *falsely recognized,* which means that system operated correct, but the fault alarm occurred; third – *correctly recognized*, which means that fault occurred and was correctly recognized. The number of repetitions for CloudBus-BS was limited to 10, because it is the basic version of the protocol and it does not provide any fault detection features. In all taken attempts, after disconnecting the one of the nodes, the system hanged and stopped in the current synchronization point without detecting the fault. Such behavior is consistent with the model and taking other repetitions does not make sense. However, it reproduces the real system failure and shows the weak point of the CloudBus-BS variant. All of the CloudBus protocol fault detection variants had properly recognized the system failure. Nevertheless, we noticed 5% of false alarms under the initial assumptions. They were caused due to random Internet connection delays – connection provided by the ISP and 16 wireless networks in the range (including at least five, which used the same WiFi band). Furthermore, some of the WiFi access points (AP) in the range have worked in the auto band mode. In this case, those AP changed the used band from time to time and caused additional interferences. In order to eliminate false alarms we increased the maximum timings up to: $\tau_{MED\ max}$ = 500ms, $\tau_{RESP\ max}$ = 1500ms, $\tau_{SYNC\ max}$ = 1500ms. It increased overall maximum fault detection time, but no false alarms were noted.

VI. Related Work

The authors lead research in the field of distributed embedded systems synthesis. These studies focus mainly on data exchange, process synchronization, automatic code generation algorithms (ACG) and IoT applications. In [17], the CloudBus protocol model was presented (described in more detail in Section II and in the *ClodudBus-BS* subsection in Section III). The paper [17] describes and compares different control methods, protocols, analyzes the performance and provides comparison of the CloudBus-BS protocol with the most popular industrial protocols i.e. Modbus-RTU, Profibus-DP and DeviceNet. The research results have shown a significant reduction (up to tens of times) in the amount of data transmitted between end modules for the CloudBus protocol. However, previous studies presented only general principle and the basic version of the CloudBus protocol. Further research confirmed that basic CloudBus variant can be utilized only in a simple systems which are not safety-critical. Hence, it was necessary to develop new architectures for the CloudBus protocol, which would provide fault detection. These novel variants which were called: CloudBus-PC,

CloudBus-DPC, CloudBus-TL, CloudBus-PCTL are proposed in current paper. Moreover, the previous research included only the performance analysis with amount of transferred data, but without the analysis of the maximum reaction time. However, it is possible to partially compare the results obtained in [17] with the results described here. Table VII, VIII and IX presents the previously obtained results from [17]. Accordingly to the assumed scenarios, the tables present traffic amount comparison for different protocols in the terms of modules quantity.

TABLE VII. TRAFFIC AMOUNT COMPARISON FOR DIFFERENT VARIANTS OF THE CLOUDBUS PROTOCOLS UNDER SCENARIO I ASSUMPTIONS FROM PREVIOUS RESEARCH

Protocol	Traffic amount [Bytes]			
	Request	Response	Output set	TOTAL
CloudBus-BS	161	210	0	371
DeviceNet	6	240	240	486
Profibus-DP	90	225	225	540
Modbus RTU	160	304	304	768

Comparing results described in Section IV (Table I) to previously presented results (Table VII) allows to make conclusion that all fault detection CloudBus variants except the CloudBus-DPC give lower traffic amount (from 7% to 50%) than protocols commonly used in the industry (i.e. Modbus-RTU, Profibus-DP, DeviceNet). The CloudBus-DPC gave the worst results among all CloudBus variants. Nevertheless, it gives lower traffic amount (from 3% to 31%) than Modbus-RTU and Profibus-DP. However, the obtained results are about 8% worse than for DeviceNet.

