Event Notification in CAN-based Sensor Networks

Gedare Bloom, *Member, IEEE,* Gianluca Cena, *Senior Member, IEEE,* Ivan Cibrario Bertolotti, *Member, IEEE,* Tingting Hu, *Member, IEEE,* Nicolas Navet, and Adriano Valenzano, *Senior Member, IEEE*

***Abstract*—Preventive and reactive maintenance require the collection of an ever-increasing amount of information from industrial plants and other complex systems, like those based on robotized cells, a need that can be fulfilled by means of a suitable event notification mechanism. At the same time, timing and delivery reliability requirements in those scenarios are typically less demanding than in other cases, thus enabling the adoption of best-effort notification approaches. This paper presents, evaluates, and compares some of those approaches, based on either standard CAN messaging or a recently proposed protocol extension called CAN XR. In the second case, the combined use of Bloom filters is also envisaged to increase flexibility. Results show that the latter approaches are advantageous in a range of event generation rates and network topologies of practical relevance.**

***Index Terms*—Wired Sensor Networks, Industry 4.0, Industrial Internet of Things, Cyber-Physical Systems, Controller Area Network (CAN), Bloom Filter, CPAL**

I. Introduction and Related Work

MODERN, highly-automated industrial plants and their subsystems are becoming increasingly complex. To face competition, the time to market a new product from conception to mass production has to be as short as possible, which means that exhaustive testing of the plant as a whole before production starts is hardly possible. At the same time, plant downtime due to failures must be reduced drastically. As a consequence, preventive and reactive plant maintenance are becoming increasingly important and are going to assume a key role in Industry 4.0 [1]. The same techniques can readily be applied to other complex distributed systems, such as large intelligent vehicles [2] (trains, boats, heavy-duty vehicles or for precision agriculture). A recent trend in these scenarios is to collect large amounts of diagnostic information, including those seemingly not directly involved in equipment operation, so as to increase maintenance effectiveness by exploiting the knowledge acquired through big data analytics. This also enables automatic postmortem analysis upon failure events. Data collection is performed by means of a plant-wide distributed network of sensors. Importantly, it must be possible to retrofit this kind of network to an existing plant easily and inexpensively, to make it cope with this data-centric paradigm.

Wireless Sensor Networks (WSN) undoubtedly represent the most popular technology available to this purpose [3], [4]. Besides established solutions based on IEEE 802.15.4, others are making their way to industrial plants, like IPv6 over Time-Slotted Channel Hopping (6TiSCH) [5], which is considered a key enabler for the Industrial Internet of Things (IIoT) [6], [7]. Nevertheless, when higher performance or uninterrupted operation are required, wired solutions are probably unavoidable. Although these solutions are not intended as a general replacement for WSNs, they can profitably complement them.

Among the characteristics a wired sensor network should feature, there are: *a)* very low cost per node; *b)* inexpensive cabling and no active network equipment; *c)* ability to support a sizable number of nodes; *d)* extended coverage; *e)* possibility of carrying both data and power on the same cable. A strong candidate for the underlying transmission technology in such context is the Controller Area Network (CAN) [8], [9], which is one of the most widely used solutions in automotive applications and is adopted in many networked embedded systems as well. Interestingly, its medium access control mechanism, based on collision-free arbitration, also has been proposed for use on wireless networks [10].

The CAN protocol is especially suited to distributed event notification, because it inherently supports broadcast transmission and is optimized for transferring short payloads (up to 8 bytes) efficiently. As shown in Fig. 1, CAN addressing is based on tagging message contents with an *identifier* rather than relying on a node-based addressing scheme, a feature that expedites the design of producer-consumer systems and makes content-based, hardware-assisted message filtering possible. Being tied to data rather than communication endpoints, part of the identifier can also be used to deliver information if needed, further reducing communication overheads.

With respect to the above-mentioned wired sensor network requirements, we observe that: *a)* CAN controllers are very often embedded in modern microcontrollers and the cost of external transceivers and connectors is limited; *b)* in small networks, no network equipment at all is needed; *c)* it is pos-

This work was supported in part by the U.S. National Science Foundation (CNS Grant No 1646317) and U.S. Department of Homeland Security under Grant Award Number, 2017-ST-062-000003. Paper No. TII-18-3082. (Corresponding author: Tingting Hu)

G. Bloom is with the Howard University, Department of Electrical Engineering and Computer Science, DC 20059, Washington, United States (gedare@scs.howard.edu).

G. Cena, I. Cibrario Bertolotti, and A. Valenzano are with the National Research Council of Italy, Institute of Electronics, Computer and Telecommunication Engineering (CNR-IEIIT), I-10129 Turin, Italy ({gianluca.cena, ivan.cibrario, adriano.valenzano}@ieiit.cnr.it).

T. Hu and N. Navet are with the University of Luxembourg, Faculty of Science, Technology and Communication (FSTC), L-4364 Esch-sur-Alzette, Luxembourg ({tingting.hu, nicolas.navet}@uni.lu).

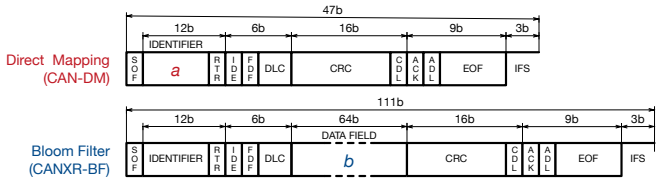

Fig. 1. Frame formats of the event notification methods (without bit stuffing).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2019.2904082, IEEE Transactions on Industrial Informatics

sible to connect up to 120 nodes to each network segment by means of high-impedance transceivers [11] and, if necessary, join multiple segments with simple multi-port repeaters [12]; *d)* a practically relevant network architecture using CAN with a bit rate $R = 50\,\mathrm{kb/s}$ (one of the bit rates recommended by CANopen [13]) allows a maximum distance between the farthest devices of up to $1.4\,\mathrm{km}$ in real scenarios [12] without impairing the quality of signals transmitted on the bus, which is enough for a relatively large system; *e)* a 4-wire cable is customarily employed to supply power to transceivers, but low-power sensors can also be powered from the same source.

With respect to conventional WSNs, a CAN-based sensor network (CSN) [14], [15] offers noticeably higher dependability because of the wired transmission support and robust MAC layer. Moreover, CAN easily supports differentiated services by enabling two concurrent classes of network traffic, namely critical and non-critical. High-priority CAN messages are reserved for safety- and time-critical data, possibly including alarms used for corrective maintenance. Network bandwidth left unused by critical data can be exploited for information related to preventive maintenance (including, e.g., non-critical warnings), which are mapped on low-priority messages and managed according to a best-effort approach.

A CSN can offer two different application-level interfaces, which differ by performance and features: either raw data are encapsulated directly into CAN frames or the IP protocol stack is layered above CAN [16]. In the first case, specific unified data encoding can be possibly adopted to achieve interoperability or to gather multiple signals in the same message (mapping and aggregation), whereas the second case is more faithful to the IIoT paradigm at the expense of some performance degradation. Since this paper aims at evaluating the raw performance that CAN offers to efficiently collect large amounts of data produced by distributed sensors, we will refer to the first case only.

Building on the preliminary ideas and results outlined in prior work [17], this paper evaluates and compares some methods for event notification in a CSN when the mean rate of randomly generated events is varied. To this extent, a simulation framework has been used, cross-checked by theoretical derivations. More specifically, a first class of methods is based on standard CAN messaging, whereas the second is based on the CAN with eXtensible in-frame Reply (CAN XR) proposal [18], which is used either to collect events by means of the combined message approach, or to compute Bloom filters [19] in a distributed manner and deliver them at the same time. Although evaluation has been performed taking classic CAN as a reference, all considered methods can benefit from the extended data field length that CAN FD frames support. It is worth noting that only simple theoretical upper bounds for the event notification rate were provided in [17], which are quite far from what can be actually obtained in the more realistic conditions we are considering here.

The paper is organized as follows: Section II highlights some contexts where the approaches based on CAN XR, including the peculiar one relying on Bloom filters, could be advantageous. Section III and IV, besides providing some background information and terminology, discuss the techniques being compared. Then, Section V describes CPAL [20], the modeling and simulation language used in the experiments, and presents the internal architecture of the simulator in detail. Last, Section VI analyzes and discusses results, and Section VII draws a conclusion.

II. Motivation and Application Fields

Overall, CAN behavior resembles a distributed priority queue: messages are inserted in the local transmission buffers of nodes and served by the network, one by one, according to their identifiers' order. Whenever the operating conditions force more events to be raised than the network is capable to drain, increased latencies are experienced, which could be possibly reduced by exploiting aggregation.

Prior work [17] postulates the use of CAN XR to gather multiple events in the same message. The simplest solution is to rely on a bitmap to this extent, but this makes the number of events that can be collected and sent at once strictly limited by the payload size of CAN frames. As shown in [17], such drawback can be overcome by employing Bloom filters. On the downside, false positives are known to be possibly generated this way. Although delays are generally perceived to be a lesser inconvenience than false positives, this is not always the case. Some examples are given below.

A. Non-Critical Event Notification

According to the IIoT paradigm, predictive maintenance is customarily modeled after a three-layer architecture [21], [22]. The *perception* layer includes sensors aimed at acquiring physical properties, while the *network* layer gathers all these pieces of information and conveys them to the *application* layer for further processing. A particularly relevant class of sensing devices raise events upon the occurrence of specific conditions, such as warnings indicating that some threshold has been exceeded. If CAN is employed at the network layer, each event triggers the transmission of a specific message. The sensor may optionally keep on sending messages while the warning condition stays active. Since the event generation pattern is typically unknown, the related network traffic is characterized by a large variability. This means that, depending on overall system conditions, transmission latency may occasionally increase noticeably and unexpectedly, which implies that no guarantees can be provided for delivery times. In these cases, graceful performance degradation is customarily sought.

