UNIVERSITY of York

This is a repository copy of *AirTight: A Resilient Wireless Communication Protocol for Mixed-Criticality Systems*.

White Rose Research Online URL for this paper:
https://eprints.whiterose.ac.uk/131997/

Version: Accepted Version

Proceedings Paper:
Burns, Alan orcid.org/0000-0001-5621-8816, Harbin, James Robert orcid.org/0000-0002-6479-8600, Soares Indrusiak, Leandro orcid.org/0000-0002-9938-2920 et al. (3 more authors) (2018) AirTight: A Resilient Wireless Communication Protocol for Mixed-Criticality Systems. In: IEEE Embedded and Real-Time Computing Systems and Applications:RTCSA.

Reuse
Items deposited in White Rose Research Online are protected by copyright, with all rights reserved unless indicated otherwise. They may be downloaded and/or printed for private study, or other acts as permitted by national copyright laws. The publisher or other rights holders may allow further reproduction and re-use of the full text version. This is indicated by the licence information on the White Rose Research Online record for the item.

Takedown
If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing eprints@whiterose.ac.uk including the URL of the record and the reason for the withdrawal request.

White Rose university consortium
Universities of Leeds, Sheffield & York

eprints@whiterose.ac.uk
https://eprints.whiterose.ac.uk/

AirTight: A Resilient Wireless Communication Protocol for Mixed-Criticality Systems

A. Burns†, J. Harbin†, L.S. Indrusiak†, I. Bate†, R.I. Davis†§ and D. Griffin†
†The University of York, UK §INRIA, Paris, France

***Abstract*—This paper describes the motivation, design, analysis and implementation of a new protocol for critical wireless communication called AirTight. Wireless communication has become a crucial part of the infrastructure of many cyber-physical applications. Many of these applications are real-time and also mixed-criticality, in that they have components/subsystems with different consequences of failure. Wireless communication is inevitably subject to levels of external interference. In this paper we represent this interference using a criticality-aware fault model; for each level of interference in the fault model we guarantee the timing behaviour of the protocol (i.e. we guarantee that packet deadlines are satisfied for certainly levels of criticality). Although a new protocol, AirTight is built upon existing standards such as IEEE 802.15.4. A prototype implementation and protocol-accurate simulator, which are also built upon existing technologies, demonstrate the effectiveness and functionality of the protocol.**

I. Introduction

Many Cyber-Phyical systems (CPS) require some form of wireless communication and contain components/subsystems of different levels of criticality. In this paper we derive, implement and evaluate a new wireless protocol (which we call AirTight) that supports time-critical communication. The particular properties of mixed criticality systems has led to the definition of criticality-aware protocols and analysis for Network-on-Chip (NoC) [6], [19] and CAN [5]; here we extend this work to cater for wireless communication. Unfortunately no existing complete protocol gives the right level of support for event- and time-based communications that have hard deadlines for packet delivery (see related work in Section III). AirTight is a new protocol that is built upon the physical and MAC layers of IEEE 802.15.4, a standard for wireless personal area networks (WPANs), widely used as the basis of protocols such as ZigBee and WirelessHART.

With wireless communication, it is not realistic to only require that deadlines are met when there are no faults. Rather, as in other considerations of fault tolerance, we require that certain levels of performance are delivered when the likelihood and severity of fault(s) is bounded by what is referred to as a *fault model*. We assume that the physical layer of the protocol incorporates the usual methods of increasing resilience (for example spectrum spreading), and we therefore focus on faults than manifest themselves as unacknowledged frame transmissions at the MAC layer.

In this paper the analysis developed for AirTight allows deadlines to be guaranteed at various levels of service, corresponding to the severity of the fault model and the level of service required by the criticality assigned to the packet being guaranteed. Different fault models can be applied; the basic behaviour of AirTight is not dependent upon any particular fault model.

In the next section we discuss the requirements for which AirTight was defined. We also give an overview of the protocol and define some necessary terms. In Section III we consider related work. Section IV describes the fault model employed. Section V completes the description of the AirTight protocol, and its analysis is given in Section VI. The construction of the necessary slot table in addressed in Section VII. An illustrative example is provided in Section VIII. A prototype implementation and protocol-accurate simulator are described in Sections IX and X, these are used to undertake a number of representative experiments, as covered in Section XI. Finally, conclusions are presented in Section XII.

II. Requirements for, and overview of, AirTight

We assume the CPS consists of a distributed system of nodes that can each perform any combination of executing tasks, producing/consuming data from sensors/tasks, and relaying data packets to and from other nodes. There may be a range of communication media within the CPS; here we focus on the use of wireless technology. The required wireless network protocol is assumed to have the following properties (most of them inherited from the parent standard IEEE 802.15.4):

- Peer-to-peer packet-switching communication between tasks/nodes is the normal use of the network. Packets are sent from a node to the next as one or more *frames*. Each successful frame transmission is always acknowledged by the receiver through the transmission of a short ACK frame.
- Multi-hop routing is required due to the limited transmission range of each node, meaning that some packets are unable to be sent directly to their destination.
- Buffers exist on each node to store frames in transit (the size of the buffers required on each node can be determined during the off-line scheduling process).
- The nodes have their clocks synchronised so the maximum drift between any two clocks is at most T_{error}.
- Nodes have line power, so energy efficiency/battery life is not a limiting concern.

- Multiple frequency bands (channels) are available in IEEE 802.15.4 (up to 16 in the 2.4GHz band) and the standard is designed so that interference from one channel to another is negligible. A node can only use one channel at a time.
- Node communications are represented by two graphs: the *communications graph* and the *interference graph*:
  - The communications graph $\mathbf{C}$: if there is an edge from $A \rightarrow B$ in $\mathbf{C}$, then the two nodes can communicate directly. This is required to be a symmetric graph due to the necessity for an acknowledgement to be returned to the sender, so $A \rightarrow B$ implies $B \rightarrow A$.
  - The interference graph $\mathbf{I}$: if there is an edge from $A \rightarrow B$ in $\mathbf{I}$, then a transmission from A will prevent B from receiving a frame from any node other than A on that channel at that time.

It is assumed that the packets to be communicated have tight timing constraints (i.e. deadlines). We also require that the system supports applications of different levels of criticality. We will, in this paper, assume just two criticality levels, high (HI) and low (LO). The main distinction between these levels is the number and duration of faults that they must tolerate (see Section IV).

The distributed CPS consists of N nodes (n_0 to n_{N-1}). Each node generates a set of packet flows (or flows), τ_i, defined by:

- Period, T_i; the minimum time between packets.
- Capacity, C_i; the packet's maximum size.
- Criticality Level, L_i; a static parameter of the flow.
- Deadline, D_i; assumed initially to be no greater than T_i.
- Destination, Des_i; packets are assumed to be peer-to-peer so there is a single destination for each flow.
- Source, Sor_i; there is an implicit source for each flow.

As part of the local scheduling, each flow is assigned a unique priority, P_i.