TABLE VIII. TRAFFIC AMOUNT COMPARISON FOR DIFFERENT PROTOCOLS IN THE TERMS OF MODULES QUANTITY UNDER SCENARIO I ASSUMPTIONS FROM PREVIOUS RESEARCH

Modules Qty.	CloudBus Protocol / Traffic amount [Bytes]			
	Cloudbus-BS	DeviceNet	Profibus-DP	Modbus RTU
8	265	246	252	384
16	377	486	540	768
32	601	966	1116	1536
64	1049	1926	2268	3072
128	1945	3846	4572	6144

Table VIII (previous research) and Table II (current research) provide results for different protocols in the terms of modules quantity under *Scenario I* assumptions. The growth of the amount of transmitted data for fault detection CloudBus protocol variants is non-linear. It means that CloudBus protocol will increase its advantage over the other protocols with the increase of node quantity. It is particularly noticeable for CloudBus-DPC. When it is over 32 nodes, it starts to have around 5% lower data amount transfer than DeviceNet (which gave better results for 16 nodes in [17]). The main cause of this situation is the growth of amount of synchronization points and outputs which has to be set. In CloudBus protocol such a situation does not occur, because each node cares by itself about its own native outputs. The diagram (Figure 11) illustrates the relation between traffic amount and the modules quantity under *Scenario I* assumptions. This figure includes current and previous research results.

It should be noted that *Scenario I* is extremely pessimistic, where each node uses shared variables from all other EMs. Nevertheless, when it is over the 32 nodes, all of the CloudBus protocol variants give better results in the amount of transmitted data than the protocols studied previously.

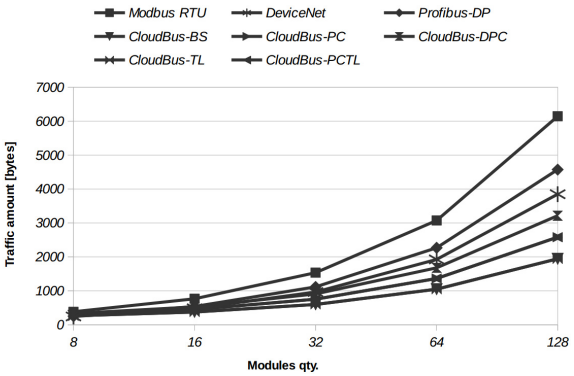

Figure 10. Traffic amount for the modules quantity under Scenario I assumptions

TABLE IX. TRAFFIC AMOUNT COMPARISON FOR DIFFERENT PROTOCOLS IN THE TERMS OF MODULES QUANTITY UNDER SCENARIO II ASSUMPTIONS FROM PREVIOUS RESEARCH

Modules Qty.	CloudBus Protocol / Traffic amount [Bytes]			
	Cloudbus-BS	DeviceNet	Profibus-DP	Modbus RTU
8	28	246	36	384
16	52	486	72	768
32	100	966	144	1536
64	183	1926	252	3072
128	349	3846	468	6144

Table IX (previous research) and Table IV (current research) provide results for different protocols in the terms of modules quantity under *Scenario II* assumptions. The differences in the amount of transmitted data for fault detection CloudBus variants and DeviceNet or Modbus RTU range tens of times. Furthermore, the growth of data amount in the terms of modules quantity for Modbus and DeviceNet protocols is almost linear. It is caused by the centralized control method, which is used by these protocols. The amount of transmitted data increases in proportion to the increase in the number of modules. Hence, duplication of the number of modules will have the effect of doubling the amount of data that need to be transmitted. Such a situation does not occur for any of the presented CloudBus protocol variants. In all cases, the fault detection variants data amount growth was non-linear. Even if we compare the worst results noticed for CloudBus-DPC and the best results from other protocol represented by Profibus-DP, we can notice that they are almost the same. Differences are from 2 to 11 bytes depending on the end module quantity. The characteristic of the data amount growth is presented in Figure 11. It illustrates the relation between traffic amount and the modules quantity under *Scenario II* assumptions. This figure includes current and previous research results. Direct comparison with other research results provided in the state of art (Section I) is impossible, because all of the methods presented in [10-15] are based on different approaches. However, depending on the application, distributed system architecture and chosen communication medium they can be used in parallel with one of the CloudBus protocol variants or combined together to multiple obtained benefits. Such an approach will allow an additional reduction in both the amount of transmitted data and the amount of interferences in wireless networks.