For non-critical events, a higher notification rate provides finer granularity in the acquired logs and better diagnostic capabilities. Since interference of low-priority frames on the high-priority ones does not depend in CAN on the overall amount and rate of the former, it is in theory possible to run the network near its capacity, with non-critical messages scheduled essentially in the background. In practice, to make behavior for the lowest priority frames fairer, an inhibit time has to be set for them, which limits the generated traffic in advance. For instance, in a large network including one thousand sensors, setting the inhibit time to $1\,\mathrm{s}$ caps the overall event generation rate to $1\,\mathrm{kHz}$.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2019.2904082, IEEE Transactions on Industrial Informatics

Sometimes, CAN XR is advantageous over plain CAN. For example, if deep learning is applied to the acquired data, the property that event A is a direct cause of event B (and not vice-versa) can be inferred by determining how frequently B takes place shortly after A. If the system is characterized by fast dynamics (tens to hundreds of milliseconds, as in mechanical gears), reliably performing this check may be impractical when message reception times suffer from unpredictable delays. Conversely, this is feasible as long as timestamps are precise, and is not precluded by the occasional occurrence of false positives, as in the case when Bloom filters are used.

Another relevant case is related to heartbeat messages that are repeatedly sent by asynchronously communicating devices to notify they are alive. In CANopen [13], heartbeats are mapped on very low priority frames, so as to limit interference on process data. Using CAN XR to convey such events permits to decrease bandwidth consumption. Additionally, Bloom filters enable packing heartbeats of all nodes in a single frame with no specific drawbacks since, irrespective of the presence of occasional false positives, the complete lack of notifications from some node is detected by sinks sooner than later.

B. Critical Event Notification

Another application scenario for the considered notification methods concerns distributed critical systems with tight timing constraints or safety requirements, like those found in many industrial environments. Their correct behavior depends on the ability of the network to deliver timely messages between devices. In the case of CAN, feasibility analysis is customarily employed to ensure that deadlines are always met [23]. While this approach suits real-time process data, which are typically exchanged according to a periodic schedule, it may be inadequate for the safety-relevant ones, as their generation is often sporadic. At best, a minimum interarrival time can be defined for them, which is often orders of magnitude shorter than the average interarrival time. This implies that, in complex systems where many safety devices (light curtains, emergency stop push buttons, safety interlock switches, safe cameras, etc.) are producing infrequent events according to unpredictable patterns, feasibility analysis may be very ineffective.

Relying on CAN XR could be advantageous, as it reduces the required bandwidth. The ability to gather notifications can drastically reduce the message rate on the bus in critical conditions, hence making feasibility analysis effective again. This also holds when the overall number of safety events is quite large (hundreds) and Bloom filters are adopted to permit a variable number of them to be packed together. In fact, occasionally triggering a (useless) safety countermeasure upon a false positive is often a better option than experiencing a late reaction to conditions that may cause a severe hazard (e.g., a human operator being injured).

In the following, we mostly focused on non-critical data exchanges. However, most results also apply, with minimal changes, to safety-critical systems. In the latter case, feasibility analysis ensures that the deadlines of CAN XR messages used to collect safety-relevant events are never exceeded.

TABLE I
GLOSSARY

Symbol	Meaning
$a \in \mathcal{A}$	Event sources, $\lvert\mathcal{A}\rvert$ represents the number of sources
$s \in \mathcal{S}$	Nodes, $\lvert\mathcal{S}\rvert$ represents the number of nodes
λ_a	Generation rate of events belonging to a certain source a
Λ	Total mean event generation rate ($\Lambda < \Lambda_{\max}^{\mathrm{dm}}$ for *CAN-DM*)
R	CAN bit rate ($R = 50\,\mathrm{kb/s}$ in simulations)
L^{dm},L^{bm},L^{bf}	Length of *CAN-DM*, *CANXR-BM*, *CANXR-BF* frames
d^{dm},d^{bm},d^{bf}	Event notif. latency for *CAN-DM*, *CANXR-BM*, *CANXR-BF*
t_m	CAN XR message transm. time, $t_m = L^{\mathrm{bm}}/R = L^{\mathrm{bf}}/R$
b	Bloom filter of length m bits
h_0	FNV-1a master hash function
h_i	Family of k hash functions, $i = 1 \ldots k$
k_{opt}	Optimum k according to Bloom filters theory
k^*	Implementation-related upper bound on k
$\beta(b, a)$	Probabilistic membership check function of event a
$\mathcal{M}(a)$	Bloom filter mask of event a
P_{fp}	False positive probability of a single filter query
$\overline{N}_{\mathrm{fp}}$	Expected number of false positives, exhaustive check
$\mathcal{F}$	False positive ratio

III. Event Notification in CAN

A. System Model

The CSNs being considered here follow a producer-consumer model, in which a number of *sources* $a \in \mathcal{A}$, residing on a set of *nodes* $s \in \mathcal{S}$, spontaneously push *events* towards *sinks*, by means of CAN messages. For clarity, Table I summarizes the symbols introduced here and in the next sections. The main advantage of this approach with respect to polling is that, in order to exchange data, sinks do not need to know where sources are. This improves dependability and makes adding sources at runtime easier. Sources typically generate events according to signals acquired from the physical world by means of sensors. In this work, events are assumed to be unqualified, that is, not accompanied by any ancillary information. For instance, a source may generate an event when a certain signal exceeds a threshold. Multiple sources may reside on the same node s, which is responsible for sending messages through its network socket, thus modeling a modular device able to manage several signals. Without loss of generality, we will also assume that each source a generates its own unique event, at its own rate λ_a. Hence, a will be used to denote either a source or the event it generates, represented as an integer between 0 and $|\mathcal{A}| - 1$ included.

B. Event Mapping and Aggregation

The representation of events as integers allows the notion of *mapping* events onto messages and their possible *aggregation*, that is, storing and conveying multiple events in the same message. Reference [17] discussed the following approaches:

1) Event Bitmap: Multiple events generated within the same node s are aggregated in the message data field, which contains a bitmap. The message identifier is typically used to distinguish messages coming from different s, while a is used to statically determine the event position within the bitmap.

2) Event List: Event identifiers are encoded as a list in the data field, rather than being mapped to predetermined bit positions. An abbreviated version of a may be put in the list, as its scope is restricted by the message identifier.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2019.2904082, IEEE Transactions on Industrial Informatics

3) Direct Mapping (CAN-DM): Each event maps to a distinct message, with a in the identifier field and an empty data field. Events are not aggregated, hence this solution is optimal when every node produces only one event. *CAN-DM* is very flexible because its flat mapping does not require the pre-allocation of any specific a to any specific s. Moreover, it generates the shortest possible CAN messages.

Considering realistic systems, in which multiple event sources can be associated to the same node, *CAN-DM* remains the best candidate as long as the likelihood that a node generates two or more events at the same time is negligible. Otherwise, event bitmaps and event lists, which support aggregation, are more suitable. Although they offer a higher notification rate than *CAN-DM*, their behavior is basically the same. In fact, local in-node aggregation in CAN may not offer the same advantages as global network-wide aggregation supported by recent proposals like CAN XR. For the aforementioned reasons, and in order to simplify reasoning, *CAN-DM* has been selected as the baseline for analysis and comparison.

C. Direct Mapping of Events in CAN

With *CAN-DM*, the number of sources (events) $|\mathcal{A}|$ is limited by the width of the identifier field, $11\,\mathrm{b}$ for classical base frame format (CBFF) CAN frames [8]. Assuming the whole identifier space is allocated to event notification, it must be $|\mathcal{A}| \leq 2^{11} = 2048$. Similarly, the total mean event generation rate $\Lambda = \sum_{a\in\mathcal{A}} \lambda_a$ for the whole system is limited by the available network bandwidth. Since, as shown in the upper part of Fig. 1, the length of the shortest CBFF frame is $L^{\mathrm{dm}} = 47\,\mathrm{b}$, the maximum rate at which generated events can be transferred over the network, calculated as per [17], is $\Lambda^{\mathrm{dm}}_{\mathrm{max}} = R/L^{\mathrm{dm}} \simeq 1.06\,\mathrm{kHz}$ when $R = 50\,\mathrm{kb/s}$. When $\Lambda^{\mathrm{dm}}_{\mathrm{max}}$ is exceeded, network behavior becomes unstable and CAN arbitration rules prevent the transmission of lower-priority messages in favor of higher-priority ones. In this and the following calculations, bit stuffing has not been taken into account for simplicity, as the exact number of stuff bits varies depending on message contents. Moreover, bit stuffing is likely to affect all event notification approaches in a similar way.

An important metric for event notification is the *notification latency*, denoted d^{dm} for *CAN-DM*, that is, the time elapsing from event generation to delivery. This case can be suitably modeled, according to Kendall's notation, as a G/D/1 queue, where G is the generation process, D indicates deterministic service time (equal to the duration of an empty frame, $1/\Lambda^{\mathrm{dm}}_{\mathrm{max}}$), and there is one single server (the CAN bus). A simple expression for the mean latency $\overline{d}^{\mathrm{dm}}$ exists for an M/D/1 queue, in which event generation is a Poisson process:

$$\overline{d}^{\mathrm{dm}} = \frac{1}{2\mu}\cdot\frac{2-\rho}{1-\rho} = \frac{1}{2\Lambda^{\mathrm{dm}}_{\mathrm{max}}}\cdot\frac{2-\Lambda/\Lambda^{\mathrm{dm}}_{\mathrm{max}}}{1-\Lambda/\Lambda^{\mathrm{dm}}_{\mathrm{max}}}, \qquad (1)$$

where $\mu = \Lambda^{\mathrm{dm}}_{\mathrm{max}}$ is the service rate and $\rho = \Lambda/\Lambda^{\mathrm{dm}}_{\mathrm{max}}$ denotes the utilization. When $\Lambda \geq \Lambda^{\mathrm{dm}}_{\mathrm{max}}$, network capacity is no longer sufficient to convey all events, and $\overline{d}^{\mathrm{dm}}$ becomes unbounded, which implies system instability. This derivation has been used to validate simulation results obtained for this scenario, as shown in Fig. 5 and discussed in Section VI.