We do not assume that the flows are purely periodic. This implies that there must be some form of run-time scheduling. However, we do not expect that centralised access control, or token passing protocols can deliver the performance required by a modern CPS. Any protocol that requires significant overhead to agree on the next packet to send is unlikely to meet strict timing requirements. The alternative of a fully table-driven time-triggered protocol lacks the flexibility needed to support event-triggered and adaptive applications.

AirTight is designed to balance efficiency and flexibility. At the system level, its media access control is table-driven, but at the node level it uses criticality-aware priority-based frame scheduling. The protocol is based around repeating slot tables which, in time, define the activities of each node – either transmission or reception on that channel, or null meaning no usage. The slot (or scheduling) table (ST) consists of a series of slots. Each slot is assigned to a node and can be used by that node to send a single data frame. The slot also accommodates the ACK frame of the respective receiver.

At each node, local scheduling decisions are made to manage the use of the node's slot allocation. We employ a fixed-priority scheme (although this is not fundamental to AirTight). A set of FIFO queues (buffers), one per priority level, are used to hold the frames that need to be transmitted. Each flow has a unique priority and hence a specific buffer. The frames from the same flow are stored in the buffer in FIFO order. Whenever the node has a slot available, it transmits the first frame in the highest priority non-empty buffer. If an ACK is received the frame is removed from the buffer; if no ACK is received, then the frame remains in the buffer and is a candidate for transmission when the next available slot for that node becomes available.

AirTight is thus a two level protocol. A collection of slot tables defines the usage of the wireless media. Each slot in a table defines whether the node can transmit in that slot (and on which channel if more than one channel is used), or whether it should listen in that slot (and on which channel), or whether it is off-duty. The collection of tables reflects the properties of the communication and interference graphs. So, within the same channel, two nodes may, in effect, be allocated the same slot if the interference graph fulfils specific properties with regards to senders and receivers [16]. The protocol is described in detail in Section V.

The fundamental time unit of AirTight is the duration (S) of a slot – the time it takes to communicate a single frame of data and receive an ACK for that frame. In our prototype implementation (see Section IX) a slot length of 10ms has been achieved, although a longer slot was used in experiments to aid collection of measurement data. All parameters of the application, the communication media and the environment (e.g. T_i, C_i, D_i, table length, fault models, etc.) are expressed as an integer number of slot times. We assume that clock drift is insignificant when compare to the slot duration: $T_{error} \ll S$.

A. An Avionics Use-case

An aircraft engine is a harsh environment for electronics and wireless communication in that there are a lot of moving mechanical parts generating both interference and attenuating radio signals. Wireless sensors have two distinct advantages: the sensors can be put deep inside the engine where it is infeasible to have cabling; and it removes the weight and maintenance of cabling. The difficulty of maintenance may also mean the designer may want to fit a number of replicas so replacement is not necessary. Current engines have a significant number of sensors (more than 10). With a shift towards more intelligent control and monitoring this number will grow. External to the engine there are a number of controlled interference sources, e.g. from the rest of the aircraft, and un-controlled interference sources, e.g. high-intensity radiated fields including lightning, mobile phones, laptops etc. This leads to complex fault behaviour that cannot be fully defined at design time. We therefore utilise a collection of fault models (one per criticality level) that are, in themselves, bounded.

Finally, a number of parts of the overall aircraft system (and logical support equipment on the ground) may want to use wireless communications and as such the aircraft engine should be designed to share the same parts of the spectrum (e.g. two nodes concurrently wanting to send messages to different destinations) especially as the whole aircraft could have hundreds if not thousands of sensors.

A good example of the potential deployment of a wireless communication media is within an aircraft engine for the purposes of active health monitoring [33]. Figure 1 shows the communication graph for a 25-node wireless network inspired by a possible engine monitoring system; it is clear that the topology of this example is a 5-node subsystem repeated 5 times. We will use this 5-node subsystem to illustrate the analysis associated with AirTight, and will validate it using a prototype network of 5 IEEE 802.15.4-compliant nodes. We will then use a protocol-accurate simulator to evaluate AirTight's performance and scalability over the complete 25-node network. In total this network has 55 packet flows mapped to the 25 nodes; 25 of these flows are defined to be of HI-criticality and 30 of LO-criticality.

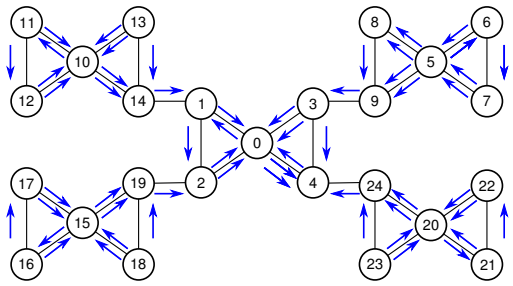

Fig. 1. Communication Graph of a 25 node Health Monitoring System

III. Related Work

In this section we consider wireless protocols that have been designed to give time predictable behaviour, and protocols that take into account mixed criticality. For information on more general purpose protocols the interested reader may follow the links from [25].

A. General Real-Time Protocols

WirelessHART [15] was developed as an extension of the HART protocol [14] designed for wired communications in industrial automation and process control scenarios. The WirelessHART protocol extensions were intended to allow mobile devices to attain the capabilities of HART networks. WirelessHART employs a time-division multiple access (TDMA) based MAC layer, with multi-hop routes centrally planned and allocated by a sink which operates as a gateway between the wireless network and external access network. A similar TDMA approach is advocated by FireFly [24], [26].

One notable aspect of WirelessHART is the avoidance of spatial reuse throughout the network, with only one simultaneous transmission allowed at any time. Although frequency reuse is permitted through the simultaneous use of multiple channels, simultaneous transmissions on any one channel are disallowed [29]. This avoids the problem of detecting interference patterns and allows the network to be stabilised, but limits scalability and restricts the viable range of the network. In addition, alternative routes are required in WirelessHART for redundant data transmission. By comparison, our current work focuses entirely on avoiding faults via temporal retransmission; that is, holding transmissions within buffers and making repeated transmissions of any failed transmission after a delay. We note that the avionics use case described earlier has little opportunity for exploiting spacial redundancy.

Saifullah [27] presents several approaches for scheduling of multi-hop routing-centric wireless networks built upon WirelessHART. Saifullah establishes that overall the wireless scheduling problem is NP-hard, and then provides heuristics to simplify the scheduling decisions. The first of these, Conflict-Free Least-Laxity First (CLLF) [29] schedules according to the laxity of the transmissions, that is the time remaining until their deadlines. However, it guides the scheduling process by focusing on conflicts, scheduling first in hotspot areas in which conflicts are likely (e.g. around the sink and congested frequently used devices). Flow set evaluation results show that CLLF is several orders of magnitude faster in determining schedulability than exhaustive search, although run time does increase with increased routing diversity. Saifullah et al. also present an end-to-end delay analysis [30] for an application with fixed priority flows, and extend this work to the case of graph routing in [28].

