Optimized Event Notification in CAN through In-Frame Replies and Bloom Filters

Gedare Bloom*, Gianluca Cena†, Ivan Cibrario Bertolotti†, Tingting Hu‡, Adriano Valenzano†
*Howard University, 2300 6th St NW, Washington, DC 20059, USA
Email: gedare@scs.howard.edu
†CNR – IEIIT, c.so Duca degli Abruzzi 24, I-10129 Torino, Italy
Email: {gianluca.cena, ivan.cibrario, adriano.valenzano}@ieiit.cnr.it
‡University of Luxembourg – FSTC, 6 Avenue de la Fonte, L-4364 Esch-sur-Alzette, Luxembourg
Email: tingting.hu@uni.lu

***Abstract*—Thanks to its distributed and asynchronous medium access control mechanism, CAN is the ideal choice for interconnecting devices in event-driven systems. When timing requirements of applications are not particularly demanding, as in the case of, e.g., reactive and proactive maintenance, constraints on event delivery can be relaxed, so that their notification may rely on best-effort approaches.**

In this paper, a number of techniques are taken into account for notifying events in such a kind of systems, and their performance has been evaluated. Besides conventional CAN, a recent proposal for extending this protocol, termed CAN XR, is considered. Moreover, the adoption of Bloom filters to cope with rare events in very large systems has also been evaluated.

I. Introduction

Controller Area Network (CAN) [1] was introduced more than 20 years ago for on-board vehicle use, but is now very popular in networked embedded systems as well because of its low cost and high robustness [2], [3]. Basically, the Medium Access Control (MAC) mechanism CAN relies on implements a bit-wise distributed arbitration procedure among nodes, which permits contentions to be resolved at runtime based on message identifiers. This prevents collisions and makes this network suitable for use in event-driven systems. In fact, good reactivity is achieved in spite of the relatively low transmission speed of CAN ($1\,\mathrm{Mb/s}$ at most, which can be increased by resorting to the newer CAN FD protocol [4]).

By coupling bit-wise arbitration and schedulability analysis, *hard* real-time constraints can be satisfied when traffic patterns (period, payload size, and deadline of all data streams) are known in advance [5]–[8]. However, CAN proves to be very suitable for the use in *soft* real-time systems as well, including those which actually have no real-time constraints. In particular, it offers interesting performances in those cases where details on message generation are only partially known at design time, provided that constraints on their delivery are relaxed. For instance, it makes little sense to define strict bounds on delivery times when warning messages have to be collected from a (possibly large) plant, where they can be generated by devices sporadically. Conversely, a *best-effort* strategy often suffices in these cases, which strives to minimize latencies on the average.

978-1-5090-5788-7/17/$31.00

In this paper, CAN-based event-driven systems, characterized by loose timing requirements, are considered. A number of techniques are proposed for dealing with event notification in such a kind of systems, by using both existing solutions (classical CAN and CAN FD) and new technologies, like the CAN with eXtensible in-frame Reply (CAN XR) proposal [9]. The adoption of Bloom filters [10], [11]—a data structure conceived for efficiently managing huge data sets—has also been considered for those scenarios where false positives are tolerated. Simple metrics have been employed to assess, in very general terms, the performance these solutions offer. A thorough analysis of these aspects requires to suitably model event generation, and will be left for future works.

The paper is structured as follows: in Sections II, III, and IV, concise descriptions are provided about event-driven systems, the CAN protocol, and Bloom filters, respectively, while in Section V the conceptual model of the architectures we are taking into account is introduced. Sections VI describes some best-effort approaches for implementing event notification in classical CAN, whereas Sections VII and VIII refer to the CAN FD protocol and the CAN XR proposal, respectively. Finally, some conclusions are drawn in Section IX.

II. Notifications in event-driven systems

Industrial distributed monitoring and control systems based on the event-driven paradigm are made up of nodes that cooperate by exchanging, asynchronously or acyclically, messages over a communication network. Unlike systems where devices are polled cyclically—with cycle times typically in the order of tens of milliseconds, and even less—noticeably lower bandwidth is required, especially in the case a large number of nodes are interconnected whose state changes slowly. A second advantage of event-driven systems is given by their higher robustness: the lack of a centralized coordinator—typically termed the application/network *master*, e.g., a programmable logic controller (PLC)—implies that not necessarily there is a single point of failure (besides, obviously, the bus). On the other hand, no warranties can be provided for the timings of message transmissions, unless their generation rate is suitably constrained (a particular case is represented by periodic traffic). This implies that latencies may grow sensibly

when many events take place at about the same time, and not always reasonably tight upper bounds can be found for them.

In this paper, we restrict our attention to interactions that can be modeled by means of "pure" unqualified events (i.e., no ancillary information is needed to further qualify them). Possible examples are digital systems where each *change of state* is mapped on a specific event. Extension to qualified events (that is, events that bear some additional information with them) is also possible, but it is slightly more complex and will be left for future work. When receiving a notification related to a particular event, a node understands that, somewhere in the network, a specific condition has occurred on another node (the one that raised the notification). For instance, the event may correspond to a warning threshold being exceeded by a specific analog signal (or that signal returning below the threshold again), a proximity sensor detecting the presence of an object (or its absence), the start/completion of an activity (e.g., a software task), a heartbeat pulse, and so on.

In the following, we will focus on systems where a large number (hundreds) of nodes (*sources*) can potentially generate a huge amount of different unqualified events (thousands to millions), which must be notified as soon as possible over a CAN bus. It is worth remembering that the adoption of repeaters to interconnect network trunks permits to increase fan-out, and hence, the number of connected devices. Moreover, when running at low speed (e.g., $50\,\mathrm{kb/s}$ or less), CAN networks can stretch over quite large areas ($1\,\mathrm{km}$ and more).

Therefore, the envisaged solutions are likely suitable to cover the whole plant (or large parts of it). Unlike centralized systems, we assume that there is not a single event engine (*sink*). Conversely, following the well-known *producer-consumer* paradigm, events are broadcast from source nodes on the CAN bus, so that they can reach all the related sink nodes at once. According to this approach, sources are not required to know which sinks will actually consume their events. This scenario closely resembles next-generation sensor networks, which are used to collect information in industrial plants for maintenance purposes (customarily classified as *big data*).

Although generation times of events by sources are not known in advance, as they are spontaneously produced by devices upon specific local conditions, we assume that only a limited number of events are typically raised system-wide at the same time, and in particular that the average (overall) traffic related to event notification, also including a suitable safety margin, is strictly lower than the portion of network bandwidth reserved to this purpose. In fact, for several classes of events (e.g., those related to warning conditions), occurrence is usually considered to be *rare*. In this way, the network almost always operates well below the saturation threshold, so that all events are eventually notified. With little loss of generality, and while not strictly necessary, events are assumed to be sporadic, i.e., a *minimum interarrival time* is defined for them. Since event generation is unpredictable and sources are not synchronized in any way, chances are that, from time to time, the amount of events to be delivered may temporarily exceed the network capacity. In turn, this causes undue delays in event notification. Since timing requirements are assumed to fall under the soft real-time category, late delivery is tolerated as long as it happens seldom.

In order to statistically improve latencies in the above scenario, suitable approaches are needed, which mostly aim at reducing network utilization by increasing communication efficiency. In the following, several techniques for accomplishing this task are described and discussed. A number of them are quite straightforward, and are customarily employed in existing CAN networks. Others are instead based on the CAN XR proposal, possibly combined with fast searching techniques like Bloom filters. Generally speaking, there is not a single, optimal solution, as many factors have to be taken into account.

It has to be pointed out that mixing different kinds of messages in CAN is indeed possible. As a consequence, besides asynchronous notifications, other pieces of information can be concurrently conveyed on the same network, like analog readings (temperature, level, flow, position, etc.) or structured data (parameter blocks, text strings, and others), possibly carried out at a periodical pace and even characterized by firm deadlines. For instance, a set of real-time streams (corresponding to, e.g., process data objects) can be mapped on higher-priority identifiers, and feasibility analysis can be exploited to assess whether or not timing constraints are met. Part of the lower-priority identifiers can instead be reserved to event notification according to the techniques described below. Hence, full composability can be achieved, merely considering the blocking effect lower-priority messages have on higher-priority ones during feasibility analysis. Importantly, encoding schemes that may cause false positives can coexist side-by-side with those which do not suffer from this limitation (by mapping them on either distinct messages or different parts of the same message). Every application will select the most suitable notification approach, depending on its specific requirements.