Summing up, the event generation rate λ_a indicates the occurrence frequency of event a, and events generated by all sources in the system contribute to the total mean event generation rate Λ. In the scope of this paper, CAN messages are used to deliver events from their source to the sinks, and R represents the bit rate of the CAN bus.

IV. CAN XR and Bloom Filters

CAN XR [18] extends CAN functionality by allowing multiple nodes to take part in the transmission of the same frame. More specifically, every frame transmission in CAN XR constitutes an atomic transaction and is started by an *initiator* node. Multiple *responders* are allowed to contribute their own data items to the frame's data field as it transits on the bus, while *consumers* observe the whole frame and have these data items at their disposal altogether. The initiator also supervises frame integrity, for instance, by inserting stuff bits and terminating the frame with a proper CRC and trailer. Since both the initiator and responders insert stuff bits based on the same bit sequence they observe on the bus during a frame transmission, stuff bits overlap perfectly.

It is worth remarking that nodes can play multiple roles with respect to the same frame. For instance, the initiator can also act as a responder, and then consume the whole frame. Although only XR-enabled nodes can act as initiators or responders, XR frames are compatible with the CAN frame format and can be received by ordinary controllers.

The data field of an XR frame is conceptually split into one or more slots, which are assigned to responders. Replies of responders can be either disjoint—because each of them owns an *exclusive* slot—or overlapping when they transmit within the same *shared* slot. In the second case, the resulting bit pattern on the bus is the bitwise AND among the bit patterns sent by all responders.

As pointed out in [17], CAN XR supports high-rate event notification by exploiting the combined message principle. The lower part of Fig. 1 shows that a CAN XR frame transporting a single shared slot (up to $64\,\mathrm{b}$ for CBFF) can be chosen for encoding events. In particular, irrespective of the node on which the related source resides, each event can be reserved a specific bit in the message payload, which is seen as a system-wide defined bitmap. We denote this method *CANXR-BM*. Unlike methods presented in Section III-B, which can merely aggregate events generated within the same node, this permits to aggregate events coming from anywhere in the network. If the largest CBFF frames are used, $|\mathcal{A}| = 64$, $L^{\mathrm{bm}} = 111\,\mathrm{b}$, and the maximum supported notification rate increases to $\Lambda^{\mathrm{bm}}_{\mathrm{max}} = |\mathcal{A}|\cdot R/L^{\mathrm{bm}} \simeq 28.83\,\mathrm{kHz}$ when $R = 50\,\mathrm{kb/s}$.

Unfortunately, a limited number of events are supported this way. This drawback can be addressed by defining a number of bitmaps, each of which is mapped on a distinct XR transaction, but doing so partitions the space of events in advance and reduces the chances of aggregation. Alternatively, the FD base frame format (FBFF) can be used, which conveys up to 64 bytes, hence increasing the maximum bitmap size to $512\,\mathrm{b}$.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2019.2904082, IEEE Transactions on Industrial Informatics

However, this enlarges the notification latency consequently and may impair feasibility analysis because of priority inversion phenomena. A third solution is to use probabilistic encoding, like Bloom filters, in the place of bitmaps.

A. Bloom Filters

A Bloom filter [19] is a data structure that can represent a set of elements a, events in this case, drawn from a possibly infinite universe $\mathcal{A}$, in a space-efficient manner. The data structure is optimized for time-efficient element insertion and probabilistic membership check. It consists of a fixed-length array b of m Booleans, denoted as $b(j),\ j = 0 \dots m-1$. Each Boolean is typically represented as a bit, whose values are written as 0 (false) and 1 (true) in the following. A family of k hash functions $h_i,\ i = 1 \dots k$, maps elements of $\mathcal{A}$ into sets of indices in b, that is, $h_i : \mathcal{A} \mapsto \{0 \dots m-1\}$. All bits of an empty filter are set to 0. The insertion of an element $a \in \mathcal{A}$ into a filter b takes place by setting all bits of b indicated by the above-mentioned hash functions when calculated in a to 1. Denoting the assignment operator as $\leftarrow$, insertion can be expressed as $\forall i\ \ b(h_i(a)) \leftarrow 1$.

In the following, with a slight abuse of notation, we will write $a \in b$ to mean that element a has been added to b as described above. A Boolean-valued function $\beta(b, a)$, given a Bloom filter b and an element a, returns 1 if a belongs to b, that is, $a \in b \rightarrow \beta(b, a)$. Informally speaking, membership check is performed by looking at all the bits of b indicated by $h_i(a)$, and yielding 1 if and only if they are all set. That is, $\beta(b, a) = \bigwedge_{i=1}^{k} b(h_i(a))$, where $\wedge$ denotes the Boolean AND operation. The reason why this is a *probabilistic* membership check becomes clear by contrasting element insertion and membership check. Due to the fact that typically $m \ll |\mathcal{A}|$, the collision probability of h_i cannot be zero and it may happen that $h_i(a) = h_{i'}(a')$ even though $a \neq a'$. It is therefore possible that the bits of b queried when checking a's membership have all been set by the insertion of other elements $a', a'', \dots \neq a$ into b. This gives rise to a *false positive* because $\beta(b, a)$ incorrectly indicates that a belongs to b, although it does not. If element deletions from a Bloom filter are forbidden, a similar reasoning leads to conclude that false negatives are impossible, because the bits of b are never reset to 0 after being set to 1 as a result of an insertion.

As shown in [24], assuming that the h_i are ideal, the false positive probability $P_{\text{fp}}(m, k, n)$ of $\beta(b, a)$ when applied to a Bloom filter of length m that makes use of k hash functions and contains n elements can be approximated by:

$$P_{\text{fp}}(m, k, n) \doteq P(\beta(b, a) \,|\, a \notin b) \approx \left(1 - e^{-\frac{kn}{m}}\right)^k. \qquad (2)$$

If m and n are known, it can also be proved that the optimal value of k, which minimizes P_{fp}, is $k_{\text{opt}} = \frac{m}{n} \ln(2)$. $P_{\text{fp}}(m, k, n)$ depends on k and the ratio n/m, but not on $|\mathcal{A}|$. Instead, $|\mathcal{A}|$ comes into play when calculating the expected number of false positives $\overline{N}_{\text{fp}}(m, k, n, |\mathcal{A}|)$ when a Bloom filter that truly contains n elements is exhaustively checked for the presence of all $a \in \mathcal{A}$. Individual checks are Bernoulli trials with a probability of false positive $P_{\text{fp}}(m, k, n)$ and $|\mathcal{A}| - n$ such trials may lead to a false positive, because the other n necessarily lead to a true positive. Therefore, the total number of false positives follows a Binomial distribution and $\overline{N}_{\text{fp}}(m, k, n, |\mathcal{A}|)$ is its expected value, that is:

$$\overline{N}_{\text{fp}}(m, k, n, |\mathcal{A}|) = (|\mathcal{A}| - n) \cdot P_{\text{fp}}(m, k, n). \qquad (3)$$

Note that the definition above pertains to a sink that is interested in all possible $a \in \mathcal{A}$. Sinks that only look for a subset of the events in $\mathcal{A}$ will encounter a proportionally lower number of false positives. On the other hand, non-ideal h_i (for instance, real-world hash functions with imperfect dispersion) obviously worsen P_{fp} and $\overline{N}_{\text{fp}}$.

Referring back to Fig. 1, a shared slot of an XR transaction mapped on a maximal-size CBFF frame, whose length is $L^{\text{bf}} = 111\,\text{b}$, is an ideal candidate to transport a Bloom filter b of length $m = 64$ for event notification, for instance according to the implementation described in Section IV-C. We will refer to this method as *CANXR-BF* in the following.

B. Comparison of Event Notification Methods

Similarly to CAN, event notification methods based on CAN XR follows the event-triggered paradigm rather than the time-triggered paradigm. Namely, any node with a pending event can start transmitting on the bus at the earliest opportunity, and no frame is sent if no event occurs. Operating in this way is more energy efficient, introduces less interference to other traffic on the same bus, and lowers the interrupt rate on sinks.

A noteworthy difference between approaches based on CAN and CAN XR is that, in the former case a implicitly determines event delivery priority through CAN bus arbitration. This can be used to provide certain events a higher quality of service (QoS) in terms of worst-case delivery latency and jitter than others, but it might lead to either unfair behavior or suboptimal performance when all events must be assured the same QoS. In the case of CAN XR, if a single frame is used to convey all events, uniform QoS is guaranteed for them, irrespective of the frame identifier, which can be chosen freely. Alternatively, multiple identifiers can be used to partition events into distinct QoS classes, at the expense of aggregation opportunities.

Unlike *CAN-DM*, the event notification latency when using *CANXR-BM* or *CANXR-BF* (d^{bm} and d^{bf}, respectively) depends on Λ only marginally, because with these approaches events are aggregated locally and kept within each node s until the next transmission opportunity, that is, for one frame transmission time at most. Then, they are further aggregated and delivered with the next frame transmission. As a consequence, $d^{\text{bm}} = d^{\text{bf}} \leq 2L^{\text{bf}}/R$ regardless of Λ. However, according to (3) a higher number of false positives may appear in *CANXR-BF* when Λ increases, since the number n of events aggregated in each transaction grows. This is because n reasonably coincides with the number of events generated while the bus is busy with the previous transaction. The main difference between notification methods lies therefore in how the system reacts to transients in which network capacity is exceeded. In *CAN-DM*, d^{dm} increases and, ultimately, events are dropped when $\Lambda \geq \Lambda^{\text{dm}}_{\text{max}}$ due to queue overflow, leading to *false negatives*. Event notification is instead always timely in *CANXR-BF*, because d^{bf} is bounded, but the number of *false*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2019.2904082, IEEE Transactions on Industrial Informatics

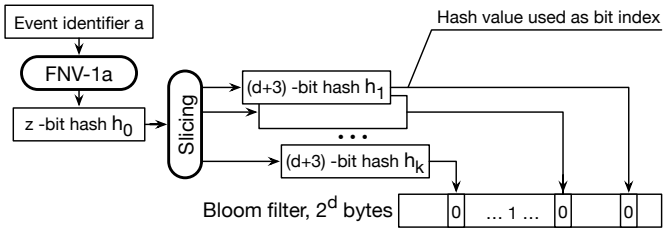

Fig. 2. Bloom filter implementation (in negative logic).

positives may increase noticeably. Finally, both false negatives and false positives are prevented by *CANXR-BM*.