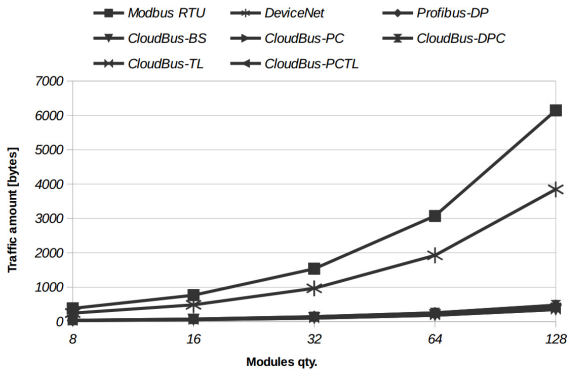

Figure 11. Traffic amount for the modules quantity under Scenario II assumptions

VII. Conclusion

This paper proposes four novel variants of the CloudBus protocol, which allows the fault detection in case of end module failure in safety-critical distributed systems. This allows taking certain steps to diagnose the fault, halt the system or enable another redundant end module. The research results showed that in the case of the synchronization of all the shared variables in each cycle, maximum response time exceeds 2000 ms for 128 nodes in the case of CloudBus-DPC protocol. The shortest reaction time for fault detection variant was recorded for CloudBus-PC protocol. Imposing the additional assumptions (*Scenario II*), which better fits to the real distributed embedded system conditions, caused that the amount of transmitted data and reaction time fall down several times when compared to *Scenario I*. Obtained reaction time results allow making the conclusion that all fault detection variants could be used in common IoT distributed embedded system application. However, it should be noted that reaction time depends directly on the architecture of the distributed embedded system. Therefore, choosing the suitable protocol should be performed in terms of architecture of specified DES and required safety level. The proposed smart garden application confirmed proper protocols operation. However, during the tests we noticed several problems related with Internet connection and wireless communication delays. All of them were independent from CloudBus protocol implementation, but it shows how important role plays data exchange in IoT implementations. The main objective of the research was achieved. We found out that all of the proposed variants do not significantly affect the communication performance of the distributed system. Furthermore, all of the variants can be used as the main communication protocol in further research in the field of automatic code generation algorithms for distributed embedded systems. Nevertheless, more research are needed to be done, i.e. how specific protocol variant affects the distributed embedded system architecture, which protocol would be the most suitable for the specific application and which will be the behavior of the protocol in real-time systems.

References

[1] J. Lee, B. Bagheri, H. A. Kao, “A cyber-physical systems architecture for industry 4.0-based manufacturing systems”, Manufacturing Letters, pp. 18-23, 2015. doi: 10.1016/j.mfglet.2014.12.001

[2] E. A. Lee, “Cyber physical systems: Design challenges”, Object Oriented Real-Time Distributed Computing (ISORC), 11th IEEE International Symposium on, pp. 363-369, 2008. doi: 10.1109/ISORC.2008.25

[3] T. A. Henzinger, J. Sifakis, “The embedded systems design challenge”, International Symposium on Formal Methods, Springer Berlin Heidelberg, pp. 1-15, 2006. doi: 10.1007/11813040_1

[4] L. Atzori, A. Iera, G. Morabito, “The internet of things: A survey.”, Computer networks, Vol. 54, no. 15, pp. 2787-2805, 2010. doi: 10.1016/j.comnet.2010.05.010

[5] J. Gubbi, R. Buyya, S. Marusic, M. Palaniswami, ”Internet of Things (IoT): A vision, architectural elements, and future directions.”, Future Generation Computer Systems, Vol. 29, no. 7, pp. 1645-1660, 2013. doi: 10.1016/j.future.2013.01.010

[6] I. Ungurean, N. C. Gaitan, V. G. Gaitan, “An IoT architecture for things from industrial environment”, Communications (COMM), 2014 10th International Conference on, IEEE, pp. 1-4, 2014. doi: 10.1109/ICComm.2014.6866713