We assume that devices (CAN nodes) can behave as either sources or sinks of events (*sensors* and *actuators*, respectively), and they can possibly assume both roles at the same time too (*controllers*). Dealing with events once they have been delivered to the sink(s) is outside the scope of this work.

III. CAN BASICS

The MAC mechanism of CAN is based on bit-wise arbitration, which is (mostly) carried out on the identifier field. Two different sizes are foreseen for this field, namely *base* (11 bits) and *extended* (29 bits). Concerning the size of the data field, it is possible to distinguish between *classical CAN* (8 bytes at most) and *CAN FD* (up to 64 bytes). By combining these options, *four* frame formats are actually defined for data frames in [1], that is: Classical Base Frame Format (CBFF), Classical Extended Frame Format (CEFF), FD Base Frame Format (FBFF), and FD Extended Frame Format (FEFF).

TABLE I
FRAME SIZE S VS. PAYLOAD SIZE D IN CAN (ALL VALUES IN BITS)

Section	CBFF	CEFF	FBFF	FEFF
Arbitration	13	33	13	33
Control	6	6	9	8
Data	$8D$	$8D$	$8D$	$8D$
Data (max)	64	64	128 [512]	128 [512]
CRC	15	15	27 [32]	27 [32]
Trailer	10	10	10	10
IMS	3	3	3	3
Total	$47 + 8D$	$67 + 8D$	62[67]+$8D$	81[86]+$8D$
Total (min)	47	67	62	81
Total (max)	111	131	190 [579]	209 [598]

A. Message size

CAN DATA messages are made up of several sections:

- *Arbitration*: Start of Frame (SOF) bit, Identifier (ID) encoded on either 11 or 29 bits—in the latter case also including the Substitute Remote Request (SRR) and Identifier Extension (IDE) bits—as well as the Remote Transmission Request (RTR/RRS) bit.
- *Control*: its exact format depends on the specific CAN flavor; besides the Data Length Code (DLC) on 4 bits, the FD Format indicator (FDF), and possibly one reserved bit, it includes, in FD formats, the Bit Rate Switch (BRS) and Error State Indicator (ESI) bits; importantly, for base formats the IDE bit is also located here.
- *Data*: Includes a variable number D of bytes ($D = 0...8$ for classical CAN and $D = 0...64$ for CAN FD); in the former case DLC directly encodes D, whereas in the latter DLC patterns above 8 are put into correspondence with $D = 12, 16, 20, 24, 32, 48,$ and 64, respectively.
- *CRC*: Stuff Count (3 bits plus parity) and CRC sequence (15 bits in Classical CAN, either 17 or 21 bits in CAN FD, depending on whether $D \leq 16$ or $20 \leq D \leq 64$); fixed stuff bits must be added in CAN FD at the beginning and after each fourth bit of the CRC field, this resulting in 6 or 7 additional bits, depending on D.
- *Trailer*: CRC delimiter (CDEL), ACK slot (ASLOT), and ACK delimiter (ADEL), followed by the End of Frame (EOF) field encoded as 7 recessive bits.
- *Intermission* (IMS): 3 recessive bits located between any pair of adjacent frames.

The lower part of Table I reports the frame size S (in bits) vs. the payload size D (in bytes) for CBFF, CEFF, FBFF, and FEFF, as well as the minimum and maximum size (for empty and maximal payloads, respectively). For FBFF and FEFF, sizes refer to both the case $D \leq 16$ and (in square brackets) the case $20 \leq D \leq 64$. Non-fixed stuff bits have been neglected for simplicity, as their exact number may vary depending on the payload. However, they affect in a similar way all formats. As we are dealing with channel occupation, intermission is included in the computation of S.

B. CAN XR

CAN with eXtensible in-frame Reply (XR) [9] is a recent proposal for extending CAN functionality. Although it applies to both classical and FD frames, the latter choice brings higher benefits because of the larger payload. While the frame format remains exactly the same as CAN (or CAN FD), in order not to impair coexistence with existing controllers, the protocol is augmented is such a way that a plurality of nodes can take part into the transmission of the same frame. Basically, every exchange in CAN XR is started by a specific node (*initiator*), and corresponds to an atomic *XR transaction*. Multiple nodes (*responders*) are allowed to be writing on the bus in the data field of XR frames (in-frame replies). Target nodes (*consumers*), which may rely on conventional CAN controllers, obtain all such pieces of information at once.

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 (*exclusive* slots) or overlapping (*shared* slots). In the latter case, which is quite relevant for this work, the resulting bit pattern on the bus corresponds to the bit-wise AND among the bit patterns sent by all responders.

Besides starting a transaction, the initiator also takes care of supervising it, by inserting stuff bits when needed so as to preserve the encoding rules of CAN (*supervisor* role). It is worth remarking that, at any time, both the active responder and the initiator carry out this task, and that the related stuff bits will overlap since they are inserted according to the same rules. The initiator also takes care of terminating the frame, by transmitting the CRC and the trailer. Additional details on a preliminary version of CAN XR can be found in [9]. A prototype implementation of CAN XR on a software-defined CAN controller (SDCC) proved that the protocol behaves correctly and retains full compatibility with existing controllers.

IV. BLOOM FILTERS BASICS

Bloom filters are typically used with data structures optimized for insert and search operations. They are a valid alternative to bitmaps, when the cardinality of the set among which elements are drawn (*universe*) is very large but the number of elements actually stored in the data structure is much smaller. Both bitmaps and Bloom filters can be used for creating and managing dynamic sets, and can be seen as functions that, given an element (unambiguously characterized by its *key*), check whether or not it belongs to the set. Although techniques are available that permit removal of elements from Bloom filters, we will consider the case where elements can only be added (by far simpler and more space-efficient).

The only drawback of Bloom filters is that, sometimes they may return false positives, i.e., they may indicate that a given element is present in the set while it is actually not there. By carefully choosing the size of the data structure, the probability of this event happening can be made arbitrarily low. However, the lower this probability is, the higher the overhead, which means that there is a trade-off between these quantities.

A. Bloom filters basics

A Bloom filter basically consists of a bit array b (bitmap) which includes m Boolean values (bits, for short), used to "store" elements drawn from a universe A. It is modeled as a function $\beta(a) : A \mapsto \{1, 0\}$ that checks the presence of element $a \in A$ in the data structure (a coincides with the key of the element). Additionally, k hash functions are defined, we denote $h_i(a) : A \mapsto \{0, ..., m-1\}, i = 1...k$.

Initially, all bits of an empty Bloom filter are set to 0. Every time an element a is inserted in the data structure, all the hash functions are evaluated and the bits of b in the corresponding positions are set to 1. In formulas, $b[h_i(a)] \leftarrow 1, i = 1...k$, where operator "$\leftarrow$" denotes assignment.

Checking the presence of element a in the data structure is also very simple and efficient. Hash functions are evaluated in a and the bits of b in the corresponding positions are checked. If all of them are at the value 1 the element is *possibly present*, while on the contrary it is *certainly absent*. In formulas, $\beta(a) = \prod_{i=1}^{k} b[h_i(a)], i = 1...k$, where operator "$\prod$" represents the logical product of a sequence.

This approach may result in false positives, i.e., a Bloom filter may occasionally report that an absent element is present. The probability of false positives can be calculated [12], [13], and mostly depends on the ratio between the size m of b and the number n of inserted elements, as well as on k. For instance, if $m/n = 8$, then such a probability can be as low as 2.16% when k is set to 6. Importantly, unlike conventional bitmaps, the cardinality of A is, in theory, irrelevant.

B. Using Bloom filters for event notification

Bloom filters can be applied to event notification in distributed systems where a certain amount of false positives (i.e., when sinks mistakenly assume that a certain event has been notified, while it was not) is tolerated. It is worth remarking that, unlike the case where Bloom filters are used with data structures, having sinks checking separately every event notification to discover false positives nullifies most of the advantages of using Bloom filters. Hence, we must look at use cases where these additional checks do not take place.

A first example is given by system warnings, whose presence is used to update diagnostic data in distributed control systems (e.g., to achieve reactive and proactive maintenance). In these cases, changing marginally statistics (due to a relatively small number of false positives being counted as actual warnings) is usually perfectly acceptable.