Concerning the supported universe of events, its cardinality $|\mathcal{A}|$ in *CANXR-BM* is strictly bounded because of static encoding. Bloom filters in *CANXR-BF* do not pose, in principle, any bound on $|\mathcal{A}|$. However, as shown in (3), the expected number of false positives increases linearly with $|\mathcal{A}|$. Finally, bounds on $|\mathcal{A}|$ in *CAN-DM* depend on the size of the CAN identifier. They can be overcome by using the extended frame format, to the detriment of the notification rate.

Deciding which solution is better depends on the specific application context and, in particular, on whether worsening latency (culminating in false negatives) or generating false positives has the least impact on performance and correct operation. In the following, for space reasons, we will mainly focus on the two most antithetical solutions, that is, *CAN-DM* and *CANXR-BF*, in order to highlight differences in their behavior. Properties of *CANXR-BM* lie in between.

C. Implementation Details

The Bloom filter employed in the simulator has been realized as in Fig. 2, which illustrates an insertion operation.

1) Master hash function: The choice of hash function that lies at the base of the filter fell on the FNV-1a variant of the Fowler-Noll-Vo algorithm [25], [26]. FNV-1a is a byte-based iterative algorithm that, operating on input data a, calculates a z-bit hash denoted by $h_0(a)$, where z is a power of two, typically between 32 and 1024. Using a non-cryptographic hash function does not pose any issue in this application, while offering unquestionable advantages in terms of implementation complexity and execution performance.

2) Hashes derivation and validation: Function h_0 outputs a single, relatively large z-bit hash. Instead, as described in Section IV-A, Bloom filters need a family of k functions, and hashes must be in the $0 \dots m-1$ range. In general, if the Bloom filter must fit in a CAN XR data field of $D = 2^d$ bytes, it must be $m = 8 \cdot 2^d = 2^{d+3}$, which corresponds to a $(d+3)$-bit hash. An efficient method to determine h_i from h_0 is to *slice* the z-bit hash into chunks of $d+3$ bits, by means of shift and mask operations. By convention, h_1 is derived from the lowest-order bits of h_0 and h_k from the highest. The maximum k that can be obtained in this way is $k^* = \lfloor \frac{z}{d+3} \rfloor$.

For instance, $d = 3$ for a maximum-size CBFF frame. If $z = 64$, a sensible choice for processors with 64-bit integer arithmetic, this method can accommodate up to $k^* = 10$ hashes. In the simulations, the maximum admissible value $k = k^*$ has been used because, for the chosen value of m and the foreseen range of n, it is $k_{\text{opt}} \geq 10$. To validate the hash quality, the normalized entropy of h_i when $d = 3$ and $|\mathcal{A}| = 16000$ was calculated as a measure of dispersion. Experimental results show that, even though 2 iterations of the FNV-1a algorithm would nominally suffice to handle the value of $|\mathcal{A}|$ being considered—because it can be represented on 2 bytes—the dispersion of h_i for $i > 3$ would be unacceptably low. In the model, the issue has been addressed by padding the event identifier with zeros and incrementing the number of iterations to 8. Other, more complex methods of generating h_i, such as seeding the hash algorithm with a value derived from i, were not further considered in this work because, given the validation results just mentioned, they would not have significantly improved the hash quality.

3) Parallel implementation in negative logic: Although Section IV-A might suggest that Bloom filter insertion and membership check are serial, bit-by-bit operations, a parallel, register-based implementation is also possible provided the filter fits in a machine register, or can be easily manipulated by means of multiple-precision instructions. This happens, for instance, when $m \leq 64$ on processors with 64-bit integer arithmetic. In this case, each a can be associated with an m-bit mask $\mathcal{M}(a)$ in which a bit is set at one if and only if it is indicated by any $h_i(a)$, that is, $\mathcal{M}(a) = \bigvee_{i=1}^{k} M(h_i(a))$, where $\vee$ denotes the bitwise OR operation and $M(j),\ j = 0 \dots m-1$, is a function that returns a bit mask with the j-th bit set to 1. Accordingly, we can write:

$$b \leftarrow b \vee \mathcal{M}(a) \quad \text{(insertion) and} \quad (4)$$

$$\beta(b, a) = \begin{cases} 1 & \text{if } b \wedge \mathcal{M}(a) = \mathcal{M}(a) \\ 0 & \text{otherwise} \end{cases} \quad \text{(membership).} \quad (5)$$

Most importantly, even though Bloom filter theory is traditionally formulated in *positive* logic, as in Section IV-A, a *negative* logic implementation is also possible by means of a straightforward application of De Morgan's laws. With this approach, an empty filter b' is set to all 1 instead of all 0, and the bitwise NOT of $\mathcal{M}(a)$, denoted as $\overline{\mathcal{M}(a)}$, is used as a mask. Then, (4) and (5) become:

$$b' \leftarrow b' \wedge \overline{\mathcal{M}(a)} \quad \text{(insertion) and} \quad (6)$$

$$\beta(b', a) = \begin{cases} 1 & \text{if } b' \vee \overline{\mathcal{M}(a)} = \overline{\mathcal{M}(a)} \\ 0 & \text{otherwise} \end{cases} \quad \text{(membership).} \quad (7)$$

Conversion of the positive logic to negative logic results in the insertion function changing from bitwise OR to AND, which is efficiently implemented by the CAN bus, as the logic that the CAN bus itself represents is analogous to a "wired AND". More specifically, in the terminology of CAN, a dominant bit transmitted on the bus corresponds to logical 0 whereas a recessive bit corresponds to logical 1. When transmitted at the same time on the bus, the dominant bit wins. This implementation choice is of utmost importance to make event aggregation possible both *within a node s* and *across the network*, as hinted at in Section IV-A. This is because, according to (6), event insertion corresponds to a sequence of m-bit AND operations, which can then be performed in two phases:

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2019.2904082, IEEE Transactions on Industrial Informatics

1) By the processor of each node s, *before* CAN XR frame transmission, for all a generated within s, to build $|\mathcal{S}|$ partial filters that aggregate events node by node.
2) By the bus bit logic, *while* the CAN XR shared slot is being transmitted, to complete event aggregation across the whole network.

As a result, thanks to the associative property of AND, the Bloom filter eventually delivered to the sinks aggregates all events progressively inserted into it, regardless of their origin.

4) Execution time and memory overheads: The evaluation of execution time overhead focused on function h_0 because all the other operations just discussed (essentially, bitwise shifts and Boolean operations) are executed in one clock cycle or less on any modern processor, provided their operands fit in a register. The reference, public-domain C implementation of FNV-1a [25], when compiled for the ARM Cortex-M3 instruction set, using gcc 4.9.3 at optimization level O3 and the compiler's own 64-bit arithmetic code generator, calculates h_0 in 146 clock cycles. The execution time can be further improved by enabling function inlining, thus bringing the overhead down to 128 cycles in the test program. However, the performance gain of inlining is generally affected by the surrounding code. When considering a core frequency of $100\,\mathrm{MHz}$, which is typical of inexpensive microcontrollers like the NXP LPC1768 [27], these cycle counts correspond to an execution time between $1.28\,\mu\mathrm{s}$ and $1.46\,\mu\mathrm{s}$. As detailed in Section V-B, h_0 calculation is only used to prepare the masks $\mathcal{M}(a)$ corresponding to all events of interest a, an operation performed either offline or at system initialization time. Hence, the execution time overhead is likely to be acceptable in most applications.

The memory overhead associated with storing each $\mathcal{M}(a)$ is equal to the mask width m rounded up to a multiple of the machine word size, that is, 8 bytes when $m = 64$. The total size of the table holding all $\mathcal{M}(a)$ on each sink is linear in the number of events of interest to that sink. When implemented by means of a simple, sequential table scan, the execution time of a membership check applied to all events of interest is linear in the number of events, too. Notwithstanding, the membership check against individual elements of the table as in (5) or (7) is performed very quickly, because it only consists of a bitwise Boolean operation followed by a comparison.

V. CPAL Modeling and Simulation Architecture

In order to validate analytic results and facilitate the comparison of diverse event notification mechanisms, a simulator has been designed and implemented based on the Cyber-Physical Action Language (CPAL) introduced in Section V-A. The general architecture modeling both *CAN-DM* and *CANXR-BF* is depicted in Fig. 3 and discussed in Section V-B. Many relevant aspects about *CANXR-BM* behavior can be directly inferred from simulations related to *CANXR-BF*.

A. The CPAL Language, Runtime, and Multi-Interpreter

The modeling language and simulation environment used for the simulations is CPAL [20], which is a domain-specific language that supports the programming, graphical representation and simulation of embedded systems. CPAL is a high-level language based on Finite-State Machines (FSM), and other abstractions that are natural in the domain of embedded systems and communication protocols such as processes and communication channels with queue and stack semantics.

The concept of process is an abstraction at the core of the language. Process is a built-in type for a recurrent activity. Processes can be seen as functions that are activated in a time-triggered (e.g., periodically) or event-triggered manner (e.g., when certain activation conditions are met). Embedded in each process is an FSM, possibly reduced to a single state, to define the logic of the process in a non-ambiguous manner. A transition from one state to another state is triggered by a Boolean condition evaluating to true, or after a certain time has been spent in the state. When a process is activated, it resumes in the latest state it was in. First, a transition out of this state is taken if any can be, and only then the code of the current state is executed. The process can then yield the flow of control by terminating, or continue its execution.