[7] L. Zhou, H. C. Chao, ”Multimedia traffic security architecture for the internet of things.”, IEEE Network, Vol. 25, no. 3, pp. 35-40, 2011. doi: 10.1109/MNET.2011.5772059

[8] J. Milos, S. Hanus, “Analysis of LTE Physical Hybrid ARQ Control Channel”, Advances in Electrical and Computer Engineering, Vol. 14, no. 2, pp. 97-100, 2014. doi:10.4316/AECE.2014.02016

[9] H. Kopetz, “The complexity challenge in embedded system design”, Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on. IEEE, pp. 3-12, 2008. doi: 10.1109/ISORC.2008.14

[10] K. Gomadam, V. R. Cadambe, S.A. Jafar, “A distributed numerical approach to interference alignment and applications to wireless interference networks”. IEEE Transactions on Information Theory, Vol. 57, no. 6, pp. 3309-3322, 2011. doi: 10.1109/TIT.2011.2142270

[11] T. M. Chiwewe, C. F. Mbuya, G. P.Hancke, “Using cognitive radio for interference-resistant industrial wireless sensor networks: An overview”, IEEE Transactions on Industrial Informatics, Vol. 11, no. 6, pp. 1466-1481, 2015. doi: 10.1109/TII.2015.2491267

[12] S. Hong, J. Brand, J. Choi, M. Jain, J. Mehlman, S. Katti, P. Levis, “Applications of self-interference cancellation in 5G and beyond”, IEEE Communications Magazine, Vol. 52, no. 2, pp. 114-121, 2014. doi: 10.1109/MCOM.2014.6736751

[13] E. Hossain, M. Rasti, H. Tabassum, A. Abdelnasser, “Evolution toward 5G multi-tier cellular wireless networks: An interference management perspective”, IEEE Wireless Communications, Vol. 21, no. 3, pp. 118-127, 2014. doi: 10.1109/MWC.2014.6845056

[14] L. He, J. Pan, J. Xu, “Reducing data collection latency in wireless sensor networks with mobile elements”, Computer communications workshops (INFOCOM WKSHPS), 2011 IEEE Conference, pp. 572-577, 2011. doi: 10.1109/INFCOMW.2011.5928878

[15] M. Bagaa, Y. Challal, A. Ksentini, A. Derhab, N. Badache, “Data aggregation scheduling algorithms in wireless sensor networks: Solutions and challenges”, IEEE Communications Surveys & Tutorials, Vol. 16, no. 3, pp. 1339-1368, 2014. doi: 10.1109/SURV.2014. 031914.00029

[16] M. Adamski, K. Krzywicki, G. Andrzejewski, “EmbedCloud–Design and implementation method of Distributed Embedded Systems”, Technological Innovation for Cloud-Based Engineering Systems, Springer International Publishing, pp. 157-164, 2015. doi: 10.1007/978-3-319-16766-4_17

[17] G. Andrzejewski, K. Krzywicki, “Data exchange methods in distributed embedded systems”, New trends in digital systems design, Fortschritt – Berichte VDI, Dusseldorf, pp. 126-141, 2014. ISBN: 978-3-18-383610-9

[18] A. Barkalov, L. Titarenko, G. Andrzejewski, K. Krzywicki, M. Kolopienczyk, “SoC Research and Development Platform for Distributed Embedded Systems”, Przeglad Elektrotechniczny, Vol. 2016, no. R. 92 10, pp. 262-265, 2016. doi: 10.15199/48.2016.10.59

[19] “Modbus Application Protocol Specification v1.1b”, Modbus Organization, 2006

[20] E. Tovar, F. Vasques, “Real-time fieldbus communications using Profibus networks”, IEEE Transactions on Industrial Electronics, Vol. 46, no. 6, pp. 1241-1251, 1999. doi: 10.1109/41.808018

[21] S. Biegacki, D. VanGompel, “The application of DeviceNet in process control”, ISA transactions, Vol. 35, no. 2, pp. 169-176, 1996. doi: 10.1016/0019-0578(96)00022-5