A second example is given by modern safety systems, where moving near a piece of equipment (e.g., a robot) causes it to move more slowly, and only when the distance falls below the safety limit to stop completely. This is sometimes referred to as Dual Check Safety (FANUC DCS). Again, if from time to time the equipment is unnecessarily slowed down, the impact on system performance is negligible.

A third example is found in control systems where a certain event, notified by a source, is the direct cause of another event, which is consequently generated by the sink. In the case the source senses the second event but it did not raise the originating event, it can try to undertake corrective actions (only applicable to non-critical systems with slow dynamics).

V. System Model

Let $\mathsf{S} = \{\mathsf{s}_1, ..., \mathsf{s}_{|S|}\}$ be the set of *nodes* in the network, while $\mathsf{E_S} = \{\mathsf{e}_1, ..., \mathsf{e}_{|\mathsf{E}_S|}\}$ is the set of all *events* defined in the system (universe). Operator $|\cdot|$ denotes the cardinality of the set it is applied to.

A. Encoding of Events

In order to send notifications over CAN, events are mapped on specific messages. Let $\mathsf{M_S} = \{\mathsf{m}_1, ..., \mathsf{m}_{|\mathsf{M_S}|}\}$ be the set of messages reserved to event notification. Quite obviously $|\mathsf{M_S}| \leq N_{\mathrm{ID}}$, where $N_{\mathrm{ID}}^{std} = 2^{11}$ for base (standard) CAN identifiers and $N_{\mathrm{ID}}^{ext} = 2^{29}$ for extended ones. In the following, for simplicity, we will implicitly assume that the whole identifier space is available for events.

One or more messages $\mathsf{m}_{i,j}$ can be assigned to node s_i for notifying events. Let $\mathsf{M}_{\mathsf{s}_i} = \{\mathsf{m}_{i,1}, ..., \mathsf{m}_{i,|\mathsf{M}_{\mathsf{s}_i}|}\}$ be the set of such messages. These sets are exhaustive, i.e., $\mathsf{M_S} = \bigcup_{i=1...|\mathsf{S}|} \mathsf{M}_{\mathsf{s}_i}$, and exclusive, i.e., $\mathsf{M}_{\mathsf{s}_x} \cap \mathsf{M}_{\mathsf{s}_y} = \emptyset, x \neq y$. The latter property can be relaxed if empty CAN messages are considered, for which more than one producer can be active at the same time in the network. This means, that the same empty message can be assigned to more than one node.

Roughly speaking, two classes of solutions can be devised for event mapping: either a *flat* event space is envisaged or a *hierarchical* scheme is adopted. In the former case, each event is mapped onto a distinct empty message, which ensures the highest flexibility in the system configuration phase (every node can in theory notify all events). In the latter case, events are exclusively assigned to (and managed by) the related source node (up to $|\mathsf{S}|$ distinct devices). So as to optimize notifications, multiple distinct events can be mapped onto the same non-empty message with hierarchical schemes, by suitably encoding them in the data field. This is not possible when mapping is flat, because non-empty messages in conventional CAN must have a unique producer.

Let $\mathsf{E}_{\mathsf{m}_{i,j}}$ be the set of events that are mapped by s_i on a given message $\mathsf{m}_{i,j}$. If $\mathsf{E}_{\mathsf{m}_{i,j}} = \{\mathsf{e}_g\}$ the notification of e_g coincides with the transmission of $\mathsf{m}_{i,j}$. Otherwise, if $\left|\mathsf{E}_{\mathsf{m}_{i,j}}\right| > 1$ at least one event has to be active in $\mathsf{E}_{\mathsf{m}_{i,j}}$ so as to trigger the message exchange. The set of all events generated by node s_i is denoted $\mathsf{E}_{\mathsf{s}_i}$, and can be expressed as $\mathsf{E}_{\mathsf{s}_i} = \bigcup_{j=1...|\mathsf{M}_{\mathsf{s}_i}|} \mathsf{E}_{\mathsf{m}_{i,j}}$. Generally speaking, $|\mathsf{E}_{\mathsf{s}_i}|$ depends on the number of messages assigned to s_i, their size, and the encoding scheme used for events. Overall, $\mathsf{E_S} = \bigcup_{i=1...|\mathsf{S}|} \mathsf{E}_{\mathsf{s}_i}$.

When $\left|\mathsf{E}_{\mathsf{m}_{i,j}}\right| > 1$, not necessarily all pending events in $\mathsf{E}_{\mathsf{m}_{i,j}}$ are able to fit into a single instance of $\mathsf{m}_{i,j}$. This property, in fact, only holds for some of the encoding schemes. Let $\mathsf{m}_{i,j,\ell}$ denote instance ℓ of $\mathsf{m}_{i,j}$, and $\mathsf{V}_{\mathsf{m}_{i,j,\ell}}$ the set of distinct events which are conveyed in that instance. For simplicity, in the following we will assume that encoding is such that the maximum number $\mathcal{V}_{\mathsf{m}_{i,j}}$ of events that can be included in a single instance of $\mathsf{m}_{i,j}$ (*message capacity*) is fixed and does not depend on the specific events included.

This means, $\left|\mathsf{V}_{\mathsf{m}_{i,j,\ell}}\right| \leq \mathcal{V}_{\mathsf{m}_{i,j}}, \forall \ell$. Whenever message capacity is exceeded several instances of $\mathsf{m}_{i,j}$ will be sent on the bus.

B. Performance Metrics

The following metrics are considered in order to characterize solutions for event notification:

- *Maximum number of events* ($\mathcal{E}_\mathsf{S}$): how many distinct events can be defined in the system whose notification is supported by the considered solution, $|\mathsf{E}_\mathsf{S}| \leq \mathcal{E}_\mathsf{S}$.
- *Maximum event notification rate* (Λ): how many events can be conveyed in a specific time interval under sustained traffic conditions (all events always active).
- *Effective event notification rate* (λ): how many events can be conveyed in a specific time interval under specific event generation conditions.

Let R be the *bit rate* on the CAN bus, $R = 1/T_{\text{bit}}$, and let C_m be the *duration* of message m on the bus, which depends on its *size* S (in bits) and R. For message m it can be expressed as $C_\mathsf{m} = S_\mathsf{m}/R$, where S_m depends on the data field size D_m (in bytes) and the frame format (either *base* or *extended* identifier, for both *classical* and *FD* frames).

The maximum event notification rate Λ_m for message m (that is, assuming that it can exploit the whole network bandwidth and all the related instances are always filled with events up to their capacity), can be evaluated as

$$\Lambda_\mathsf{m} = \mathcal{V}_\mathsf{m}/C_\mathsf{m} = R \cdot \mathcal{V}_\mathsf{m}/S_\mathsf{m}. \tag{1}$$

Since only one node can be transmitting in CAN at any given time, if all messages in M_S are assumed to have the same size and encoding, then $\Lambda_\mathsf{S} = \Lambda_\mathsf{m}$.

Unlikely, in a well-dimensioned network, many events may be pending on the same node at the same time waiting for transmission. Nevertheless, the ability to pack several pending events into the same message can help to overcome temporary overload conditions quickly. The effective event notification rate λ_m can be evaluated as

$$\lambda_\mathsf{m} = \bar{\nu}_\mathsf{m}/C_\mathsf{m} = R \cdot \bar{\nu}_\mathsf{m}/S_\mathsf{m}. \tag{2}$$

where $\bar{\nu}_\mathsf{m}$ denotes the average number of events included in each single instance of m ($\bar{\nu}_\mathsf{m} \leq \mathcal{V}_\mathsf{m}$). Let $\bar{\epsilon}_\mathsf{m}$ be the mean number of pending events, among those mapped on message m, a node has to notify when gaining bus access ($\bar{\epsilon}_\mathsf{m} \leq |\mathsf{E}_\mathsf{m}| \leq \mathcal{E}_\mathsf{m}$), where $\mathcal{E}_\mathsf{m}$ is the maximum number of events that can be mapped on m. If $\mathcal{V}_\mathsf{m} \geq |\mathsf{E}_\mathsf{m}|$ then all pending events can always fit in m, and hence $\bar{\nu}_\mathsf{m} = \bar{\epsilon}_\mathsf{m}$. Conversely, if $\mathcal{V}_\mathsf{m} < |\mathsf{E}_\mathsf{m}|$ several instances of m may be required to convey the pending events ($\sim \lceil \bar{\epsilon}_\mathsf{m}/\mathcal{V}_\mathsf{m} \rceil$, on average). In steady-state conditions, the mean number of events carried in an instance of m can be approximated as $\bar{\nu}_\mathsf{m} \simeq \bar{\epsilon}_\mathsf{m}/\lceil \bar{\epsilon}_\mathsf{m}/\mathcal{V}_\mathsf{m} \rceil$.