In CPAL, functional and non-functional concerns are decoupled. The non-functional concerns, such as timing and scheduling, are expressed in annotations. For instance, it is possible through timing annotations to model dynamically changing execution times, transmission times, activation patterns, and execution jitters. This enables the designer to reproduce in simulation a timing realistic behavior of the model, early in the design phases. Another distinctive feature of CPAL is that the execution of CPAL models does not rely on code generation but on model interpretation: the models are interpreted by a lightweight real-time execution engine that can be hosted by an OS, or run on bare hardware on certain embedded platforms. The CPAL binaries and the documentation are freely available from https://www.designcps.com.

CPAL has been previously used for the simulation of communication protocols, such as for evaluating the performance of a service-oriented automotive middleware [28] and the precision of the Time-Triggered Ethernet clock-synchronisation protocol [29]. Because it is a domain-specific language, it leads to more readable and compact code than C, C++ or Java. For instance, the core of the CAN model used in this study is less than 200 lines long. Like typically done in Model-Driven Engineering flow, views can be extracted from the code of the models: the functional architecture (*i.e.*, processes and data-flows between processes), the automata defining the logic of the processes and a Gantt chart of the processes' executions.

The nodes in this study are asynchronous, in the sense that they possess their own local clock and their behavior is governed by independent processes. A modeling pattern for such asynchronous systems is to allocate each node to a dedicated interpreter and let the interpreters share information through communication channels. The use of multiple interpreters, one per node or, more generally, one per computational unit, is the *multi-interpreter* feature of CPAL. It provides discrete-event simulation with a local time on each interpreter and a global simulation time. The local times of the interpreters progress independently up to a time point where an event in the simulation necessitates a synchronisation point between the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2019.2904082, IEEE Transactions on Industrial Informatics

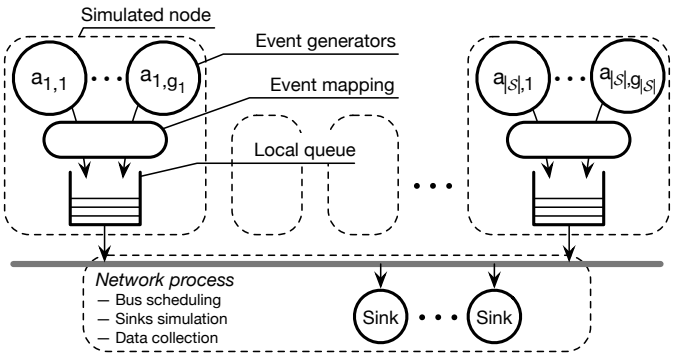

Fig. 3. General architecture of the CPAL-based simulator.

interpreters, such as a process reading input data and a process terminating and making updated data available to the other processes. At such time points, the input and output communication channels between interpreters are updated. Underlying, the multi-interpreter relies on a synchronous semantics where the processes executing on the interpreters can only read input data upon the start of their execution and write output data upon their termination, but not at any other times. Building on this synchronous semantics, the multi-interpreter simulation engine iteratively increases the global simulation time, by jumping to the next scheduled event on one of the interpreter, in such a way as to enforce the precedence constraints induced by the data flows between processes (e.g., no data can be read before it has been produced).

B. CPAL-based Simulator

As shown in Fig. 3, the modelled system based on CPAL consists of two kinds of component, namely the node model and the network model. Multiple nodes can be instantiated from the general node model and communicate with each other through the network.

1) Node model: A node s is associated with g_s event sources (denoted by $a_{s,1} \dots a_{s,g_s}$). A separate event generator is specified for each source $a_{s,l}$ and implemented as a CPAL process $P_{a_{s,l}}$. Events pertaining to $a_{s,l}$ are generated upon each execution of $P_{a_{s,l}}$ whose activation is controlled by a Poisson law, with a configurable event generation rate denoted as $\lambda_{a_{s,l}}$. This models memoryless systems, where the inter-arrival time between subsequent events of the same source is exponentially distributed. More specifically, besides generating an event, each process also determines its next activation time and notifies the interpreter by manipulating the per-process activation queue. In this way, the event generation time $t_{a_{s,l}}$ is translated to the activation time of $P_{a_{s,l}}$. It is worth remarking that, process execution takes zero time (regardless of its complexity) in the simulation mode unless otherwise specified with timing annotations. This guarantees that events are generated at the expected time.

Events generated within the same time interval by different sources will either be mapped onto individual frames or aggregated into a single frame, depending on the mapping and aggregation method in use. After that, frames are transmitted to the bus through the network interface, which is abstracted as a local queue. If *CAN-DM* is used, messages are built upon event generation, using $a_{s,l}$ as the frame identifier as shown in Fig. 1 and enqueued to the local queue for transmission.

For what concerns *CANXR-BF*, since the event sources associated to a node s are known a priori, their corresponding mask $\overline{\mathcal{M}(a_{s,l})}$ can be computed in advance and stored in a suitable data structure (e.g., a table), which permits fast access and reduces runtime overhead. In this way, the Bloom filter $b'_{s,l}$ for event $a_{s,l}$ can be obtained straightforwardly according to (6) upon event generation. Event aggregation within a node s is performed when a transmission starts as it is possible that events occur in the middle of an ongoing transmission. The node needs to wait for the current transmission to complete before initiating another one to deliver the incoming events. As a result, Bloom filters are enqueued directly.

2) Network model: The network model, represented as a CPAL process as well, implements bus scheduling, sinks simulation and data collection.

First of all, frame transmission can be initiated by any node with pending events when the bus is idle. This is emulated by modelling the network as an event-triggered process. As long as there is data available in local queues, the network process will be activated, given the previous transmission has concluded. If new events are supposed to be generated exactly at the same time, precedence is given to event generation, instead of network process activation. This can be achieved by proper configuration at the system level and then enforced by the CPAL multi-interpreter during simulation.

Upon execution, the network process scans through local queues and determines the message to be sent next. For CAN, it is the one with the highest priority among all local queues, which is determined by the frame identifiers, in turn by $a_{s,l}$. This simulates the arbitration mechanism in CAN. In the case of CAN XR, events currently available in the local queues will be aggregated into a single message, and its data field is constructed by following the two-step event aggregation illustrated in Section IV-C. If $b'_s,\ s = 1 \dots |\mathcal{S}|$ denotes the partially aggregated filter on each node, the final data field $\mathcal{D}$ is given by $\mathcal{D} = \bigwedge_{s=1}^{|\mathcal{S}|} b'_s$.

Since CAN is a broadcast medium, events observed on the bus are also observed by the sinks, given errors are outside the scope of this work. Hence, instead of modelling sinks explicitly, they are integrated as part of the network model. This optimization helps reduce the complexity of the model and improve simulation efficiency, without losing generality.

To facilitate post-analysis, two pieces of information are collected upon every execution of the network process, namely what events are detected on the bus and when events are delivered to the sink. Regarding CAN, the event can be retrieved directly from the frame header. Instead for CAN XR, $\mathcal{D}$ is checked against all possible Bloom filters in the system:

$$\forall s, \forall l \ \ \beta(\mathcal{D}, a_{s,l}) = \begin{cases} 1 & \text{if } \mathcal{D} \vee \overline{\mathcal{M}(a_{s,l})} = \overline{\mathcal{M}(a_{s,l})} \\ 0 & \text{otherwise} \end{cases}. \quad (8)$$

As discussed in Section IV-A, false positives are possible. In order to assess the false positive ratio $\mathcal{F}$, two counters $\mathcal{E}_d$ and $\mathcal{E}$ are used to keep record of the total number of events detected during a simulation run (updated upon every transmission),

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2019.2904082, IEEE Transactions on Industrial Informatics

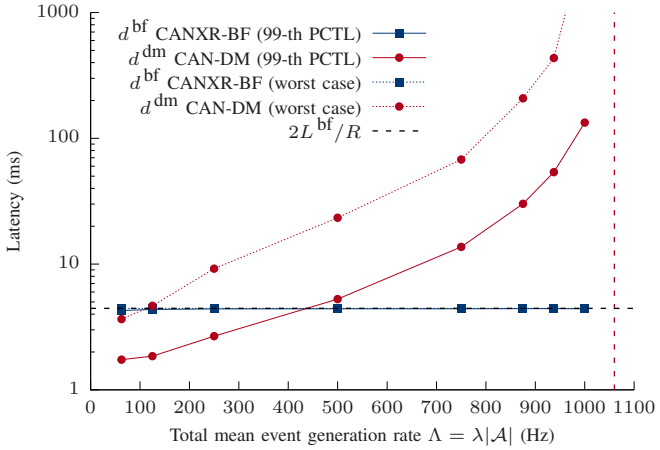

Fig. 4. Event notification latency comparison, $|\mathcal{A}| = 1000$ sources.

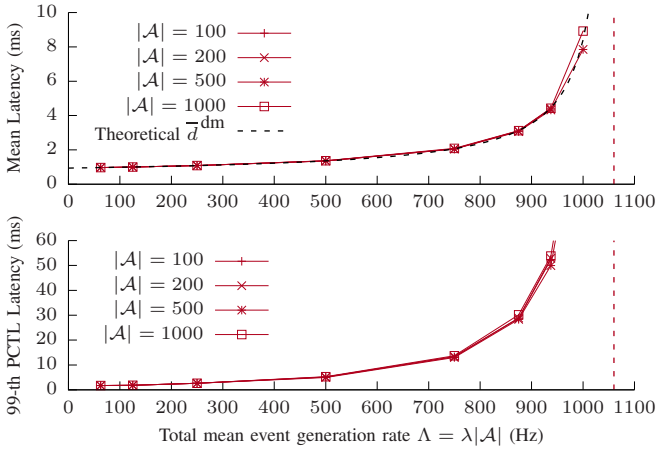

Fig. 5. *CAN + Empty data field* (*CAN-DM*) mean and 99-th percentile latency.

and the events truly generated as it is refreshed every time the network process iterates through all the local queues during event aggregation. Hence, $\mathcal{F} = (\mathcal{E}_d - \mathcal{E})/\mathcal{E}$.