A very different approach for broadcast wireless communication is provided via Glossy [13], which attempts to simultaneously achieve time synchronisation as well as error tolerant communication via collaborative flooding of data packets. If time synchronisation is sufficiently tight, multiple nodes are able to receive packets and rebroadcast them simultaneously. Glossy therefore uses spatial diversity to compensate for any localised faults; it has been successfully used as a primitive layer to build network services such as LWB [12] and Blink [34]. However, the drawbacks of Glossy include requirements for low-level tuning of the code by inserting NOPs (no-operation opcodes) to ensure correct timing. There are also security problems such as the Arpeggio attack [17], in which an attacker is able to hijack the flooding mechanism to broadcast attack packets more widely.

B. Mixed Criticality Protocols and Applications

Several papers have focused on mixed criticality in WSNs and CPSs. Alemayehu et al. [1] consider mixed-criticality used for video transmission in a multimedia sensor network. The paper uses different criticality levels for different resolution video frames, to provide graceful degradation under error by providing a low resolution alternative. They

focus on changing criticality in response to a reduction in overall network bandwidth, and can discard data in response to criticality and priority to improve overall performance. An interesting contrast to our work is that the paper did not use TDMA and a pure wireless topology but a hybrid CSMA/CA over 802.11, yet still achieved end-to-end response time reductions.

Shen et al. [31] present the PriorityMAC protocol. The paper uses a concept of priority levels, but their definition overlaps with criticality – the importance of reliable delivery from the application perspective. It defines four traffic types and requires windows at the start of every application message for the two highest priority traffic levels to reserve capacity. Nodes must listen to these windows and detect them as clear if they wish to transmit the two highest priority traffic types. This allows dynamic adjustments in priority by allowing nodes with higher priority traffic to use slots previously assigned to other nodes, but at the cost of channel capacity reductions. Moreover, their analysis and evaluation only considers average-case latencies and does not provide a worst-case guarantee.

Jin et al. [21] considers delay analysis for WirelessHART networks supporting mixed criticality under fixed priorities. An interesting aspect of their model is that they assume a global network criticality level, and a broadcast mechanism to signal a criticality mode change similar to WPMC-FLOOD [19] within a network-on-chip (NoC). However, they do not specify any low level router behaviour to achieve the change, merely assuming the change takes a maximum defined time. It does not appear to use criticality monotonic arbitration, instead assuming that the LO-criticality flows are dropped rather than buffered, due to the low memory of the devices.

StealRM [20] is an alternative protocol which provides redundancy and is employed for HI-criticality flows, with the transmissions of HI-criticality packets along two duplicate routes to protect against interference or damage to one copy. The algorithm centrally establishes schedules for both the LO and HI-criticality flows, but allows nodes to make the distributed decisions for transmission of HI-criticality in place of a LO-criticality when required. This is done by means of a clear channel assessment performed by the radio before making every LO-criticality transmission, which ensures that interference is avoided. Therefore, LO-criticality packets may be destroyed when HI-criticality packets require the resource, referred to as slot stealing. This approach is similar to the approach of Shen et al. with PriorityMAC [31], since it allows nodes to request additional capacity dynamically.

Dimopoulos et al. [11] consider mixed criticality systems, specifically for smart building infrastructure. One interesting aspect they mention is the potential use of software defined radios in the implementation of CPS, in order to present more adaptability in the behaviour and protocols employed. In nodes with more resources than conventional WSN motes this may be a viable solution. They also consider mixed criticality in wireless systems to require levels of autonomous management in different regions, contrasting with the implicitly centralised management and routing-centric designs required for WirelessHART (and assumed in the extensive scheduling studies performed in [27]).

In summary, by comparison with the above approaches, AirTight is the first mixed-criticality wireless protocol that incorporates local scheduling decisions and which delivers time-bounded performance that is sensitive to whatever fault model is deemed appropriate for the system under consideration.

IV. Fault Model

A wireless network, even in a protected domain, will suffer interference that will result in some packets being corrupted. A predictable network can only be derived and analysed if there is a bound on the level of interference suffered by each node in the system. This bound is usually expressed as a *fault model*. If the level of interference is no worse than that implied by the fault model then temporal guarantees can be made. The quality of the fault model can itself be modelled using a probabilistic estimate of the likelihood of exceeding a given fault severity during, say, an hour of operation [7]. With mixed-criticality systems the required quality will vary with criticality; so for a LO-criticality transmission the fault model may bound the number of deadlines misses to be no more than 1 in 1,000, for a HI-criticality transmission this number may be extended to 1 in 1,000,000.

In general a node will suffer interference from a number of different sources. Each source will produce a pattern of interference. Moreover, in a geographically spread network each node will experience different levels of interference from different sources. To model a particular node's (n_k) level of interference we need a *fault load function*, F_k. This function, when given an interval of duration t, will return the level of interference assumed by the fault model for this node at criticality level, L; i.e. the function is defined as $F_k(L,t)$. As the basic time unit in the analysis model is the duration, S, of a single slot, both t and F_k are represented as an integer multiple of S.

We note that the fault model is always assumed to be more severe for HI-criticality packets than for LO-criticality packets. Hence we require that:

$$\forall t, \forall n_k : \quad F_k(HI,t) \geq F_k(LO,t)$$

The function F_k can be decomposed into a combination of fault load functions (f_k) for each of the w sources of interference:

$$F_k(L,t) \;=\; G_L\{f_k^1(L,t), f_k^2(L,t), \ldots, f_k^w(L,t)\}$$

where G_L is a criticality-specific application-defined means of combining the different sources of interference.

So, for example, for LO-criticality packets G_{LO} may be defined to be the MAX operator, and hence at this criticality

level the node is assumed to only suffer interference from one source at a time – but the maximum possible single level is used to define F_k. For HI-criticality packets G_{HI} may be defined to be the SUM operator, and hence all sources are assumed to contribute their maximum levels – a situation that may be impossible as interference is not cumulative.

The most straightforward way of representing a single source of interference f_k is via a duration and a frequency. So a single corruption could cause a blackout for duration $b(L)$, with the minimum time between faults: $T^b(L)$. Note these parameters are functions of criticality level and are measured in units of S. A single source of interference could, for instance, be modelled by $b(LO) = 4$, $T^b(LO) = 100$, $b(HI) = 6$, $T^b(HI) = 80$: for LO-criticality transmissions this interference is assumed to last up to 4 units of time and repeat every 100 units; but for HI-criticality transmissions a more severe view is taken, the blackout can last 6 units of time and repeat every 80 units.

For an actual deployment of AirTight various signal processing schemes (for example Fourier Analysis) are available that allow the overall interference experienced at a node to be decomposed into component sources defined by these two parameters. These are statistical methods and hence the parameters derived are a functions of the level of confidence required. Higher levels of criticality will require higher levels of confidence and hence more conservative parameters will be obtained.

In this paper we do not address further this analysis of actual interference, rather we assume that by the time an implementation requires analysis the necessary fault load functions have been obtained. All that the analysis requires is that $F_k(L, t)$ is defined for all t and for each criticality level contained in the system.