VI. Mapping Events on CAN Messages

Thanks to its access scheme based on arbitration, CAN is a very good choice for connecting devices in event-driven systems. Unlike most industrial communication systems based on the master-slave (centralized) approach, there is no need for a node that continuously polls networked devices to determine if some event has possibly arisen. When a condition occurs on a device which need to be quickly notified to the system, a message is spontaneously broadcast over the CAN bus according to the producer-consumer (distributed) approach. This means that more than one sink can be notified at the same time using a single message transmission.

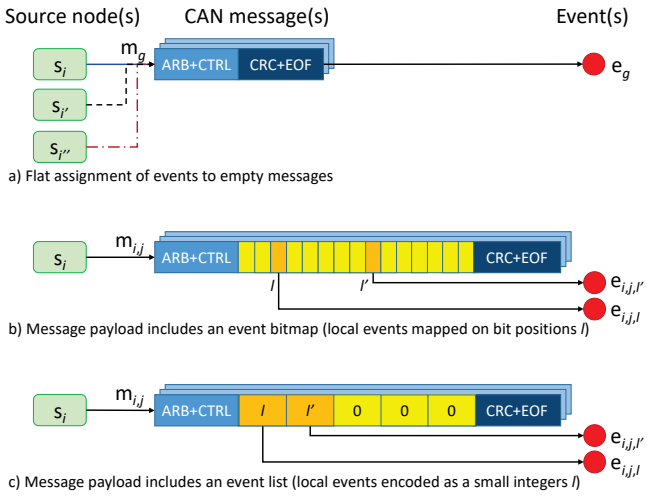

Fig. 1. Mapping of events in conventional CAN (flat, bitmap, list).

In the following, a number of basic approaches are described for dealing with event notification in classical CAN. Besides using a separate message per event, solutions like bitmaps and lists will be taken into account, which permit several events to be gathered in the same message.

A. Flat event mapping

As depicted in Fig. 1-a, each event $\mathsf{e}_g \in \mathsf{E}_S$ is directly mapped on a distinct CAN message m_g, where g is a *global* index which identifies that specific event system-wide. In other words, $\mathsf{V}_{\mathsf{m}_g} = \mathsf{E}_{\mathsf{m}_g} = \{\mathsf{e}_g\}$. If, as assumed, no ancillary information has to be provided along with the event, empty messages can be used. By exploiting the properties of empty messages in CAN, the same event (characterized by a specific CAN identifier) can be notified by more than one node, without causing any issues to the arbitration mechanism. The related bit sequences, in fact, are exactly the same and will overlap on the bus. This is the simplest and most straightforward solution, and in the following it will be used as the *baseline*.

Mapping between events and CAN identifiers can be modeled as an *injective* function, but it is typically not *bijective* as in real networks some identifiers are likely to be reserved for other kinds of data exchanges. Thus, $\mathcal{E}_\mathsf{S} = |\mathsf{M}_\mathsf{S}|$, which implies $\mathcal{E}_\mathsf{S}^{std} = 2048$ and $\mathcal{E}_\mathsf{S}^{ext} \simeq 537 \cdot 10^6$. If we consider a network made up of a single node, which generates only one kind of event, mapped on message m, $\Lambda_\mathsf{m}^{std} = 21.28 \cdot 10^{-3} \cdot R$ while $\Lambda_\mathsf{m}^{ext} = 14.93 \cdot 10^{-3} \cdot R$. As all empty messages have the same size, for the whole network we have $\Lambda_\mathsf{S} = \Lambda_\mathsf{m}$. For example, on conventional CAN running at $50\,\text{kb/s}$, the notification rate can be as high as about $1\,\text{kHz}$.

The case where more than one event can be generated by the same node can be trivially dealt with by assigning the node more than one message (one per event). The overall maximum number of events $\mathcal{E}_\mathsf{S}$ does not change, because the amount of CAN identifiers available network-wide remains the same. The same holds for the maximum notification rate Λ_S.

B. Hierarchical event mapping

When devices are allowed to generate multiple events, possibly at the same time, *hierarchical* mapping could be a better solution than flat mapping. Each event e_g in the whole system is mapped on a triple, $g \mapsto \langle i, j, l \rangle$, where s_i is the producing node, $\mathsf{m}_{i,j}$ is the message used by s_i for notifying e_g, and l is a *local* index that identifies a specific event among those encoded in $\mathsf{m}_{i,j}$. Unless a single, fixed event is mapped on the message (which bring us back to the case of flat mapping), the data field can not be empty, which implies that the producing device of each event must be unique on the whole network.

So as to make comparison of the performance indices for the different solutions easier, in the following we will assume that, for any given solution, all messages used for event notification by every node in the network have the same encoding and size. This implies that the maximum cardinality $\mathcal{E}_\mathsf{m}$ of the set of events that can be mapped on any message m, as well as its actual capacity $\mathcal{V}_\mathsf{m}$, are fixed and do not depend on m. Under the above hypotheses, $\mathcal{E}_\mathsf{S} = |\mathsf{M_S}| \cdot \mathcal{E}_\mathsf{m}$. Moreover, the expressions of the maximum network event notification rate Λ_S are greatly simplified, as this quantity becomes equal to the maximum notification rate Λ_m in the case only message m is repeatedly being exchanged over the network.

The case of the effective notification rate is noticeably more complex to deal with. In this paper, for sake of simplicity we will assume that the generation law for every event in $\mathsf{E_S}$, although random, is the same. Therefore, the average number $\bar{\nu}_\mathsf{m}$ of events included in each message instance does not depend on m. This means that the rate at which events are notified in the whole system can be reasonably approximated by the notification rate evaluated for a single message (that is, λ_S is about the same as λ_m). An in-depth analysis, based on statistical characterization of event generation on nodes and their distribution on messages, requires a suitable network simulator, and will be left for future works.

Concerning notification rates, it should be noted that the frame size S_m in (1) and (2) depends on D_m, which in turn is decided depending on $\mathcal{E}_\mathsf{m}$, $\mathcal{V}_\mathsf{m}$, and the scheme adopted to encode local events. Clearly, optimized solutions can be also devised, possibly based on mixed notification schemes, where the above assumptions no longer hold.

Two sample encoding schemes will be described below. While meaningful for real applications, by no means they have to be considered exhaustive.

1) Event Bitmap: A very efficient approach to encode the events raised by a node is to use the data field of its messages as a bitmap, as shown in Fig. 1-b. In particular, the simplest scheme is to rely, for each message m, on a static assignments of the events in $\mathsf{E_m}$ to the bits in the data field on m. In this case, l coincides with the bit position in the bitmap. Moreover, $\mathcal{E}_\mathsf{m}$ coincides with $\mathcal{V}_\mathsf{m}$. In particular, if the data field of m includes D_m bytes, $\mathcal{E}_\mathsf{m} = \mathcal{V}_\mathsf{m} = 8D_\mathsf{m}$.

As per our simplifying assumptions, all messages have the same size and encoding, and so for the whole network we have $\mathcal{E}_\mathsf{S} = |\mathsf{M_S}| \cdot 8D_\mathsf{m}$ and $\Lambda_\mathsf{S} = \Lambda_\mathsf{m} = R \cdot 8D_\mathsf{m}/S_\mathsf{m}$, whatever the assignment of messages to nodes. If the maximal frame size allowed in classical CAN is taken into account ($D_\mathsf{m} = 8$), then $\mathcal{E}_\mathsf{S}^{std} \simeq 131 \cdot 10^3$ and $\Lambda_\mathsf{S}^{std} = 576.58 \cdot 10^{-3} \cdot R$, much higher than with flat mapping. However, unlikely all local events of a given device will be raised at the same time. Since for bitmaps $\mathcal{E}_\mathsf{m} = \mathcal{V}_\mathsf{m}$, all pending events will always fit in a single instance of m (that is, $\bar{\nu}_\mathsf{m} = \bar{\epsilon}_\mathsf{m}$). Thus, the effective event notification rate is $\lambda_\mathsf{S}^{std} \simeq \lambda_\mathsf{m}^{std} = \bar{\epsilon}_\mathsf{m} \cdot 9.01 \cdot 10^{-3} \cdot R$.

Obviously, the unused message capacity of $\mathsf{m}_{i,j}$ cannot be reused by nodes other than s_i, and not even by node s_i itself for its events mapped on messages other than $\mathsf{m}_{i,j}$. However, this is not a severe issue, given the non-negligible protocol overhead in CAN frames. Comparing λ_S to the baseline solution (flat mapping), it turns out that, in the case $D_\mathsf{m} = 8$, bitmaps are advantageous when at least $\bar{\nu}_\mathsf{m} = 3$ events are conveyed, on average, in every CAN message, whereas 2 events suffice in the case $D_\mathsf{m} \leq 5$.

It is worth noting that, when $|\mathsf{E}_{\mathsf{s}_i}| > 64$, more than one message has to be allocated to s_i, that is, $|\mathsf{M}_{\mathsf{s}_i}| > 1$. Because of our simplifying assumptions, this does not change neither $|\mathsf{E_S}|$ and not even Λ_S. In this case, unless events generated by the same node are statistically correlated, the optimal solution is to minimize the number of messages allocated to each node, by enlarging their size D_m as much as possible.

2) Event List: In order to provide higher flexibility, the message data field can be used to convey a variable number of local events encoded as a list, as sketched in Fig. 1-c (many different implementations can be devised to this aim). Unlike bitmaps, $\mathcal{E}_\mathsf{m}$ and $\mathcal{V}_\mathsf{m}$ are typically not the same. Generally speaking, encoding local events using patterns of w bits ($w \leq 8D_\mathsf{m}$) permits the assignment of up to $\mathcal{E}_\mathsf{m} = 2^w$ distinct events to message m (one less, in the case a specific pattern is reserved to encode the "*no event*" condition). In other words, $1 \leq l \leq 2^w - 1$. In the simplest case when, as per our simplifying assumptions, w is the same for every message in the network, $\mathcal{E}_\mathsf{S} = |\mathsf{M_S}| \cdot (2^w - 1)$. Instead, the capacity of message m is up to $\mathcal{V}_\mathsf{m} = \lfloor 8D_\mathsf{m}/w \rfloor$.

For instance, if $D_\mathsf{m} = 8$ bytes (largest classical CAN frame) and $w = 8$ bits (local events are encoded on one byte, which means $\mathcal{E}_\mathsf{S} = |\mathsf{M_S}| \cdot 255$), then up to $\mathcal{V}_\mathsf{m} = 8$ events can be conveyed at a time in the same message, which implies $\Lambda_\mathsf{S}^{std} = \Lambda_\mathsf{m}^{std} = 72.07 \cdot 10^{-3} \cdot R$. Message capacity $\mathcal{V}_\mathsf{m}$ is lower than for bitmaps, even though set $\mathsf{E_m}$ can be noticeably larger. Typically, $\mathcal{V}_\mathsf{m} \leq |\mathsf{E_m}|$, in which case $\lambda_\mathsf{S}^{std} \simeq \lambda_\mathsf{m}^{std} \simeq \bar{\epsilon}_\mathsf{m} / \lceil \bar{\epsilon}_\mathsf{m} / \lfloor 8D_\mathsf{m}/w \rfloor \rceil \cdot 9.01 \cdot 10^{-3} \cdot R$. The case $\mathcal{V}_\mathsf{m} > |\mathsf{E_m}|$ is hardly interesting, as this means that bandwidth is being wasted (more room is allocated for events in messages than needed to notify them).

TABLE II
EVENT NOTIFICATION SCHEMES BASED ON CONVENTIONAL CAN (FLAT, BITMAPS, AND LISTS)

Scheme	D (B)	ID	$\mathcal{E}_\mathsf{S}$	$\mathcal{E}_\mathsf{m}$	$\mathcal{V}_\mathsf{m}$	w	Λ_S (kHz)	λ_m (kHz)	Notes
CAN flat (baseline)	0	std	$\sim 2 \cdot 2^{10}$ $(2.05 \cdot 10^3)$	1	1	—	$21.28 \cdot R$	$21.28 \cdot R$	one message per event
CAN flat (baseline)	0	ext	$\sim 512 \cdot 2^{20}$ $(537 \cdot 10^6)$	1	1	—	$14.93 \cdot R$	$14.93 \cdot R$	one message per event
CAN bitmap	1	std	$\sim 16 \cdot 2^{10}$ $(16.4 \cdot 10^3)$	8	8	—	$145.45 \cdot R$	$\bar{\nu}_\mathsf{m} \cdot 18.18 \cdot R$	$\bar{\nu}_\mathsf{m}$ refers to a single message
CAN bitmap	1	ext	$\sim 4 \cdot 2^{30}$ $(4.29 \cdot 10^9)$	8	8	—	$106.67 \cdot R$	$\bar{\nu}_\mathsf{m} \cdot 13.33 \cdot R$	$\bar{\nu}_\mathsf{m}$ refers to a single message
CAN bitmap	8	std	$\sim 128 \cdot 2^{10}$ $(131 \cdot 10^3)$	64	64	—	$576.58 \cdot R$	$\bar{\nu}_\mathsf{m} \cdot 9.01 \cdot R$	$\bar{\nu}_\mathsf{m}$ refers to a single message
CAN bitmap	8	ext	$\sim 32 \cdot 2^{30}$ $(34.4 \cdot 10^9)$	64	64	—	$488.55 \cdot R$	$\bar{\nu}_\mathsf{m} \cdot 7.63 \cdot R$	$\bar{\nu}_\mathsf{m}$ refers to a single message
CAN FD bitmap	64	std	$\sim 1 \cdot 2^{20}$ $(1.05 \cdot 10^6)$	512	512	—	$884.28 \cdot R$	$\bar{\nu}_\mathsf{m} \cdot 1.73 \cdot R$	$\bar{\nu}_\mathsf{m}$ refers to a single message
CAN FD bitmap	64	ext	$\sim 256 \cdot 2^{30}$ $(275 \cdot 10^9)$	512	512	—	$856.19 \cdot R$	$\bar{\nu}_\mathsf{m} \cdot 1.67 \cdot R$	$\bar{\nu}_\mathsf{m}$ refers to a single message
CAN list	8	std	$\sim 30 \cdot 2^{10}$ $(30.7 \cdot 10^3)$	15	16	4	$144.14 \cdot R$	$\bar{\nu}_\mathsf{m} \cdot 9.01 \cdot R$	$\bar{\nu}_\mathsf{m}$ refers to a single message
CAN list	8	ext	$\sim 7.5 \cdot 2^{30}$ $(8.05 \cdot 10^9)$	15	16	4	$122.14 \cdot R$	$\bar{\nu}_\mathsf{m} \cdot 7.63 \cdot R$	$\bar{\nu}_\mathsf{m}$ refers to a single message
CAN list	8	std	$\sim 510 \cdot 2^{10}$ $(522 \cdot 10^3)$	255	8	8	$72.07 \cdot R$	$\bar{\nu}_\mathsf{m} \cdot 9.01 \cdot R$	$\bar{\nu}_\mathsf{m}$ refers to a single message
CAN list	8	ext	$\sim 128 \cdot 2^{30}$ $(137 \cdot 10^9)$	255	8	8	$61.07 \cdot R$	$\bar{\nu}_\mathsf{m} \cdot 7.63 \cdot R$	$\bar{\nu}_\mathsf{m}$ refers to a single message
CAN list	8	std	$\sim 128 \cdot 2^{20}$ $(134 \cdot 10^6)$	65535	4	16	$36.04 \cdot R$	$\bar{\nu}_\mathsf{m} \cdot 9.01 \cdot R$	$\bar{\nu}_\mathsf{m}$ refers to a single message
CAN list	8	ext	$\sim 32 \cdot 2^{40}$ $(35.2 \cdot 10^{12})$	65535	4	16	$30.53 \cdot R$	$\bar{\nu}_\mathsf{m} \cdot 7.63 \cdot R$	$\bar{\nu}_\mathsf{m}$ refers to a single message
CAN FD list	64	std	$\sim 510 \cdot 2^{10}$ $(522 \cdot 10^3)$	255	64	8	$110.54 \cdot R$	$\bar{\nu}_\mathsf{m} \cdot 1.73 \cdot R$	$\bar{\nu}_\mathsf{m}$ refers to a single message
CAN FD list	64	ext	$\sim 128 \cdot 2^{30}$ $(137 \cdot 10^9)$	255	64	8	$107.02 \cdot R$	$\bar{\nu}_\mathsf{m} \cdot 1.67 \cdot R$	$\bar{\nu}_\mathsf{m}$ refers to a single message
CAN FD list	64	std	$\sim 128 \cdot 2^{20}$ $(134 \cdot 10^6)$	65535	32	16	$55.27 \cdot R$	$\bar{\nu}_\mathsf{m} \cdot 1.73 \cdot R$	$\bar{\nu}_\mathsf{m}$ refers to a single message
CAN FD list	64	ext	$\sim 32 \cdot 2^{40}$ $(35.2 \cdot 10^{12})$	65535	32	16	$53.51 \cdot R$	$\bar{\nu}_\mathsf{m} \cdot 1.67 \cdot R$	$\bar{\nu}_\mathsf{m}$ refers to a single message

In theory, more than one message can be allocated to each node in order to enlarge $\mathcal{E}_{\mathsf{s}_i}$, but this is usually pointless because, with event lists, the same goal can typically be achieved more effectively by a proper selection of w. For example, by setting $w = 16$ then $\mathcal{E}_\mathsf{S} = |\mathsf{M}_\mathsf{S}| \cdot 65535$ (while $\mathcal{V}_\mathsf{m} = 4$ events per message). It is worth pointing out that many different ways exist to encode the list of pending events. For instance, efficiency can be further increased by resorting to Huffman codes, so that more frequent events take less bits and can be packed more densely. A thorough analysis of these aspects is beyond the scope of this paper.

3) Bloom filters: Applying Bloom filters to hierarchical schemes over conventional CAN has little practical relevance. As said above, their use can be advantageous with respect to bitmaps when the number of distinct events that can be generated by a node is huge (in theory, sets E_m with unlimited size are supported) but their occurrence is rare, and false positives may be occasionally tolerated. For instance, when events are encoded using this approach in a classical CAN frame with maximal size using $k = 5$ hash functions (increasing this value excessively is likely a bit tricky), the probability P_{fp} of false positives does not exceed 0.00633% when up to $\mathcal{V}_\mathsf{m} = 2$ events are conveyed in m, but increases to 0.139% and 2.17% when the events are 4 and 8, respectively [13].

However, using an event list and allocating the whole data field to encode at most $\mathcal{V}_\mathsf{m} = 2$ events (i.e., setting $w = 32$), yields (about) the same notification rate and permits to deal potentially with up to 2^{32} distinct events per message (which is far beyond most reasonable applications' needs), but does not lead to any false positives. For this reason, we will not analyze Bloom filters over conventional CAN in detail.

VII. MAPPING EVENTS ON CAN FD MESSAGES

Basically, the same considerations made above for classical CAN also hold for CAN FD. The main difference is that, the data field can consist of up to 64 bytes (i.e., 512 bits) instead of 8, which means that the number of events $\mathcal{V}_\mathsf{m}$ that can fit in every message increases sensibly, in spite of the slightly worse overhead due to the larger frame header and trailer.

Exploiting bit-rate switching [14], [15], which consists in increasing the network bit rate during data transmission (with the exception of the initial and final parts of the frame, where arbitration and acknowledgment are carried out, respectively) usually leads to noticeably shorter transmission times, even when the larger payload size achieved by CAN FD is used.

A. Hierarchical event mapping

Using flat mapping is hardly advantageous when CAN FD is taken into account, since the larger payload (up to 64 bytes) is left unused. For this reason, only the case of hierarchical mapping is considered in the following.

1) Event Bitmap: When the data field contains a bitmap, the maximum number of distinct events that can be encoded in a single message grows by a factor 8 with respect to classical CAN. In particular, up to $\mathcal{E}_\mathsf{m} = \mathcal{V}_\mathsf{m} = 512$ events are made available for any message m (and can be included in it), which implies that one message per node is often sufficient. System-wide we have $\mathcal{E}_\mathsf{S} = |\mathsf{M}_\mathsf{S}| \cdot 512$ events.

When $D_\mathsf{m} = 512$ the maximum notification rate $\Lambda_\mathsf{S}^{std} = \Lambda_\mathsf{m}^{std} = 884.28 \cdot 10^{-3} \cdot R$ is $\sim 53\%$ higher than using bitmaps with classical CAN, while the effective notification rate is $\lambda_\mathsf{S}^{std} \simeq \lambda_\mathsf{m}^{std} = \bar{\nu}_\mathsf{m} \cdot 1.73 \cdot 10^{-3} \cdot R$. Since unlikely all the local events will become active at the same time, the higher payload of CAN FD is going to be wasted most of the times, which means that throughput actually decreases. As a consequence, in order to globally increase $\mathcal{E}_\mathsf{S}$, switching to the extended CAN identifier format is probably a better solution than moving from CAN to CAN FD.

2) Event List: The larger payload offered by CAN FD can be useful also with event lists. For instance, for any

given size w of the patterns on which events are encoded, message capacity increases to $\mathcal{V}_\mathsf{m} = \lfloor 64D_\mathsf{m}/w \rfloor$ events. It is worth pointing out that the maximum notification rate Λ_S only improves marginally with respect to classical CAN (for example, $110.54 \cdot R$ vs. $72.07 \cdot R$ when $w = 8$ bits, and $55.27 \cdot R$ vs. $36.04 \cdot R$ when $w = 16$).

Moreover, as for bitmaps, increasing too much the number of events that can be collected by a node into the same message is often pointless, because bandwidth may be wasted uselessly.

3) Bit rate switching: Bit rate switching in CAN FD (by setting BRS to recessive) is an effective way to improve the notification rate. In this case, the transmission speed is increased for the part of frame included between the sampling points of the BRS bit and the CRC delimiter. For the FD base format (FDFF), this means that 29 bits (arbitration field, initial part of the control field, and most of the trailer) are sent at the nominal bit rate R, while $38 + 8D$ bits (final part of the control field, plus data and CRC fields) are transmitted at the (higher) data bit rate $\alpha \cdot R$. For example, when $\alpha = 5$ and $D_\mathsf{m} = 512$, the maximum notification rate, when bitmaps are used, can be as high as $\Lambda_\mathsf{m} = 3683.5 \cdot 10^{-3} \cdot R$ (the largest value achievable in CAN). However, when its effective value is considered, it shrinks to $\lambda_\mathsf{m}^{std} = \bar{\nu}_\mathsf{m} \cdot 7.19 \cdot 10^{-3} \cdot R$, slightly lower than when bitmaps are used with maximal-size classical CAN frames.

Table II reports a synoptic about the simple performance metrics we considered, for a number of approaches based on conventional CAN (either classical or FD), which rely on flat message assignment, event bitmaps, and event lists. The most interesting columns are $\mathcal{E}_\mathsf{m}$ and λ_m. Calculations about the event notification rate in overclocked CAN FD are quite trivial, and hence no values are explicitly included in the table.

VIII. Mapping events on CAN XR shared slots

Static slots offered by the CAN XR proposal permit multiple nodes to write dominant values in selected parts of the data field of the same message. It is worth remarking again that compliance to the CAN and CAN FD frame formats, including proper bit stuff insertion, is carried out by the initiator/supervisor, irrespective of the values the event sources (modeled as XR responders) actually write on the bus. We verified the proper operation of CAN XR by means of an experimental campaign on a software-defined CAN controller.

This behavior can be exploited to increase the event notification rate. In order not to loose the ability to carry out notifications in a truly distributed way, the *implicit initiator* feature of CAN XR has to be exploited [9]. In practice, any node wishing to notify an event initiates the related XR transaction on the bus. If there are other nodes in the same conditions, they join the data exchange as responders, including their events in the data field.

Let $\mathsf{X_S} = \{\mathsf{x}_1, ..., \mathsf{x}_{|\mathsf{X_S}|}\}$ be the set of XR transactions defined in the system to support event notifications. Although they are almost indistinguishable from other CAN messages, a different symbol has been used to improve clarity.

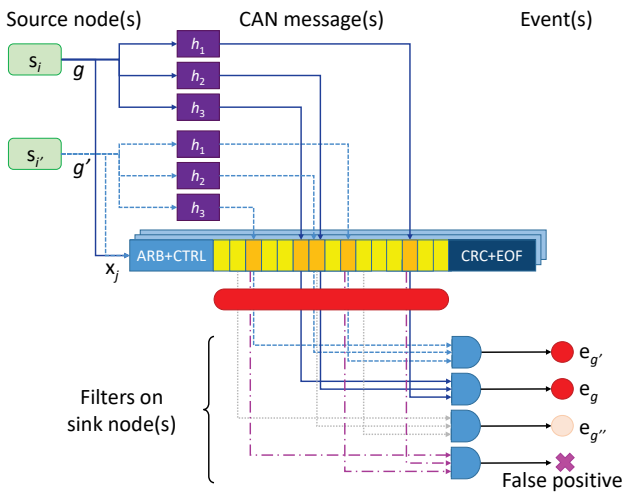

Fig. 2. Mapping of events in CAN XR using Bloom filters.

A. Mapping through event bitmaps

The data field of a CAN XR frame (or part of it) can be seen by nodes as made up of an array of slots, where each slot takes exactly one bit. A dominant slot value denotes the presence of the event, whereas a recessive value stands for its lack. If event mapping is flat, slots are *shared* among nodes and globally assigned to specific events, so that any node can set their value dominant. This is useful when the same event could be raised by a plurality of distinct sources. Conversely, if mapping is hierarchical, then each slot is *exclusively* assigned to a specific node. In this specific case, event lists can be employed as well, besides bitmaps, which require the size of the slots to be enlarged to $w > 1$ bits. Mixed solutions are also possible, but they are not considered here.

As the frame format in CAN XR is exactly the same as CAN/CAN FD, the maximum number $\mathcal{E}_\mathsf{x}$ of events that can be mapped onto an XR message x, and the maximum number $\mathcal{V}_\mathsf{x}$ of events that can be included in one of its instances, are the same as in non-XR cases. This also means that performance metrics, apparently, do not vary, for example, $\mathcal{E}_\mathsf{S} = |\mathsf{X_S}| \cdot \mathcal{E}_\mathsf{x}$. However, events in CAN XR are allocated to slots network-wide, and not on a per-node basis. This may lead to a dramatic improvement of the effective capacity of the network to quickly deliver notifications, since a single CAN XR message can collect events generated, almost at the same time, by a plurality of nodes. In this way, a larger number of pending events are likely to be collected in the same message than in the case of conventional CAN, where event gathering can be carried out only in hierarchical schemes and on a local basis.

Roughly speaking, in solutions based on conventional CAN messages, like those analyzed in the previous sections, the average number of pending events network-wide can be approximated as $\bar{\epsilon}_\mathsf{S} = |\mathsf{M_S}| \cdot \bar{\epsilon}_\mathsf{m}$, while in CAN XR they are $\bar{\epsilon}'_\mathsf{S} = |\mathsf{X_S}| \cdot \bar{\epsilon}_\mathsf{x}$. In systems with a large number of devices, each of which generates a small number of events, $|\mathsf{X_S}| \ll |\mathsf{M_S}|$.

TABLE III
EVENT NOTIFICATION SCHEMES BASED ON CAN XR (BITMAPS AND BLOOM FILTERS)

Scheme	D (B)	ID	$\mathcal{E}_{\mathsf{S}}$	$\mathcal{E}_{\mathsf{x}}$	$\mathcal{V}_{\mathsf{x}}$	$\bar{w}$	Λ_{S} (kHz)	λ_{x} (kHz)	P_{fp}	Notes
XR bitmap	64	std	$\sim 1 \cdot 2^{20}$ $(1.05 \cdot 10^6)$	512	512	—	$884.28 \cdot R$	$\bar{\nu}_{\mathsf{x}} \cdot 1.73 \cdot R$	0%	$\bar{\nu}_{\mathsf{x}}$ refers to a single XR message
XR bitmap	64	ext	$\sim 256 \cdot 2^{30}$ $(275 \cdot 10^9)$	512	512	—	$856.19 \cdot R$	$\bar{\nu}_{\mathsf{x}} \cdot 1.67 \cdot R$	0%	$\bar{\nu}_{\mathsf{x}}$ refers to a single XR message
XR Bloom	64	std	∞	∞	64	8	$110.54 \cdot R$	$\bar{\nu}_{\mathsf{x}} \cdot 1.73 \cdot R$	2.17%	$\bar{\nu}_{\mathsf{x}}$ refers to a single XR message
XR Bloom	64	std	∞	∞	32	16	$55.27 \cdot R$	$\bar{\nu}_{\mathsf{x}} \cdot 1.73 \cdot R$	0.139%	$\bar{\nu}_{\mathsf{x}}$ refers to a single XR message
XR Bloom	64	std	∞	∞	16	32	$27.63 \cdot R$	$\bar{\nu}_{\mathsf{x}} \cdot 1.73 \cdot R$	0.00633%	$\bar{\nu}_{\mathsf{x}}$ refers to a single XR message

For instance, if there are 100 nodes and 500 events, $|\mathsf{M}_{\mathsf{S}}|$ has to be at least 100. Conversely, a single XR transaction (i.e., $|\mathsf{X}_{\mathsf{S}}| = 1$) mapped on an FD frame permits to encode all such events at once using a bitmap. If $\bar{\epsilon}'_{\mathsf{S}}$ was equal to $\bar{\epsilon}_{\mathsf{S}}$, then $\bar{\epsilon}_{\mathsf{x}} \gg \bar{\epsilon}_{\mathsf{m}}$, which means that, on average, a larger number of pending events can be collected together, and hence a smaller portion of the capacity of XR messages goes wasted. In reality, improvements are not so high since, due of the higher capacity of XR-based solutions to drain notifications, the mean number of pending events (all over the system) shrinks, i.e., $\bar{\epsilon}'_{\mathsf{S}} < \bar{\epsilon}_{\mathsf{S}}$.

The price to pay for the increase in the overall notification rate is that the maximum number of distinct events that can be defined system-wide is relatively small. Using multiple CAN XR messages, on which distinct events are mapped, permits to overcome this limitation. If a number $|\mathsf{X}_{\mathsf{S}}|$ of such messages are used, on which different global events are encoded as a bitmap, then $\mathcal{E}_{\mathsf{S}} = |\mathsf{X}_{\mathsf{S}}| \cdot 512$ (in the case XR transactions are mapped on FD frames). When doing so, however, events are scattered across more than one XR message, which means that $\bar{\epsilon}_{\mathsf{x}}$ is likely to decrease by about the same factor, and so does the effective notification rate λ_{x}. Again, the best option is to use as few messages as possible.

B. Bloom Filters

As Fig. 2 shows, applying Bloom filters to event notification in CAN XR is quite straightforward:

1) A *shared* slot is defined in the data field of a CAN XR message (possibly taking all D bytes) and used as the supporting data structure. Basically, it mimics a shared bitmap made up of $m = 8D$ entries.
2) When an event e_g has to be notified, the related *source* determines which bits have to be set dominant by evaluating k independent hash functions $h_i(g), 1 \leq i \leq k$, in g, each of which returns an index in the range $[0...m-1]$. More than one event may be included by each node in the same message.
3) The dominant–recessive behavior of the CAN bus is exploited to *merge* results. Having a node writing dominant values on specific bits of a shared slot corresponds to the insert-only operation carried out by Bloom filters on the data structure.
4) The content of the shared slot in the CAN XR message exchanged on the bus corresponds to the data structure after all the pending events have been inserted. This frame is received by all *sinks* at the same time.

Clearly, it is just impossible for a sink to obtain the original events back: in fact, Bloom filters rely on hash functions and are not intended to be reverted. However, every sink can easily assess whether or not the events it is interested in have been included by the related source. Simply, it has to evaluate the hash functions for these events, and check if all the corresponding bits in the received message are dominant.