The delivery time of an event $t'_{a_{s,l}}$ is marked by the time instant that the corresponding message transmission completes. Together with $t_{a_{s,l}}$, the event notification latency (d^{dm} or d^{bf}) can be calculated for each event. It is worth noting that the two timings just concern real events, rather than events being mis-reported.

3) System-level orchestration: The CPAL multi-interpreter provides direct support to specify nodes making up of a distributed system and data flows among them. Nodes are mapped to separate interpreters and communication among them are handled automatically by the multi-interpreter. Less than 60 lines of CPAL code are sufficient to set up the system depicted in Fig. 3 for simulation.

It is worth remarking that, independent time domains are maintained for each interpreter and synchronized with a common time base. The event generation time $t_{a_{s,l}}$ is mapped to the activation time of $P_{a_{s,l}}$ running on node s. Instead, the event delivery time $t'_{a_{s,l}}$ is calculated based on the activation time of the network process (which marks the beginning of a transmission), and the frame transmission time on the bus (that is L^{dm}/R for CAN and L^{bf}/R for CAN XR). The execution time of the network process is set to the frame transmission time by means of timing annotation. When transmission completes, the time notion of all interpreters will be updated to reflect time elapsed.

VI. Results and Analysis

In order to provide meaningful figures in the CSN scenarios we envisaged (e.g., predictive maintenance), simulations were performed running the CAN bus at $R = 50\,\text{kb/s}$. However, results do not actually depend on R. Better, generic plots can be provided for, e.g., the false positive ratio or the latency $d \cdot R$ normalized to the bit time versus the normalized generation rate λ/R. This means that considerations below also apply when faster CAN bit rates are taken into account, e.g., for in-vehicle systems where $R = 500\,\text{kb/s}$, which stretch over much smaller areas (overall cable length is below $100\,\text{m}$).

A. Notification Latency

As discussed in Sections III and IV, an important metric of any event notification system is its latency. Accordingly, a first round of simulations was carried out to compare the two selected notification methods from this point of view. Fig. 4 depicts, using a semi-logarithmic scale, both the worst-case value and 99-th percentile of d^{bf} and d^{dm}, with respect to the total mean event generation rate Λ, measured with $|\mathcal{A}| = 1000$ sources uniformly distributed across $|\mathcal{S}| = 100$ nodes, that is, $\forall s \;\; g_s = 10$. All sources have been configured to generate exponentially distributed events at the same rate $\lambda = \Lambda/|\mathcal{A}|$ (symmetric configurations are ordinarily assumed in literature). Moreover, in this and all the other experiments discussed in the following, the simulation time has been configured to collect $\mathcal{E} = 100000$ event instances starting at a time $t_0 = 180\,\text{s}$. This delay ensures that the simulation has reached a steady state before data collection begins.

As shown in the figure, d^{bf} remains *bounded* across the whole range of Λ considered in the simulations. More specifically, when using *CANXR-BF*, any event can indeed be delivered within $2L^{\text{bf}}/R$, as foreseen in Section IV-A. As long as the size of the frame payload is the same, notification latency d^{bm} of *CANXR-BM* coincides with *CANXR-BF*. In fact, in both cases all pending events are combined and sent at once (two different encoding schemes are used). By using a maximal-size FBFF frame, the number of supported sources in *CANXR-BM* can be increased from $|\mathcal{A}| = 64$ to $|\mathcal{A}| = 512$, which is comparable to simulations. By doing so, however, the frame size becomes $L^{\text{bm}} = 579\,\text{b}$ and latency increases to $2L^{\text{bm}}/R \simeq 23.16\,\text{ms}$. Note that, a blocking time in the order of $20\,\text{ms}$ may affect negatively other real-time messages.

On the contrary, due to queuing delays, d^{dm} grows more than linearly as Λ increases, even though individual message transmission times remain constant, and quickly becomes unacceptable as the limit $\Lambda^{\text{dm}}_{\text{max}}$ (shown in the figure as a vertical dashed line) is approached, as postulated in [17]. Additional tests were performed to analyze a possible dependency of d^{dm} on $|\mathcal{A}|$ when using CAN direct mapping. Eq. (1) rules out such a dependency for what concerns its mean value $\overline{d}^{\text{dm}}$, but

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2019.2904082, IEEE Transactions on Industrial Informatics

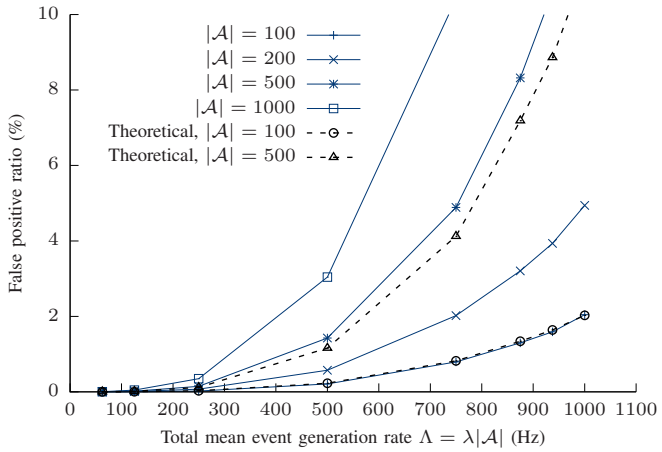

Fig. 6. *CAN XR + Bloom filter* (*CANXR-BF*) false positive ratio.

not for its distribution. Fig. 5 plots d^{dm} as a function of Λ for varying $|\mathcal{A}|$. The upper part of the figure shows $\overline{d}^{\text{dm}}$, while the lower part pertains to the 99-th percentile of d^{dm}. Experimental results confirm the absence of significant dependencies and, at the same time, offer the opportunity to further validate the simulator, because the measured $\overline{d}^{\text{dm}}$ completely matches the theoretical value given by (1), also reported in the figure as a dashed plot.

Results in Fig. 4 clearly indicate that, latency-wise, *CAN-DM* is advantageous when $\Lambda < 500\,\text{Hz}$. The main reason of this behavior is that empty CAN messages are much shorter than the CAN XR messages used to deliver the Bloom filter, namely, $L^{\text{dm}} \simeq 2/5\,L^{\text{bf}}$. As shown in the figure, when Λ approaches $0.95\,\text{kHz}$ the worst-case latency experienced in *CAN-DM* quickly reaches (and exceeds) $500\,\text{ms}$. This corresponds to about 500 times the CAN frame transmission time (L^{dm}/R), and may be unacceptable in several practical cases. Generally speaking, when Λ is small queuing delays are negligible even without performing event aggregation, and message transmission time dominates the latency. On the other hand, when Λ grows, the queuing delay of *CAN-DM* grows according to (1), whereas for methods based on CAN XR it remains constant because of event aggregation, as discussed in Section IV-C.

B. False Positive Ratio

A second set of experiments was aimed at assessing the false positive ratio $\mathcal{F}$ of *CANXR-BF*, measured as a function of Λ and $|\mathcal{A}|$, as described in Section V-B. This is a peculiar characteristic of this method, which *CAN-DM* and *CANXR-BM* do not exhibit. As specified in Section IV-A, simulations have been carried out assuming that the sink is looking for any possible $a \in \mathcal{A}$, which leads to the worst false positive ratio for that sink. As $h_i(a)$ are computed offline, seeking individual events in *CANXR-BF* has about the same complexity as performing frame filtering on identifiers in *CAN-DM*. Hardware assisted approaches can be envisaged in CAN XR, in the same way as done in CAN with filter ID and mask.

Results are summarized in Fig. 6, in which the *measured* false positive ratio $\mathcal{F}$, expressed as a percentage of the total number of events generated in the simulation, is shown. Values of $\mathcal{F}$ above 10% have not been displayed to improve readability. Systems exhibiting such a high false positive ratio are hardly useful in practice although, as shown in Fig. 4, d^{bf} still stays constant because m is constant.

The simulation results shown in Fig. 6 confirm both the complex dependency of $\mathcal{F}$ on Λ through P_{fp} and n, according to (3) and (2), as well as the linear dependency on $|\mathcal{A}|$ expressed by (3). It is also worth remarking that, when $\Lambda = 500\,\text{Hz}$, that is, at the latency break-even point between the two methods shown in Fig. 4, it is $\mathcal{F} \leq 3\%$, even with the maximum number of sources being considered, $|\mathcal{A}| = 1000$.

By reducing the number of event sources, it is possible to keep $\mathcal{F}$ in the same range as given above even for much higher values of Λ, for instance, up to $1\,\text{kHz}$ when $|\mathcal{A}| = 100$. It is worth pointing out that $|\mathcal{A}| = 100$ is still above what a bitmap in *CANXR-BM* is able to support, that is, at most 64 distinct events. Although mapping XR transactions on CAN FD enables larger bitmaps to be used, Bloom filter performance would also benefit from a larger m, thus supporting a higher $|\mathcal{A}|$ with an acceptable $\mathcal{F}$. Hence, especially for large values of $|\mathcal{A}|$, the maximum tolerable $\mathcal{F}$ is likely going to be the limiting factor on the maximum Λ that *CANXR-BF* is reliably able to support.

By observing both Fig. 6 and 5, we can conclude that reducing $|\mathcal{A}|$ improves the performance of *CANXR-BF*, whereas it does not affect *CAN-DM*. This further confirms the advantage of *CANXR-BF* concerning reactivity in the range $500 \leq \Lambda \leq 1000\,\text{Hz}$ for low values of $|\mathcal{A}|$. It should be noted that, in the same range of Λ, if $|\mathcal{A}|$ is small enough to fit on a single frame then *CANXR-BM* becomes the optimal solution.