V. The AirTight Protocol

In general a wireless network can be characterised by a number of properties:

- Single-hop or multi-hop (i.e. is the communications graph fully connected?).
- Single-domain or multi-domain (i.e. is the interference graph fully connected?).
- Single-channel or multi-channel.

In this first paper on AirTight we focus on multi-hop, single-domain, single-channel networks. With a single-domain, single-channel network, assuming that all nodes listen in all slots, a single table specifying which node may transmit in each slot is sufficient to define the behaviour of all nodes. The extension to multi-domain does not introduce any fundamental issues, but requires the construction of multiple tables. We also assume that the allocation of tasks and sensors/actuators to nodes has already been undertaken and hence that the sets of packet flows (and their destinations) for each node are fixed and known.

The protocol has three main phases:

1) The construction of the slot table. This is derived from the requirements of all the packet flows on all the nodes. The table is communicated to all nodes during system initialisation.
2) The run-time local scheduling of flows. Each node will, independently, make use of the slots allocated to it. This will take account of priorities, errors, and re-transmissions.
3) An adaptive system will, over time, look to modify the slot table – for example there could be free slots that nodes compete for, or unused slots that are reallocated, or potentially the complete table could change due to a system mode change.

In this paper, and the prototype implementation, a simple heuristic is used to construct the table, and the third phase (adaptation) is left for future work.

Analysis is used on each node to check for packet flow schedulability. This requires knowledge of the slot table; however, the structure of the slot table is itself a function of the schedulability of all nodes. In Section VI we first derive analysis, assuming a known slot table, and then show how the slot table can be constructed with a simple heuristic. In future work we intend to make use of a search-based algorithm (e.g. a genetic algorithm or simulated annealing) to: construct near-optimal slot table layouts, cover all required routing decisions, and cater for multi-domain and multi-channel systems.

A schedulable AirTight network behaves as follows:

- If there are no faults experienced by the system then all packets will meet their deadlines.
- If the faults experienced by the system are no worse than that implied by the LO-criticality fault model then all packets will meet their deadlines.
- If the faults experienced by the system are no worse than that implied by the HI-criticality fault model then all HI-criticality packets will meet their deadlines.
- If the faults experienced by the system are worse than that implied by the HI-criticality fault model then each node will apply a best-effort approach. The faults are deemed to be beyond the level at which guarantees can be provided.

Following the behaviour of mixed criticality task scheduling [4], three modes of operation are defined. Each node is, independently, in either LO-criticality, HI-criticality or Best-Effort mode. In the LO-criticality mode all of the node's packets are sent and they are delivered by their deadlines. If the LO-criticality fault model is exceeded, then the node moves to HI-criticality mode. In this mode, LO-criticality packets are abandoned (or moved to local background priority); however, all HI-criticality packets are still delivered by their deadlines. If the HI-criticality fault model is exceeded, then the node moves to Best-Effort mode. At any time that the output buffer of the node is empty the node can return to LO-criticality mode. (This is equivalent to the return to LO-criticality mode on an idle-tick in task scheduling).

A. Jitter Elimination

Usually with distributed systems it is assumed that the packets inherit significant release jitter from the variability in the completion times of the tasks that generate them. This jitter can then be factored into the response-time analysis [3]. Here we apply a protocol that eliminates release jitter whilst not extending the worst-case overall (i.e. end-to-end) response-time [10]. Release jitter is eliminated by the following protocol which is applied to all frames of all packets. For clarity we describe its application to a single frame f of a single packet flow τ_i. The time q when frame f of the first packet of packet flow τ_i is received by node n_k is recorded. When at a later time t the node receives frame f of the next packet of the same packet flow, then: (i) if $t \geq q + T_i$, then the frame is immediately eligible for onward transmission by n_k and q is set to t; (ii) if $t < q + T_i$, then the frame is held (i.e. delayed) and is not eligible for further transmission along its route by n_k until time $q + T_i$ is reached. At that point q is set to $q + T_i$. The same process is repeated for subsequent frames f of all packets of that flow. This protocol also applies to frames "received" by the source node from the sending task, with the initial maximum jitter due to the sending task (i.e. its worst-case response time) deducted from the end-to-end deadline. The effect of the protocol is to eliminate the interference effects of jitter, and thereby improve schedulability.

VI. AirTight Analysis

The starting point for the analysis of a complete system is a set of packet flows with Destination and Source nodes directly linked in the communications graph. In other words, all routing requirements have been met by the addition of intermediate flows passing between connected nodes (we revisit this issue as part of the slot table construction heuristic – see Section VII). We also assume local priorities have been assigned to all packet flows (an optimal assignment can be obtained by applying Audsley's algorithm [2]).

For any particular phase of the analysis the slot table is known. It has duration T_{SL} (measured in slots). Each node has one or more slots within the table; let this allocation be represented by a_i We have (note, as before, N is the number of nodes in the system):

$$\sum_{j \in N} a_j = T_{SL}$$

The required analysis is obtained from adapting three schemes/notions:

- Modelling the impact of faults by a fault load function($F_k(L,t)$), which gives the maximum number of failed slots for criticality level L in time t for node n_k.
- Basic fixed-priority analysis for mixed-criticality task scheduling – using the AMC approach [4].
- Modelling the supply function ($S(X)$), the maximum time which the slot table can take to supply X slots to the node.

A. AMC Analysis for AirTight

The AMC analysis for a collection of tasks exploits the fact that the load on the system is lighter during the LO-criticality mode, and the fact that LO-criticality tasks are dropped once the system transitions to HI-criticality mode. With task scheduling the load is less in the LO-criticality mode as tasks have smaller worst-case execution time estimates in that mode [32]. When analysing packet flows, this is not the case (although the model could easily be extended to include this). Rather it is the fault load that is lower in the LO-criticality mode. This allows us to define response-time analysis for each packet flow, τ_i on node n_k. In the LO-criticality mode:

$$R_i(LO) \; = \; C_i + F_k(LO, R_i(LO)) + \sum_{j \in \mathbf{hp}(i)} \left\lceil \frac{R_i(LO)}{T_j} \right\rceil C_j \tag{1}$$

where $\mathbf{hp(i)}$ is the set of all local (i.e. also transmitted by node n_k on part of their route) flows with priority higher than that of τ_i.

In the HI-criticality mode:

$$R_i(HI) \; = \; C_i \; + \; F_k(HI, R_i(HI)) + \sum_{\tau_j \in \mathbf{hpH}(i)} \left\lceil \frac{R_i(HI)}{T_j} \right\rceil C_j + \sum_{\tau_k \in \mathbf{hpL}(i)} \left\lceil \frac{R_i(LO)}{T_k} \right\rceil C_k \tag{2}$$

where $\mathbf{hpH(i)}$ is the set of local HI-criticality packet flows with priority higher than that of flow τ_i; and $\mathbf{hpL(i)}$ is the set of local LO-criticality packet flows with priority higher than that of flow τ_i. Note $R_i(HI)$ is only defined for packet flows of HI-criticality.

B. Sufficient Analysis for AirTight

The above analysis assumes that the required resources (the slots of the slot table) are always available for the node under investigation. This is a valid assumption for tasks executing on a single processor, since the processor is always available. With AirTight the slots are not as readily available. Indeed, as few as one in T_{SL} slots may be all that is available for the node under investigation. We therefore represent the availability of slots as a supply function: $S_k(X)$, which provides an upper bound on the time it will take for X slots to be available for node n_k. Equation (1) is now split into two parts:

$$X \; = \; C_i \; + F_k(LO, S_k(X)) \; + \sum_{j \in \mathbf{hp}(i)} \left\lceil \frac{S_k(X)}{T_j} \right\rceil C_j \tag{3}$$

with

$$R_i(LO) = S_k(X) \tag{4}$$

The equations are solved via fixed point iteration in the usual way, starting with an initial value of X of C_i. Iteration continues until either the value of X converges, in which case $R_i(LO)$ gives the worse-case response-time, or $R_i(LO)$ exceeds D_i, in which case the packet flow is not schedulable.

Similarly equation (2) becomes

$$X = C_i + F_k(HI, S_k(X)) + \sum_{\tau_j \in \mathbf{hpH}(i)} \left\lceil \frac{S_k(X)}{T_j} \right\rceil C_j + \sum_{\tau_k \in \mathbf{hpL}(i)} \left\lceil \frac{R_i(LO)}{T_k} \right\rceil C_k \quad (5)$$

with

$$R_i(HI) = S_k(X) \quad (6)$$

An example of the application of this analysis is given in Section VIII.

A number of different formulas for $S_k(X)$ are possible. When node n_k has only one slot in the table ($a_k = 1$) then it must be assumed that the worst-case possible phasing between this slot and the packet flow under consideration occurs. This implies that a frame of the packet flow arrives just after the slot has been assigned to a lower priority packet, or indeed a null or background packet is assigned. Hence there is a 'blocking time' of 1 slot. Minor clock drift is also accommodated within this blocking term.

If no internal structure for the table is known then a sufficient model for the supply function is

$$S_k(X) = 1 + \left\lceil \frac{X}{a_k} \right\rceil T_{SL}$$

So, for example, if the table is of length of six and a node has one slot within the table, then the supply function returns 7 for $S(1)$, 13 for $S(2)$ and so on. Similarly if the node has two slots in the table, then its conservative supply function is $S(1) = S(2) = 7$, $S(3) = S(4) = 13$. If the internal structure of the table is known, then a less pessimistic estimate is possible. For example, the two slots cannot both come at the end of the table, so an improvement is $S(1) = 6$, $S(2) = 7$, $S(3) = 12$, $S(4) = 13$ etc. Moreover, if the table is known to have allocated the two slots to positions (1 and 4, or 2 and 5, or 3 and 6) then the supply function becomes: $S(1) = 4$, $S(2) = 7$, $S(3) = 10$ etc. This latter supply function dominates the more pessimistic ones discussed above, since it provides the same number of available slots in the same or less time, for any number of required slots.

C. Priority Assignment

Each node has a separate priority-ordered output buffer for its frames. An optimal priority ordering for this buffer can be obtained by applying Audsley's algorithm [2]. The applicability of this algorithm comes directly from the fact that the scheduling scheme being applied is equivalent (indeed identical) to that employed in mixed-criticality task scheduling.

VII. Slot Table Construction

In this section we describe a simple heuristic for constructing an appropriate slot table; however, first the issue of packet routing must be considered. In general routing must be addressed as part of the mapping of an application to the available hardware. Here we again take a simpler approach.

1) The shortest route between Source and Destination is chosen (an arbitrary choice is made if there is more than one route with the same length).
2) The deadline of the packet is partitioned between each hop of the route, and priorities assigned by applying Audsley's algorithm [2].
3) Response times are computed for each hop and summed to obtain the end-to-end response-time that is then compared with the end-to-end deadline.
4) Initially the packet deadline is divided equally between the hops, if any hop is unschedulable (i.e. $R > D$) its deadline is increased (to R, but not exceeding T) (while others are decreased when $R < D$).

Clearly this is a non-optimal approach. Although the heuristic does account for 'busy' nodes by allowing them to have more slots in the table (see below). As indicated earlier, under future work we aim to explore the use of a search-based algorithm to deal with routing, table construction, non-interfering flows and multiple channels. The purpose of the current paper is to give a definition of the protocol, the derivation of its enabling analysis and argue for the feasibility of the protocol via a complete proof-of-concept prototype.

The following steps are undertaken to construct a slot table. Here we aim to derive a small slot table (ST) that delivers schedulability for all nodes. (Note, we do not claim that the size of the slot table is necessarily minimized).

1) Collect the set of flow requirements for each node.
2) Add multi-hop routing flows where appropriate.
3) Initiate the slot table to have length $T_{ST} = N$, where N is the number of nodes in the system.
4) Assuming each node has one slot in the table of length N, check for local schedulability of all nodes. If any node is not locally schedulable, compute how many extra slots that node would require in the table.
5) Extend the size of the table to accommodate these extra slots.
6) Repeat the schedulability test for all nodes with the longer table.
7) Repeat this process (testing and extending the table) until either all nodes are schedulable with the new table size, or the table keeps growing until 'unschedulability' is declared if the table grows to beyond the hyper-period of all the flows.

VIII. Illustrative Example

In this section we analyse a simple five node example which is motivated by the subsystem identified in the avionics use case (as illustrated in Figure 1). The 5 nodes (which are the central group in Figure 1) are depicted in Figure 2 and form a star topology: $n_1, n_2 \leftrightarrow n_0 \leftrightarrow n_3, n_4$. So n_0 can communicate directly with all nodes; but n_1, for example, can only communicate with n_2 and n_0, and not

with n_3 or n_4. However, we assume conservatively that the interference graph of the system is complete such that all nodes may potentially interfere with each other, even though they may be out of range for intelligible communication. Therefore, we do not allow node n_1 to transmit to n_2 at the same time as n_3 transmits to n_4. We use periods and deadlines that are 1/5 of those of the larger example as this allows the tightness of the analysis to be illustrated in this section. It also means that the full 25 node example is obviously schedulable if this subsystem is.

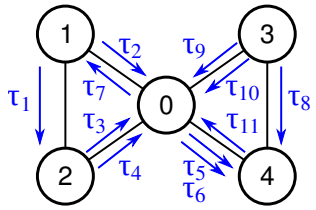

Fig. 2. Diagram of the nodes and communicating packet flows

There are nine end-to-end packet flows of two criticality types. Two packet flows must be routed through n_0; these are accommodated by simply, for this example of the analysis, dividing the deadline in two. This results in eleven packet flows that are given in Table I (where P is the local priority, with 1 being the highest)

For this simple example we assume a single source of interference; the fault model is defined as follows:

1) For $b = 5$ and $T^b = 100$, all deadlines must be met
2) For $b = 15$ and $T^b = 100$, the deadlines of all HI-criticality flows must be met
3) For $b > 15$ and any $T^b < 100$, best effort – send HI-criticality packets when possible, perhaps using a secondary parameter (importance) to order local access.

If we start with a table of length 5 with all nodes having a single slot then n_1, n_2, n_3, and n_4 are schedulable, but n_0 is not. However, if the table length is increased to 6 with n_0 having two slots then all nodes are schedulable. Worst-case response times are given in Table I. These invariably occur following faults of the worst possible magnitude (as defined by the fault model). Note that no assumptions have been made about where in the table any node's particular slot(s) actually are positioned. We use a simple formulation of the analysis, the supply function for all nodes apart from n_0 is 1 in 7, 2 in 13, 3 in 19 etc. For n_0 it is 2 in 7, 4 in 13, 6 in 19, 8 in 25, 10 in 31 and 12 in 37.

To give an example of the analysis; consider τ_5 which is the lowest priority packet flow on node n_0. It is a HI-criticality flow but first its worst-case response-time in the LO-criticality mode must be computed. Using equations (3) and (4) we initially have $X = 3$ and $R_5(LO) = 13$. Now equation (3) becomes:

$$X = 3 \ + F_5(LO, 13) + \left\lceil \frac{13}{26} \right\rceil 1 + \left\lceil \frac{13}{64} \right\rceil 1$$

Name	From	To	Criticality	T	D	C	P	R
τ_1	n_1	n_2	LO	30	30	2	2	25
τ_2	n_1	n_0	LO	26	13	1	1	13
τ_3	n_2	n_0	HI	40	40	1	2	31
τ_4	n_2	n_0	LO	13	13	1	1	13
τ_5	n_0	n_4	HI	38	38	3	3	37
τ_6	n_0	n_4	LO	26	13	1	1	13
τ_7	n_0	n_1	HI	64	32	1	2	31
τ_8	n_3	n_4	LO	32	14	1	1	13
τ_9	n_3	n_0	HI	64	32	1	2	31
τ_{10}	n_3	n_0	LO	32	32	2	3	31
τ_{11}	n_4	n_0	HI	40	40	2	1	31

TABLE I
EXAMPLE, PARAMETERS AND RESPONSE-TIME CALCULATIONS

$F_5(LO, 13)$ is 2, since one table is corrupted within which there are two slots, hence X becomes 7 and $R_5(LO) = 25$. At this point, iteration has converged, as the value of X does not change when 13 is replaced by 25.

To compute $R_5(HI)$ we can start with a value of 25 so equation (5) becomes:

$$X = 3 \ + F_5(HI, 25) + \left\lceil \frac{25}{26} \right\rceil 1 + \left\lceil \frac{25}{64} \right\rceil 1$$

The fault load, $F_5(HI, 25)$ is now 6 (three tables corrupted), so $X = 11$ and $R_5(HI) = 37$. Another iteration gives:

$$X = 3 \ + F_5(HI, 37) + \left\lceil \frac{25}{26} \right\rceil 1 + \left\lceil \frac{37}{64} \right\rceil 1$$

which is again 11; so $R_5(HI)$ has converged to 37. Note the interval for interference from the LO-criticality packet flow τ_6 is capped at 25, which is the response-time of τ_5 in LO-criticality mode (i.e. $R_5(LO)$).

This example is used as the basis for the experiments undertaken on the prototype implementation of AirTight described in Section IX. In the prototype one slot per table was required for clock synchronisation. To keep the table size at 6, all the packet flows on node 0 would need to be schedulable with an allocation of only 1 in 5, but as illustrated above, an allocation of 1 in 5 is not sufficient (it requires 2 in 6). However, a simple modification to the example (extending τ_5's period and deadline to 55) does deliver a schedulable node with an allocation of 1 in 6. In this case, R_6 and R_7 remain unchanged (at 13 and 31 respectively), but R_5 is now 55.

IX. PROTOTYPE IMPLEMENTATION OF AIRTIGHT

The prototype CPS implementation is based on IEEE 802.15.4-compliant node hardware (the Iris XM-2110 nodes [18] manufactured by Crossbow Technology) and the TinyOS version 2 operating system [22] as obtained from the development repository in July 2017 [9]. This section describes the decisions we made to implement the protocol services defined in Section V.

The core of the MAC services are built on several TinyOS components from the tinyos-contrib repository [8]. Firstly, the Slotter, SlotterControl and FrameConfiguration interfaces, and GenericSlotter components are used to implement the

timeslot management service. The prototype implementation of the *time synchronisation* service uses one of the table slots to transmit a beacon frame, but this is an implementation decision and not a fundamental requirement for the protocol. For the application described in Section VIII, the slot table is 6 slots long; slot 1 is designated for time synchronisation (via a broadcast beacon frame) and the remaining 5 slots for AirTight application data. The Iris node RF230 radio transceiver can provide, to a transmitting node, a hardware acknowledgement of frames received correctly by the recipient node, which we used to implement the *acknowledged frame delivery* service.

The prototyped application is as described in Section VIII, although it is modified to operate with five data slots in the table, with the change of τ_5's period and deadline to 55 as described in the last paragraph of Section VIII.

The prototype implements the *node criticality level management* service, and its interaction with the priority-based frame scheduler, according to the following rules:

- M0 The counter of ACK failures, for the node, is initialised to zero. The node's mode is initialised to LO-criticality.
- M1 Every time a node fails to receive an acknowledgement for a transmission, its failed ACK counter is incremented.
- M2 When the failed ACK counter reaches a predefined value (which is 2 in the experiments described below), the criticality mode flag of the transmitting node is set to HI-criticality mode. On transitioning to the HI-criticality mode, the contents of LO-criticality buffers are discarded/flushed.
- M3 While in the HI-criticality mode, newly arriving LO-criticality transmission requests (for the transmission of new packets) are ignored.
- M4 While in the LO-criticality mode, the highest priority frame is selected for transmission when there is a slot available.
- M5 While in the HI-criticality mode, the highest priority HI-criticality frame is selected for transmission.
- M6 The node returns to its LO-criticality mode when all its HI-criticality buffers are empty. The ACK failure counter is also reset to zero.
- M7 If the failed ACK counter reaches a second predefined value (which is 4 in the experiments described below) then the system is deemed to have moved beyond its HI-criticality fault model – in this situation the prototype implementation flushes all buffers and resets the ACK count to zero and the node mode to LO-criticality.

A. Scenario and Data Gathering

Five nodes are configured as standard nodes to implement the AirTight protocol. One of the participating nodes (node 0) is connected to a computer (see Figure 3), while the remaining nodes receive their power either through onboard batteries or USB power via a programming board.

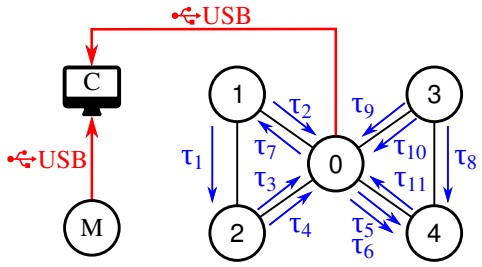

Fig. 3. Logical network structure for the experiments, showing monitoring node M and computer C

In order to gather necessary experimental data, an external node is connected as a monitoring node. This node operates in snooping mode which permits it to observe all the packets transmitted throughout the network. This monitoring node is entirely passive; it does not participate in data transmission via the AirTight protocol described in this paper, but merely listens constantly to the channel and records all of the AirTight packets observed. This is used during the experiments to collect performance statistics. During protocol operation, nodes maintain a running 16-bit counter of their slot number, and record both the slot number when each packet is generated, and the slot number of its transmission. This allows the response time of the received packets to be determined on the monitoring computer.

X. Simulation

In order to perform more extensive and controlled performance assessments for AirTight, an abstract simulation model of the protocol was developed. The simulator is a discrete-event simulator similar in concept to [23] which models the generation of application data, protocol rules and the transmission of individual frames across the network. The simulator provides a time slotted MAC protocol layer arranged in a repeating slot table similar to that described in Section IX. Although time synchronisation is not required due to the absence of any clock drift, one slot is also reserved to time synchronisation for consistency with the prototype implementation. Faults can be injected of varying length and periodicity, allowing the behaviour of the protocol under faults to be tested. The simulator provides a simple channel model with either guaranteed delivery between connected nodes, or discarding of the packets, depending on the fault model in use during an experiment.

For the simulated flow set and topology, two alternatives are available. The 5-node, 11-flow case as described in Section IX-A is simulated in the same configurations as the implementation. A larger case study was also created, consisting of five repeated copies of the standard topology, as depicted earlier in Figure 1.

XI. Experimentation

To perform an experiment, first the system nodes are programmed with the appropriate settings; for example: node

ID, slot configuration, and the parameters for their active packet flows. Then the listener node is activated and connected over USB. The listener node monitors the network in snooping mode, and the connected computer processes this information to produce the results, such as the flow latency between injection and transmission and the number of LO criticality flows which were discarded due to the protocol rules.

The results in the case with no explicit fault injection are presented in Figure 4. The background bars indicate the worst-case response-time of each packet flow as derived by the analysis given in Section VIII. The scatter plots indicate for each flow the distribution of timings (in slots) between injection by the application and transmission for each flow. The results are broken down first by per flow ID and, second by the number of retransmissions for each packet. The HI and LO criticality flows are grouped together. The data in Figure 4 is obtained from a single (though typical) experimental run lasting 5 hours.

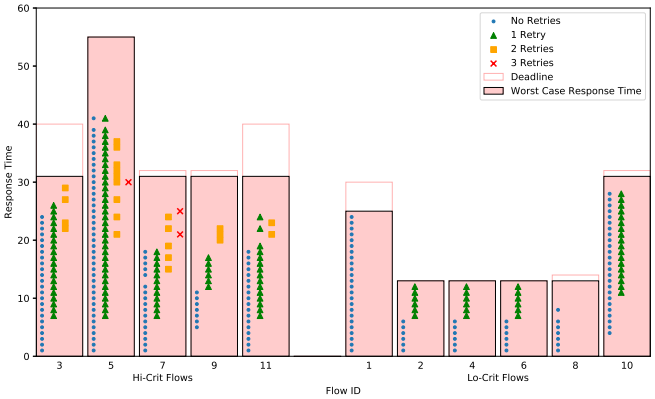

Fig. 4. Timing results with no deliberately injected faults (office environment)

The most important observation is that under these conditions, all HI-criticality and LO-criticality transmission times are below their analytically computed response time bounds. It is notable that even without the intentional injection of faults, occasionally some HI-criticality packets experienced multiple retransmissions. This occurs because of the office environment in which the experiments were undertaken. The frequency band used was subject to interference from wireless access points and other network users, thus some retransmissions were required as part of normal system operation. Although all transmitted flows met their deadlines, some LO-criticality flows were dropped (as the protocol requires in order to ensure the HI-criticality flow can have up to three re-tries). In the experiment reported in Figure 4 the six LO-criticality flows (1, 2, 4, 6, 8, and 10) have 0.19%, 0.34%, 0.90%, 3.88%, 0.21% and 0.83% of their flows dropped.

The simulation results in the presence of the controlled injection of faults are illustrated here showing simulated performance of the larger 25 node example in the presence of various levels of faults. These graphs contain less data per flow as there as now 55 flows.

As the 25 node system is in essence composed of 5 copies of the subsystem analysed earlier (though with periods and deadlines 5 times greater than those give in Table I), the analysis indicates that a slots table of length 30 delivers a schedulable system. Each simulation lasted for the hyper-period of the 55 flows (several hours of real-time, simulation time approximately 20 seconds).

We noted that in the presence of a single table fault repeated no closer than every 500 slots, all LO-criticality flows meet their deadlines and are within the analytically calculated response times, and all flows are delivered. In the presence of longer faults of up to 3 tables in duration (see Figure 5) some of the LO-criticality transmissions have to be discarded in order to meet the deadlines of the HI-criticality packets, although for the transmitted packets, all the deadlines are met. This is illustrated in Figure 5. In all, 17.18% of the LO-criticality flows were dropped.

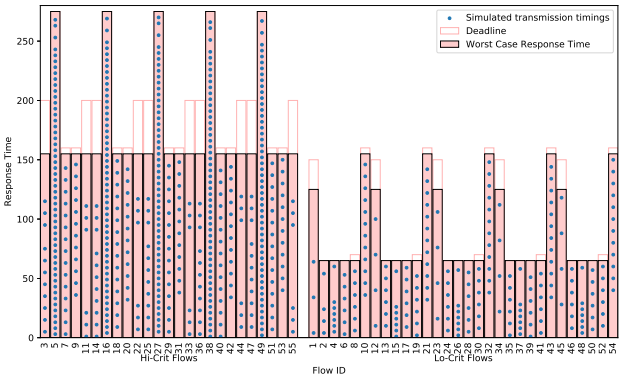

Fig. 5. Simulation – Faults within the HI-criticality Fault Model

XII. Conclusion and Future Work

In this paper we have demonstrated the feasibility of a protocol for wireless communication that is intended for use in time-critical cyber-physical systems. Various levels of faults (both in duration and frequency) are characterised within a fault model. Packet flows themselves are characterised by their criticality (as well as the usual parameters of deadline, size and minimum inter-arrival time). At run-time, performance is guaranteed in the sense that criticality and fault model are combined to give assurances of the form: if the faults experience by the system are below a certain level (as defined in the fault model) then all packets at or above a specified criticality level will be delivered by their deadlines.

The proposed AirTight protocol is motivated and defined in this paper, analysis is derived and a prototype implementation and protocol-accurate simulator are described. Experiments on the prototype and simulator demonstrate the feasibility of the protocol in practice and provide evidence of the soundness (and tightness) of the analysis.

Acknowledgements

The research described in this paper is funded, in part, by the EPSRC grants MCCps (EP/P003664/1). No new primary data were created during this study.

REFERENCES

[1] A. Alemayehu, L. George, V. Sciandra, and M. Agueh. Mixed criticality scheduling applied to jpeg2000 video streaming over wireless multimedia sensor networks. In *Proc/ of the Workshop on Mixed Criticality (WMC 2013)*, pages 55–60, 2013.

[2] N.C. Audsley. On priority assignment in fixed priority scheduling. *Information Processing Letters*, 79(1):39–44, 2001.

[3] N.C. Audsley, A. Burns, M. Richardson, K. Tindell, and A.J. Wellings. Applying new scheduling theory to static priority preemptive scheduling. *Software Engineering Journal*, 8(5):284–292, 1993.

[4] S.K. Baruah, A. Burns, and R. I. Davis. Response-time analysis for mixed criticality systems. In *Proc. IEEE Real-Time Systems Symposium (RTSS)*, pages 34–43, 2011.

[5] A. Burns and R.I. Davis. Mixed criticality on controller area network. In *Proc. Euromicro Conference on Real-Time Systems (ECRTS)*, pages 125–134, 2013.

[6] A. Burns, J. Harbin, and L.S. Indrusiak. A Wormhole NoC protocol for mixed criticality systems. In *Proc. IEEE Real-Time Systems Symposium*, pages 184–195. IEEE, 2014.

[7] A. Burns, S. Punnekkat, L. Strigini, and D.R. Wright. Probabilistic scheduling guarantees for fault-tolerant real-time systems. In *Proc. of the 7th International Working Conference on Dependable Computing for Critical Applications. San Jose, California*, pages 339–356, 1999.

[8] TinyOS contributing developers. Tinyos 2.x contributed code repository. https://github.com/tyll/tinyos-2.x-contrib, 2017. Master branch as of July 2017.

[9] TinyOS core developers. Tinyos development repository. https://github.com/tinyos/tinyos-main, 2017. Master branch as of July 2017.

[10] R.I. Davis and N. Navet. Traffic shaping to reduce jitter in controller area network (CAN). *SIGBED Review*, 9(4):37–40, 2012.

[11] A.C. Dimopoulos, G. Bravos, G. Dimitrakopoulos, M. Nikolaidou, V. Nikolopoulos, and D. Anagnostopoulos. A multi-core context-aware management architecture for mixed-criticality smart building applications. In *11th System of Systems Engineering Conference (SoSE)*, pages 1–6, 2016.

[12] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele. Low-power wireless bus. In *Proc. of the 10th ACM Conference on Embedded Network Sensor Systems*, pages 1–14, 2012.

[13] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh. Efficient network flooding and time synchronization with glossy. In *Proc. Information Processing in Sensor Networks (IPSN)*, pages 73–84, 2011.

[14] HART Communications Foundation. Document no 7.5: Hart communication protocol specification. Technical report, HARTComm, 2013.

[15] HART Communications Foundation. Iec 62591: Industrial networks - wireless communication network and communication profiles - WirelessHART (tm). Technical report, IEC Geneva, 2016.

[16] S. Gandham, M. Dawande, and R. Prakash. Link scheduling in wireless sensor networks: Distributed edge-coloring revisited. *Journal of Parallel and Distributed Computing*, 68(8):1122–1134, 2008.

[17] Z. He, K. Hewage, and T. Voigt. Arpeggio: A penetration attack on glossy networks. In *2016 13th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, pages 1–9, 2016.

[18] MEMSIC Inc. Iris wireless measurement system. www.memsic.com/userfiles/files/Datasheets/WSN/IRIS_Datasheet.pdf, 2011. Original document from Crossbow Technologies.

[19] L.S. Indrusiak, J. Harbin, and A. Burns. Average and worst-case latency improvements in mixed-criticality wormhole networks-on-chip. In *Proc. European/Euromicro Conference on Real-Time Systems (ECRTS)*, pages 47–56. IEEE, 2015.

[20] X. Jin, X. Changqing, J. Wang, and P. Zeng. Mixed criticality scheduling for industrial wireless sensor networks. *Sensors*, 16(9):1376, 2016.

[21] X. Jin, J. Wang, and P. Zeng. End-to-end delay analysis for mixed-criticality WirelessHART networks. *IEEE/CAA Journal of Automatica Sinica*, 2(3):282–289, 2015.

[22] P. Levis and D. Gay. *TinyOS Programming*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.

[23] S. Määttä, L. Möller, L.S. Indrusiak, L. Ost, M. Glesner, J. Nurmi, and F. Moraes. Joint validation of application models and multi-abstraction network-on-chip platforms. *IJERTCS*, 1(1):86–101, 2010.

[24] R. Mangharam, A. Rowe, R. Rajkumar, and R. Suzuki. Voice over sensor networks. In *27th IEEE International Real-Time Systems Symposium (RTSS'06)*, pages 291–302, 2006.

[25] R. Rajkumar, I. Lee, L. Sha, and J. Stankovi. Cyber-physical systems: The next computing revolution. In *Proceedings of the 47th Design Automation Conference*, DAC '10, pages 731–736, New York, NY, USA, 2010. ACM.

[26] A. Rowe, R. Mangharam, and R. Rajkumar. Rt-link: A time-synchronized link protocol for energy- constrained multi-hop wireless networks. In *3rd Annual IEEE Communications Society on Sensor and Ad Hoc Communications and Networks*, volume 2, pages 402–411, 2006.

[27] A. Saifullah. *Real-time wireless sensor-actuator network for cyber-physical systems*. PhD thesis, Department of Computer Science and Engineering, Washington University in St Louis, 2013.

[28] A. Saifullah, D. Gunatilaka, P. Tiwari, M. Sha, C. Lu, B. Li, C. Wu, and Y. Chen. Schedulability analysis under graph routing in WirelessHART networks. In *IEEE Real-Time Systems Symposium*, pages 165–174, 2015.

[29] A. Saifullah, Y. Xu, C. Lu, and Y. Chen. Real-time scheduling for WirelessHART networks. In *2010 31st IEEE Real-Time Systems Symposium*, pages 150–159, 2010.

[30] A. Saifullah, Y. Xu, C. Lu, and Y. Chen. End-to-end delay analysis for fixed priority scheduling in WirelessHART networks. In *17th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 13–22, 2011.

[31] W. Shen, T. Zhang, F. Barac, and M. Gidlund. Prioritymac: A priority-enhanced mac protocol for critical traffic in industrial wireless sensor and actuator networks. *IEEE Transactions on Industrial Informatics*, 10(1):824–835, 2014.

[32] S. Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *Proc. of the IEEE Real-Time Systems Symposium (RTSS)*, pages 239–243, 2007.

[33] X. Zhao, H. Gao, G. Zhang, B. Ayhan, F. Yan, C. Kwan, and J.L Rose. Active health monitoring of an aircraft wing with embedded piezoelectric sensor/actuator network: I. defect detection, localization and growth monitoring. *Smart Materials and Structures*, 16(4):1208, 2007.

[34] M Zimmerling, L Mottola, P Kumar, F Ferrari, and L. Thiele. Adaptive real-time communication for wireless cyber-physical systems. Technical report, ETH Zurich, 2016.