What is particularly relevant about Bloom filters coupled with CAN XR shared slots is that they permit to map events drawn from a very large set (much larger than allowed by a bitmap) onto one (or few) XR messages. This makes them advantageous for systems where a huge amount of distinct events are foreseen (many thousands to millions), generated by a large number of devices (many hundreds), but each one of them occurs seldom.

By referring to the typical notation used for Bloom filters, reported in Section IV, the number n of elements inserted in an XR transaction x is on average $\bar{\nu}_{\mathsf{x}}$, and, if the whole data field is used as the supporting data structure, the related size is $m \equiv 8D_{\mathsf{x}}$. Let $\bar{w} = m/n = 8D_{\mathsf{x}}/\bar{\nu}_{\mathsf{x}}$ be the mean number of bits per event in x. The probability of false positives P_{fp} depends on $\bar{w}$ and, to a lesser extent, k. When using Bloom filters, $\bar{\nu}_{\mathsf{x}} = \bar{\epsilon}_{\mathsf{x}}$ because all pending events can in theory be collected in the same XR message. If only one message is considered in the system, then $\bar{\epsilon}_{\mathsf{S}} = \bar{\epsilon}_{\mathsf{x}}$ and, consequently, $\bar{\nu}_{\mathsf{x}} = \bar{\epsilon}_{\mathsf{S}}$. In this case, $\bar{w} = 8D_{\mathsf{x}}/\bar{\epsilon}_{\mathsf{S}}$.

As an example, let us assume that, on average, $\bar{\epsilon}_{\mathsf{S}} = 64$ events are pending, at any time, network-wide, which have to be notified as soon as possible, and that $D_{\mathsf{x}} = 64$ bytes (i.e., 512 bits). This means that $\bar{w} = 8$ bits in each XR message are allocated on average per event. In such conditions, the overall effective notification rate is $\lambda_{\mathsf{S}} = 110.53 \cdot 10^{-3} \cdot R$. When $\bar{\epsilon}_{\mathsf{S}}$ is cut by half and by 4 (which means that $\bar{w} = 16$ and 32 bits), the notification rate λ_{S} falls to $55.27 \cdot 10^{-3} \cdot R$ and $27.63 \cdot 10^{-3} \cdot R$, respectively. According to Section VI-B3, the probability P_{fp} of false positives for these three cases, when $k = 5$, is about 2.17%, 0.139%, and 0.00633%.

A synoptic about the metrics we took into account, for the case of CAN XR, when either bitmaps and Bloom filters are adopted, is shown in Table III. As can be seen, Bloom filters have two main disadvantages with respect to the case when a bitmap is directly coupled with CAN XR: first, the notification rate is noticeably lower (ten times or more), and second, false positives are possible. However, they also show a peculiar advantage, since the number of events that can be potentially

defined in the system, even with a single XR message, is virtually unlimited, $\mathcal{E}_{\mathsf{S}} = \infty$.

For these reasons, Bloom filters on CAN XR are mostly suitable for dealing with rare events in large systems, and should be more correctly compared against classical CAN solutions which rely on either a flat mapping on extended frames ($\mathcal{E}_{\mathsf{S}} \simeq 537 \cdot 10^6$, $\Lambda_{\mathsf{S}}^{ext} = 14.93 \cdot 10^{-3} \cdot R$) or a hierarchical mapping using event lists where, e.g., $D_{\mathsf{m}} = 2$ bytes and $w = 16$ bits ($\mathcal{E}_{\mathsf{S}} \simeq 134 \cdot 10^6$, $\Lambda_{\mathsf{S}}^{ext} = \bar{\epsilon}_{\mathsf{S}} \cdot 15.87 \cdot 10^{-3} \cdot R$). When false positives up to 2.17% are tolerated, coupling CAN XR and Bloom filters is about 7 times faster than CAN.

An important aspect to be taken into account when using CAN XR with Bloom filters is that the number of events system-wide to notify at any given time is not known a priori, and cannot be checked at run time because there is no coordination among nodes. While the number of events that can be "inserted" in an XR message is unbounded, when it increases above the expected value the probability of false positives may become unacceptably high. So as to lower the likelihood of this condition happening, multiple XR messages can be foreseen, and events can be scattered among them in the configuration phase. Importantly, in this case increasing the number of messages is not meant to enlarge $\mathcal{E}_{\mathsf{S}}$, but just to reduce statistically the occurrence of false positives.

IX. Conclusions

Controller Area Networks are very suitable to interconnect devices in event-driven systems, where interactions occur asynchronously. Besides real-time applications, they can be profitably employed also in those cases where information to be exchanged is characterized by relaxed timing constraints. For instance, wired sensor networks used for online diagnostics, as well as for reactive and proactive maintenance, can be inexpensively implemented and deployed using this communication technology.

In this paper, best-effort techniques for efficiently managing event notifications in such a kind of systems have been considered, and their performance evaluated by means of quite simple and generic metrics, like the maximum number of distinct events supported by each solution and the rate at which events are transferred from sources to sinks over the network. Concerning the underlying communication technology, both classical CAN and CAN FD have been taken into account, also including the recent CAN XR proposal, which enables data slotting in CAN without losing backward compatibility

Results show that CAN XR, by allowing multiple nodes to be writing at the same time into the same message, permits a higher number of events to be collected (and exchanged) together, which in turn increases the effective overall notification rate. The use of Bloom filters, possibly coupled with CAN XR, is useful in the case of systems where a very large number of rare events are defined, provided that false positives are occasionally tolerated.

with existing devices and systems. To improve communication efficiency, techniques can be employed that allow a set of events to be gathered in the same message. Besides obvious solutions like bitmaps and lists, Bloom filters were also considered to this purpose.

As future work we plan to assess performance by considering some specific event generation schemes for nodes. Doing so will probably require a suitable ad-hoc simulator to be purposely developed.

Acknowledgment

This research has been supported in part by the US National Science Foundation (CNS Grant No 1646317).

References

[1] 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.

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

[3] G. Cena, I. Cibrario Bertolotti, T. Hu, and A. Valenzano, "Design, verification, and performance of a MODBUS-CAN adaptation layer," in *Proc. 10th IEEE International Workshop on Factory Communication Systems (WFCS)*, May 2014, pp. 1–10.

[4] *CAN with Flexible Data-Rate Specification Version 1.0*, Robert Bosch GmbH, Apr. 2012.

[5] H. A. Hansson, T. Nolte, C. Norstrom, and S. Punnekkat, "Integrating reliability and timing analysis of CAN-based systems," *IEEE Transactions on Industrial Electronics*, vol. 49, no. 6, pp. 1240–1250, Dec. 2002.

[6] R. Davis, A. Burns, R. Bril, and J. Lukkien, "Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised," *Real-Time Systems*, vol. 35, no. 3, pp. 239–272, 2007.

[7] R. I. Davis and N. Navet, "Controller area network (CAN) schedulability analysis for messages with arbitrary deadlines in FIFO and work-conserving queues," in *Proc. 9th IEEE International Workshop on Factory Communication Systems (WFCS)*, May 2012, pp. 33–42.

[8] M. Di Natale and H. Zeng, "Practical issues with the timing analysis of the Controller Area Network," in *Proc. 18th IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, Sept 2013, pp. 1–8.

[9] G. Cena, I. Cibrario Bertolotti, T. Hu, and A. Valenzano, "CAN XR: CAN with eXtensible in-frame Reply," in *Proc. 14th IEEE Intl. Conference on Industrial Informatics (INDIN)*, Jul. 2016, pp. 1198–1201.

[10] 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.

[11] 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.

[12] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: a scalable wide-area Web cache sharing protocol," *IEEE/ACM Transactions on Networking*, vol. 8, no. 3, pp. 281–293, Jun. 2000.

[13] P. Cao, "Bloom filters — the math," Available online, at http://pages.cs.wisc.edu/~cao/papers/summary-cache/node8.html, Mar. 2017.

[14] G. Cena and A. Valenzano, "Overclocking of Controller Area Networks," *Electronics Letters*, vol. 35, no. 22, pp. 1923–1925, Oct. 1999.

[15] F. Hartwich, "CAN with flexible data-rate," in *Proc. Intl. CAN Conference (iCC)*, Mar. 2012, pp. 14-1–14-9.