C. Validation of CANXR-BF Behavior

It can be proved that $\mathcal{N}$, a discrete random variable with support $[1, +\infty[$ that represents the number of events collected in the same message when notification methods based on CAN XR are used (both *CANXR-BM* and *CANXR-BF*), follows a Poisson distribution with mean Λt_m, in which the first two bins have been grouped. Hence, the probability mass function (pmf) of $\mathcal{N}$ is:

$$\begin{aligned} P[\mathcal{N} = 1] &= \mathcal{P}(0; \Lambda t_m) + \mathcal{P}(1; \Lambda t_m) \\ &= e^{-\Lambda t_m}(1 + \Lambda t_m), \end{aligned} \tag{9}$$

$$P[\mathcal{N} = n] = \mathcal{P}(n; \Lambda t_m) = e^{-\Lambda t_m}\frac{(\Lambda t_m)^n}{n!}, \quad n > 1, \tag{10}$$

where $\mathcal{P}(\cdot)$ is the Poisson pmf and $t_m = L^{\text{bm}}/R = L^{\text{bf}}/R$ is the message transmission time. The mean number of events per message with CAN XR can be generically expressed as:

$$E[\mathcal{N}] = \sum_{n>0} nP[\mathcal{N} = n] = \Lambda t_m + e^{-\Lambda t_m}. \tag{11}$$

The *expected* false positive ratio in *CANXR-BF* is given by:

$$\hat{\mathcal{F}} = \frac{\sum_{n>0} P[\mathcal{N} = n]\,\overline{N}_{\text{fp}}(m, k, n, |\mathcal{A}|)}{E[\mathcal{N}]}, \tag{12}$$

which can readily be calculated because $\overline{N}_{\text{fp}}(m, k, n, |\mathcal{A}|)$ is given by (3) while m, k, and $|\mathcal{A}|$ are simulation parameters.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2019.2904082, IEEE Transactions on Industrial Informatics

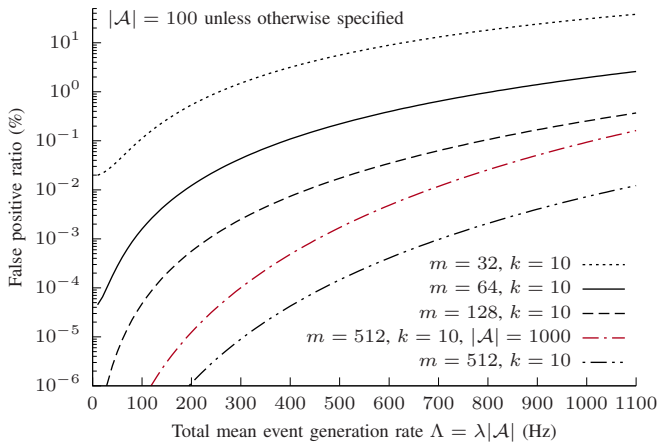

Fig. 7. Relationship between *CANXR-BF* false positive ratio and m.

The theoretical prediction of the false positive ratio given by (12) has been included in Fig. 6 for comparison with simulations. As can be seen, experimental results match very accurately the prediction for $|\mathcal{A}| = 100$. When using a higher number of event sources, for instance, $|\mathcal{A}| = 500$, the higher than expected false positive ratio can easily be explained by remembering that (2) assumes that h_0 is a perfect hash function. This is not true in practice, and the discrepancy becomes more evident as the size of the universe $\mathcal{A}$ the function has to work on increases. The histogram of the number of messages holding n true events collected during the simulations provided a further way to cross-validate the correctness of (9) and (10). Namely, experimental results matched the theoretical prediction within $\pm 0.3\%$ across the whole set of simulation runs.

Fig. 7 shows the *CANXR-BF* false positive ratio $\hat{\mathcal{F}}$, calculated according to (12), as a function of m, in the range of Λ considered in the previous experiments and with $|\mathcal{A}| = 100$. The solid line in the figure acts as a reference and corresponds to the circled, dashed line in Fig. 6, drawn on a semi-logarithmic graph for easier reading. As expected, halving the Bloom filter size by setting $m = 32$ increases the false positive probability by about two orders of magnitude and makes this choice suitable only for small values of Λ, as shown by the top-most dotted line in Fig. 7. For instance, it must be $0 \leq \Lambda \leq 200\,\text{Hz}$ to keep the false positive ratio below 1%. Although the presence of m in the equation of k_{opt} (see Section IV-A) might suggest that a value of k lower than 10 would lead to better results, $k = 10$ still leads to a significantly lower false positive probability in the range of Λ of practical interest. This is because k_{opt} also depends on n and, according to (11), $E[\mathcal{N}]$ shrinks as Λ decreases.

Further increasing m is also possible. As shown by the dashed and dot-dashed lines in Fig. 7, concerning the cases $m = 128$ and $m = 512$, respectively, this leads to a significant improvement in the false positive ratio although, in this case, k cannot be increased because it is capped by k^*. In turn, k^* is fixed by the hash generation algorithm (see Section IV-C2). In particular, when $m = 512$ also the case $|\mathcal{A}| = 1000$ (in red) results in a quite low $\hat{\mathcal{F}}$. Overall, (12) provides a convenient way of predicting the false positive ratio of *CANXR-BF*.

Note that the transmission of a Bloom filter with $m > 64$ requires either the use of a CAN FD frame or the fragmentation of the filter into multiple classical CAN messages. In both cases, there is an adverse impact on d^{bf} that shall be taken into account when dimensioning the system. In the most general conditions, conventional CAN schedulability analysis can be used to assess the worst-case event delivery latency depending on the other possible sources of traffic in the system.

VII. Conclusion

This paper compared some event notification methods, suitable for CAN-based sensor networks, by a combination of theoretical analysis and simulation-based measurements. In particular, *CAN-DM* is based upon the standard CAN protocol and encodes the event identifier in the CAN's identifier field, leaving the message payload empty. Instead, *CANXR-BM* and *CANXR-BF* take advantage of the in-frame reply feature of the CAN XR protocol to aggregate multiple events, built and delivered in the same message payload, by using a bitmap and a Bloom filter, respectively. These methods are well suited, for instance, to gather diagnostic information and warnings sporadically generated by nodes in a distributed system. Such events can be profitably conveyed as background, low-priority traffic, using the bandwidth left available by cyclic real-time exchanges. It is worth pointing out that, because of their unpredictable generation pattern, the bandwidth at disposal may be occasionally exceeded. Another interesting application field of above methods is related to safety-critical events. In this case, however, bounded delivery times shall be guaranteed for them, even though their occurrence is basically unforeseeable.

Results show that, for CAN busses running at $R = 50\,\text{kb/s}$, methods based on CAN XR compare favorably with respect to *CAN-DM* from the point of view of event notification latency when the total event generation rate is $\Lambda \geq 500\,\text{Hz}$. Above the break-even point, the latency with *CAN-DM* increases rapidly due to queuing delays, whereas with *CANXR-BM* and *CANXR-BF* it is bounded and remains below an acceptable value. Among the solutions based on CAN XR, *CANXR-BM* is likely the most appropriate in typical operating conditions. Nevertheless, *CANXR-BF* is able to accommodate a variable (and potentially larger) number of events. Moreover, unlike bitmaps, no static allocation is required in advance for Bloom filters, which tangibly lowers configuration efforts in large sensor networks. All that is needed is that event identifiers are network-wide unique, as for frame identifiers in CAN.

Timeliness in high traffic conditions shown by *CANXR-BF* comes at the expense of a certain number of false positive indications, which are the counterpart of *CAN-DM*'s false negatives due to excessive notification latency and the limitations of *CANXR-BM* on the number of event sources. Therefore, *CANXR-BF* is a valid alternative when: a) deterministic and low notification latency is demanded in a system characterized by many event sources (hundreds to thousands), b) variability of event generation may cause Λ to occasionally grow beyond the expected limit, and c) the application at hand can tolerate false positives better than false negatives. It is worth remarking

that modern Bloom filter designs [30] may reduce and, in some cases, completely avoid false positives. Their applicability to *CANXR-BF* has not been explored for the time being.

Last, but not least, unlike techniques based on legacy CAN where only local in-node event aggregation is performed, none of the considered notification methods depend on how the $|\mathcal{A}|$ event sources are distributed across the $|\mathcal{S}|$ nodes, that is, on the values of g_s. In the end, deciding the best option definitely depends on the specific application context.

As part of our future work, we plan to investigate more efficient event notification methods that, given the limits on channel capacity, try to set specific tradeoffs on performance metrics, for instance, latencies and false notification rates.

REFERENCES

[1] J. Wan, S. Tang, D. Li, S. Wang, C. Liu, H. Abbas, and A. V. Vasilakos, "A manufacturing big data solution for active preventive maintenance," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 4, pp. 2039–2047, Aug. 2017.

[2] C. P. Ward, P. F. Weston, E. J. C. Stewart, H. Li, R. M. Goodall, C. Roberts, T. X. Mei, G. Charles, and R. Dixon, "Condition monitoring opportunities using vehicle-based sensors," *Proceedings of the Institution of Mechanical Engineers, Part F: Journal of Rail and Rapid Transit*, vol. 225, no. 2, pp. 202–218, 2011.

[3] B. Lu and V. C. Gungor, "Online and remote motor energy monitoring and fault diagnostics using wireless sensor networks," *IEEE Transactions on Industrial Electronics*, vol. 56, no. 11, pp. 4651–4659, Nov. 2009.

[4] O. Kreibich, J. Neuzil, and R. Smid, "Quality-based multiple-sensor fusion in an industrial wireless sensor network for MCM," *IEEE Transactions on Industrial Electronics*, vol. 61, no. 9, pp. 4903–4911, Sep. 2014.

[5] D. Dujovne, T. Watteyne, X. Vilajosana, and P. Thubert, "6TiSCH: deterministic IP-enabled industrial internet (of things)," *IEEE Communications Magazine*, vol. 52, no. 12, pp. 36–41, Dec. 2014.

[6] G. Choudhary and A. K. Jain, "Internet of Things: A survey on architecture, technologies, protocols and challenges," in *Proc. International Conference on Recent Advances and Innovations in Engineering (ICRAIE)*, Dec. 2016, pp. 1–8.

[7] F. Tao, J. Cheng, and Q. Qi, "IIHub: an Industrial Internet-of-Things hub towards smart manufacturing based on cyber-physical system," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 5, pp. 2271–2280, 2018.

[8] ISO, *ISO 11898-1:2015 – Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signalling*, International Organization for Standardization, Dec. 2015.

[9] B. Andersson, N. Pereira, W. Elmenreich, E. Tovar, F. Pacheco, and N. Cruz, "A scalable and efficient approach for obtaining measurements in CAN-based control systems," *IEEE Transactions on Industrial Informatics*, vol. 4, no. 2, pp. 80–91, May 2008.

[10] N. Pereira, B. Andersson, E. Tovar, and A. Rowe, "Static-priority scheduling over wireless networks with multiple broadcast domains," in *Proc. 28th IEEE International Real-Time Systems Symposium (RTSS)*, 2007, pp. 447–458.

[11] *SN65HVD233-HT 3.3V CAN Transceiver Data Sheet*, Texas Instruments Inc., Jan. 2015.

[12] *CAN-CBM I or Y Repeater — Hardware Manual*, Esd electronics, Inc., Jul. 2003.

[13] CiA, *CiA 301 V4.2.0 – CANopen application layer and communication profile*, CAN in Automation e.V., Feb. 2011.

[14] M. A. Pillai, S. Veerasingam, and D. Y. Sai, "CAN based smart sensor network for indoor air quality monitoring," in *Proc. 3rd International Conference on Computer Science and Information Technology*, vol. 8, Jul. 2010, pp. 456–460.

[15] K. C. Lee and H.-H. Lee, "Network-based fire-detection system via controller area network for smart home automation," *IEEE Transactions on Consumer Electronics*, vol. 50, no. 4, pp. 1093–1100, Nov. 2004.

[16] G. Cena, I. Cibrario Bertolotti, T. Hu, and A. Valenzano, "Seamless integration of CAN in intranets," *Computer Standards & Interfaces*, vol. 46, pp. 1–14, May 2016.

[17] G. Bloom, G. Cena, I. Cibrario Bertolotti, T. Hu, and A. Valenzano, "Optimized event notification in CAN through in-frame replies and Bloom filters," in *Proc. 13th IEEE International Workshop on Factory Communication Systems (WFCS)*, May 2017, pp. 1–10.

[18] G. Cena, I. Cibrario Bertolotti, T. Hu, and A. Valenzano, "CAN with extensible in-frame reply: Protocol definition and prototype implementation," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 5, pp. 2436–2446, Oct. 2017.

[19] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, Jul. 1970.

[20] N. Navet and L. Fejoz, "CPAL: High-level abstractions for safe embedded systems," in *Proc. ACM International Workshop on Domain-Specific Modeling (DSM)*, Oct. 2016, pp. 35–41.

[21] M. R. Abdmeziem, D. Tandjaoui, and I. Romdhani, "Architecting the internet of things: State of the art," in *Robots and Sensor Clouds*, A. Koubaa and E. Shakshuki, Eds. Cham: Springer International Publishing, 2016, pp. 55–75.

[22] G. Bloom, B. Alsulami, E. Nwafor, and I. Cibrario Bertolotti, "Design patterns for the industrial Internet of Things," in *Proc. 14th IEEE International Workshop on Factory Communication Systems (WFCS)*, Jun. 2018, pp. 1–10.

[23] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien, "Controller area network (CAN) schedulability analysis: Refuted, revisited and revised," *Real-Time Syst.*, vol. 35, no. 3, pp. 239–272, Apr. 2007.

[24] S. Tarkoma, C. E. Rothenberg, and E. Lagerspetz, "Theory and practice of Bloom filters for distributed systems," *IEEE Communications Surveys & Tutorials*, vol. 14, no. 1, pp. 131–155, 2012.

[25] G. Fowler, L. C. Noll, K.-P. Vo, D. Eastlake, and T. Hansen, "The FNV non-cryptographic hash algorithm," Internet Engineering Task Force, Internet-Draft, Jun. 2017.

[26] L. Chi and X. Zhu, "Hashing techniques: A survey and taxonomy," *ACM Comput. Surv.*, vol. 50, no. 1, pp. 11:1–11:36, Apr. 2017.

[27] *LPC17XX User manual, UM10360 rev. 2*, NXP B.V., Aug. 2010.

[28] J. R. Seyler, T. Streichert, M. Glaß, N. Navet, and J. Teich, "Formal analysis of the startup delay of SOME/IP service discovery," in *Proc. Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2015, pp. 49–54.

[29] L. Fejoz, B. Régnier, P. Miramont, and N. Navet, "Simulation-based fault injection as a verification oracle for the engineering of time-triggered Ethernet networks," in *Proc. Embedded Real-Time Software and Systems (ERTSS)*, Jan. 2018, pp. 1–10.

[30] S. Z. Kiss, É. Hosszu, J. Tapolcai, L. Rónyai, and O. Rottenstreich, "Bloom filter with a false positive free zone," in *Proc. IEEE International Conference on Computer Communications (INFOCOM)*, Apr. 2018, pp. 1–9.

Gedare Bloom (M'08) received his Ph.D. in computer science from The George Washington University in Washington, D.C., in 2013. He joined the Department of Computer Science at Howard University as Assistant Professor and founding director of the Embedded Systems Security Lab in 2015. His research expertise is computer system security with particular focus on real-time embedded systems, and he has published over twenty peer-reviewed articles in these areas. The techniques he applies to solve problems along the hardware-software interface range from computer architecture, computer security, cryptography, operating systems, and real-time analysis. He has served as a program committee member, artifact evaluation committee member, and technical referee for flagship conferences and journals in the area of real-time systems.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2019.2904082, IEEE Transactions on Industrial Informatics

Gianluca Cena (SM'09) received the Laurea degree in Electronic Engineering and the Ph.D. degree in Information and System Engineering from Politecnico di Torino, Italy, in 1991 and 1996, respectively. Since 2005 he has been a Director of Research with the Institute of Electronics, Computer and Telecommunication Engineering, National Research Council of Italy (CNR-IEIIT), Turin. His research interests include wired and wireless industrial communication systems, real-time protocols, and automotive networks. In these areas he has co-authored about 140 papers and one patent. He received the Best Paper Award of the IEEE Workshop on Factory Communication Systems in 2004, 2010, and 2017, and the Best Paper Award of the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS in 2017.

Dr. Cena served as Program Co-Chairman of the IEEE Workshop on Factory Communication Systems in 2006 and 2008, and as Track Co-Chairman in six editions of the IEEE Conference on Emerging Technologies and Factory Automation. Since 2009 he has been an Associate Editor of the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS.

Ivan Cibrario Bertolotti (M'06) received the Laurea degree *(summa cum laude)* in computer science from the University of Torino, Turin, Italy, in 1996.

Since then, he has been a Researcher with the National Research Council of Italy (CNR), Rome, Italy. Currently, he is with the Institute of Electronics, Computer and Telecommunication Engineering (IEIIT), Turin. He has taught several courses on real-time operating systems at Politecnico di Torino, Turin; has co-authored two books on the same topics; and serves as a Technical Referee for primary international journals and conferences. His research interests include real-time operating system design and implementation, industrial communication systems and protocols, as well as modeling languages and runtime support for cyber-physical systems. He received, as a coauthor, the Best Paper Award presented at the 8th IEEE Workshops on Factory Communication Systems (WFCS 2010).

Tingting Hu (M'11) received her master degree in Computer Engineering in 2010 and Ph.D. degree with the best dissertation award in Computer and Control Engineering in 2015 both from Politecnico di Torino, Turin, Italy.

She works as a research scientist in the University of Luxembourg with the Faculty of Science, Technology and Communication. Formerly, she was with the University of Luxembourg as a post-doc researcher (2017-2018). Between 2010 and 2016, she also worked as a research fellow in the National Research Council of Italy (CNR), Turin, Italy. Her primary research interest concerns embedded systems design and implementation, spanning through topics such as real-time operating systems, communication protocols, formal verification of software modules and communication protocols, as well as security, with a special focus on the practical application of these concepts. Currently, she is focusing on the research of model driven engineering for safety-critical embedded systems. In the meantime, she serves as a program committee member and technical referee for several primary conferences and journals in her research area.

Nicolas Navet has been a professor in Computer Science at the University of Luxembourg since May 2012. Formerly, from 1995 to 2012, he was with INRIA in France, as doctoral candidate, researcher then head of a research team in real-time systems. His research interests include real-time and embedded systems, communication protocols and risk assessment. Since the mid-1990s, he has worked on many projects with OEMs and suppliers in the automotive and aerospace domains. More information on his work can be found at http://nicolas.navet.eu.

Adriano Valenzano (SM'09) received the Laurea degree magna cum laude in electronic engineering from Politecnico di Torino, Torino, Italy, in 1980. He is Director of Research with the National Research Council of Italy (CNR). He is currently with the Institute of Electronics, Computer and Telecommunication Engineering (IEIIT), Torino, Italy, where he is responsible for research concerning distributed computer systems, local area networks, and communication protocols. He has coauthored approximately 200 refereed journal and conference papers in the area of computer engineering.

Dr. Valenzano is the recipient of the 2013 IEEE IES and ABB Lifetime Contribution to Factory Automation Award. He was also awarded for the best paper published in the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS during 2016, and received the Best Paper Awards for the papers presented at the 5th, 8th and 13th IEEE Workshops on Factory Communication Systems (WFCS 2004, WFCS 2010 and WFCS 2017).

Adriano Valenzano has served as a technical referee for several international journals and conferences, also taking part in the program committees of international events of primary importance. Since 2007, he has been serving as an Associate Editor for the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS.