Seamless Integration of CAN in Intranets

Gianluca Cena, Ivan Cibrario Bertolotti, Tingting Hu, Adriano Valenzano

Authors' Accepted Manuscript (AAM): This is the *accepted* version of the paper

Gianluca Cena, Ivan Cibrario Bertolotti, Tingting Hu, and Adriano Valenzano, *Seamless Integration of CAN in Intranets*, Computer Standards & Interfaces, vol. 46, pp. 1–14, doi 10.1016/j.csi.2015.11.004, May 2016.

available at https://doi.org/10.1016/j.csi.2015.11.004.

Seamless Integration of CAN in Intranets

Gianluca Cena[a], Ivan Cibrario Bertolotti[a,*], Tingting Hu[a,b,*], Adriano Valenzano[a]

[a] *CNR – National Research Council of Italy, IEIIT, c.so Duca degli Abruzzi 24, I-10129 Torino, Italy*
[b] *Politecnico di Torino, Dipartimento di Automatica e Informatica, c.so Duca degli Abruzzi 24, I-10129 Torino, Italy*

Abstract

Controller Area Network (CAN) is very popular in networked embedded systems. On the other hand, intranets are now ubiquitous in office, home, and factory environments. Namely, the Internet Protocol (IP) is the glue that permits any kind of information to be exchanged between devices in heterogeneous systems.

In this paper, a network architecture to seamlessly integrate CAN buses in intranets is described. Flexibility and scalability were the key design requirements to conceive a comprehensive solution that suits both inexpensive and very complex applications. A prototype implementation has been tested to confirm architecture feasibility and assess the performance it can achieve.

Keywords: IP protocol integration, Real-time distributed systems, Controller Area Network

1. Introduction

CAN is a real-time communication protocol that was purposely conceived for use in the automotive field. Not only it succeeded to become the "de facto" standard for connecting electronic control units (ECU) in cars, trucks, and other vehicles, but also it achieved good diffusion in factory automation and networked embedded systems.

CAN is very stable and quite inexpensive. Many recent microcontrollers embed one or more CAN controllers, which makes this communication technology extremely cost-effective. Unfortunately, it struggles to keep the pace with newer solutions because of the limitations affecting its network extension and transmission speed (1 Mb/s at best for buses shorter than 40 m). Conversely, real-time Ethernet (RTE) networks run at 100 Mb/s, which is (at least) two orders of magnitude higher than CAN. For these reasons, CAN is now deemed mostly suitable for application fields where good reliability is required along with very low implementation and deployment costs. For instance, it is increasingly popular in networked embedded systems.

In the past decade, the world of automation has witnessed the progressive move of industrial communications from fieldbuses to RTE solutions [1]. Besides a noticeable increase in performance and, especially, network throughput, RTE technologies typically enable seamless (non real-time) connectivity on field devices at both the Ethernet and IP levels. This permits an unprecedented degree of integration between real-time control systems located at the shop-floor and the applications found in the upper levels of the automation pyramid—e.g., manufacturing execution systems (MES) and enterprise resource planning (ERP). A similar move is currently taking place in other applications fields, e.g., power distribution [2] and vehicular systems [3].

So far, CAN has not been affected noticeably by this trend, mainly because it relies on a very simple protocol, targeted at connecting inexpensive equipment in automotive and networked embedded systems. Nevertheless, for the reasons previously mentioned, a uniform, well-agreed approach is more and more needed to achieve deeper integration between devices connected to CAN buses and factory/building communication infrastructures—which typically rely on intranet technologies.

Distributed control applications are not the only example where tangible benefits can be obtained by integrating CAN devices in higher-level networks, though. In fact, CAN could be profitably exploited to provide network connectivity which can be used to enable online condition monitoring of equipment. For instance, sensors aimed at measuring a number of quantities (thermal, infrared, acoustic, vibrational, etc.) involved in both corrective and proactive maintenance can be easily and inexpensively interconnected using built-in CAN controllers available in most existing microcontrollers. In these cases, exchanged data do not typically have any particular real-time requirements, whereas adequate degrees of flexibility (ensured by IP) and robustness (granted by CAN) are highly desirable. The combined adoption of CAN and powerline communication (PLC) [4] may be envisaged in order to reduce the overall connection costs drastically.

It is clear that the ability to effectively collect all the information about maintenance—possibly along with other environmental measurements—by integrating the related subsystems (made up, e.g., of a number of sensors connected to a CAN bus) in the enterprise communication backbone, is likely to reduce downtimes and achieve higher plant availability.

*Corresponding author. CNR-IEIIT c/o Politecnico di Torino, c.so Duca degli Abruzzi 24, I-10129 Torino, Italy. Tel.: +39 011 0905432, Fax: +39 011 0905429.

Email addresses: gianluca.cena@ieiit.cnr.it (Gianluca Cena), ivan.cibrario@ieiit.cnr.it (Ivan Cibrario Bertolotti), tingting.hu@ieiit.cnr.it (Tingting Hu), adriano.valenzano@ieiit.cnr.it (Adriano Valenzano)

The idea of enabling IP-based communication in CAN is not new. The Internet Draft [5] defines a protocol for transporting IP datagrams over CAN, including addressing aspects. In [6] a gateway between Internet and CAN is described along with a prototype implementation. However, the implementation is based on Linux PCs and may not be readily adapted to embedded systems, which are much more resource-constrained. Two different approaches for the coexistence of IP and CAN on real embedded devices, targeted at vehicular systems, are presented in [7] and [8]. Of these, [8] actually proposes a tunneling scheme to forward CAN frames through a (wireless) IP network for automotive testing applications, which is the opposite of what is proposed in this paper. On the other hand, the method presented in [7] supports only a single, rather than multiple, IP↔CAN gateway, thus posing strong constraints on possible network topologies. Moreover, it is based on a custom IP protocol stack and, according to the experimental results shown in the paper, seems unable to fully utilize the CAN bus bandwidth, due to either software or hardware performance limitations.

In all cases, the above-mentioned approaches mainly deal with the protocol used to transmit datagrams over CAN. Conversely, this paper primarily focuses on the overall network architecture, which permits CAN buses (and the related devices) to be seamlessly connected to intranets (e.g., factory backbones) and the Internet as well.

The first requirement that drove this design is *flexibility*. Besides supporting IP communication between end nodes, transmission of Ethernet frames is also foreseen, which enables switchports to be set up. In particular, a multi-point tunnelling is defined that permits Ethernet (*payload protocol*) to be transparently layered on CAN (*delivery protocol*). While doing so is typically pointless, nevertheless it is sometimes required so as to support protocols that do not rely on IP, e.g., the Link Layer Discovery Protocol (LLDP) [9], or to connect devices operating at the data-link layer, like access points.

The second key requirement is *scalability*. In the simplest cases, CAN nodes are connected to the intranet through simple routers. Both nodes and routers can be implemented inexpensively by using available open-source software (e.g., lwIP [10]). Conversely, for more complex systems suitable switches are defined, which permit bridged networks to be deployed by seamlessly interconnecting CAN buses, Ethernet links, and wireless extensions based on Wi-Fi.

The paper is organized as follows: in Section 2 the network architecture is introduced, while in Section 3 the transport protocol used for enabling data exchanges over CAN is described. Sections 4 and 5 present two architectural variants, which enable communication at the Ethernet and IP level, respectively. In Section 6 the prototype implementation is described and Section 7 discusses its performance.

2. Integrating CAN and intranets

Integration is a key requirement in every modern automation system [11]. To this extent, information must flow not only among devices in the same subsystem (horizontal integration), but also between systems located at different levels of the automation pyramid (vertical integration) [12]. In particular, the systems devoted to real-time control found at the shop-floor and based on fieldbus technology, as well as those used to carry out non-real-time maintenance, must be able to interact with office-level IP-based networks. The resulting interconnected system may be quite complex and include—besides real-time networks—the plant backbone, wireless LANs, and also public data networks to support remote (non real-time) access. This is the case, for instance, of the virtual automation networks (VAN) project described in [13]. Very similar considerations apply to CAN-based systems. In this case, besides factory automation, networked embedded systems are involved as well.

2.1. Integration of CAN buses and devices

Different approaches exist for integrating CAN into larger distributed systems—typically based on Ethernet—depending on the protocol layer at which interconnection is carried out.

The first class of solutions relies on suitable *gateways* that operate at the application layer. They are mostly commercial solutions, not backed by standard specifications. For instance, EtherCAN/2 foresees specific gateways for accessing CAN devices via Ethernet. To this purpose, the NTCAN [14] and EtherCAN Low Level Socket Interface (ELLSI) [15] application program interfaces are available for, e.g., Windows and Linux. Anybus [16] foresees network equipment aimed at connecting CAN devices to either other fieldbuses or industrial Ethernet. In this case, a specific kind of gateway is required for each host network, which means that they are not universal solutions. In many cases, mapping of functionality and behavior is not complete and not even perfect.

The second class carries out integration by means of *routers*, located below the application layer but strictly above the data-link (DL) layer. For instance, the framework defined by the Open DeviceNet Vendor Association (ODVA) foresees routers for interconnecting distinct networks, possibly based on different transmission technologies (CAN, Ethernet, etc.), provided that they rely on the Common Industrial Protocol (CIP). In this way, DeviceNet segments can be easily integrated in EtherNet/IP. Similarly, type-2 Ethernet POWERLINK (EPL) routers can be used to couple EPL and CANopen networks, enabling remote access to the object dictionary (OD) of nodes. These solutions offer seamless communication between devices, but they are limited to a specific application layer.

The last class relies on specific *bridges* that work at the DL layer according to the store and forward principle. They are used to couple distinct CAN buses, possibly with different bit rates. By decoupling MAC operations (mainly bus arbitration, as well as error detection and management), bridges permit increasing the network extension beyond the theoretical limits. Some of them enable selective forwarding based on message filtering (so as to reduce local traffic), possibly permitting identifier translation to achieve modular design. These solutions are mostly adopted for in-vehicle applications and are not suitable for interconnecting CAN to high-level networks. However, special bridges can be envisaged that link distinct CAN buses through Ethernet. They work by embedding CAN messages

(either one or more) into Ethernet frames. A discussion on this subject is reported in [17], while [18] deals with different ways of improving the related response times.

The main advantage of the above solutions is that, no modifications are required to the existing CAN devices. Coupling devices (gateways, routers, bridges) make conventional CAN nodes communicating with either devices connected to a distinct CAN bus or devices that are not directly provided with a CAN interface (e.g., those having Ethernet ports only). Although the basic timing properties of CAN are not retained completely (nor they could, because of the intermediate relay equipment and networks), such solutions may still ensure real-time behavior. Worst-case analysis of delays in bridged heterogeneous networks comprising CAN and switched Ethernet is reported in [19], whereas the case of CAN and Wi-Fi is dealt with in [20].

In this paper, a different approach is followed. In particular, integration can be carried out at either the Ethernet or IP level. While requiring modifications to the firmware of CAN devices, this enables them to communicate using potentially *every* kind of protocol defined for Ethernet and IP, respectively. Moreover, the behavior of the coupling devices can be standardized under almost every point of view—besides addressing, as will be explained in the following—irrespective of the specific underlying CAN-based network and high-level protocol.

It is worth remarking that such an approach was not conceived bearing real-time communications in mind —including process data exchanges and asynchronous event notifications. To this aim, existing CAN-based solutions can be exploited (coexistence with them is a key point that has to be carefully considered). Instead, its purpose is to achieve seamless connectivity between CAN devices and intranets. It is mostly targeted at operations like remote configuration, monitoring, and diagnostics, which can be layered on top of IP and carried out using, e.g., a conventional web browser. While making CAN devices a bit more complex, doing so removes the need for dedicated management tools. For this reason, it should be regarded as a long term solution, meant to be adopted as a universal replacement for non real-time operations, which can be carried out according to a unified approach.

2.2. *Interconnection framework*

Intranets are non-global computer networks that rely on IP and are deployed in the context of a specific organization (enterprise, plant, campus, etc.). Each such network consists of a number of subnetworks, referred to as *IP subnets*, which are interconnected through *routers*. The IP model refers generically to the underlying transmission networks as *links*, denoted DL-networks in the following. Each IP subnet may either coincide with a single DL-network or include a number of them—possibly based on different (but uniform) DL protocols—connected through bridges. Ethernet and Wi-Fi are two technologies typically employed in intranets, which rely on switches and access points, respectively. A bridged DL-network is seen by routers as a single link that, as a whole, behaves as a broadcast domain.

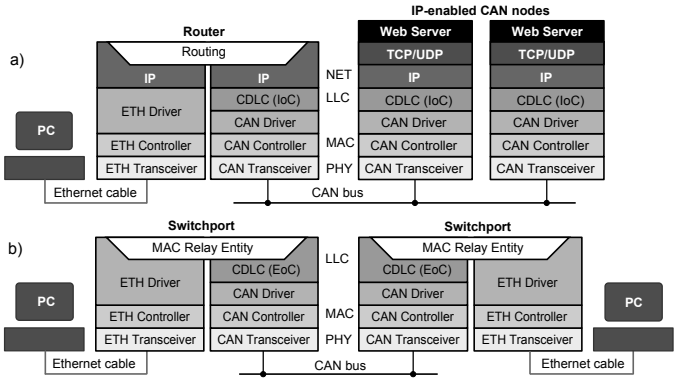

Figure 1: Sample systems exploiting interconnection of CAN and intranets.

Table 1: Table of Abbreviations

Abbreviation	Meaning
CDLC	CAN-based data link control protocol
DL-entity	Entity that implements CDLC
DLPDU	Data link protocol data unit, CDLC in this case
IoC	IP over CAN protocol
IP_ADDR	IP address
IP-SA	IP source address
IP-DA	IP destination address
EoC	Ethernet over CAN protocol
C-SW	C-switch, CAN/Ethernet switchport or bridge
C3-SW	Layer-3-enabled C-switch
TARP	Transparent address resolution process
EFDB	Extended filtering database
MAC_ADDR	Media access control (MAC) address
PORT_ID	C-switch port identifier
PORT_TYPE	C-switch port type, either CAN or Ethernet
CAN_ADDR	CAN node address
CAN-SA	CAN source address
CAN-DA	CAN destination address
NC-MSA	Network control MAC source announce
AoC	Address resolution protocol (ARP) over CAN
SHA	Sender hardware address
THA	Target hardware address
SPA	Sender protocol address
TPA	Target protocol address
FMA	Forged MAC address
FORGED_OUI	3-byte prefix of FMA, reserved for IoC
CANBUS_ID	16-bit CAN bus identifier of an IoC node
CAN_ADDR	8-bit CAN address of an IoC node
L-FMA	Locally-unique FMA
N-FMA	Network-wide unique FMA
UNAT	Uniform network addressing and traversal

The sample network architecture depicted in Fig. 1a can be used to enable IP communication over CAN, whereas Fig. 1b concerns CAN switchports. In both cases a number of generic modules are included: H/W components (controllers/transceivers for CAN and Ethernet), the related device drivers, S/W modules related to the upper layers of the protocol stack (TCP/UDP/IP), and application programs implementing high-level network protocols (e.g., a web server).

Enabling Ethernet and IP communication over CAN requires that a specific *DL transport protocol* is defined—not to be confused with TCP and UDP, which belong to the transport layer—whose purpose is to overcome the limitations of CAN. In the following, it will be referred to as *CAN-based Data Link Control* (CDLC). To make reading easier, all abbreviations introduced and used in this section and the following ones are summarized in Table 2.2.

Basically, CDLC is located immediately atop CAN drivers—i.e., it relies on the logical link control (LLC) services of CAN—and offers the upper layers a suitable interface, which permits the related *protocol data units* (PDU) to be transferred properly. CDLC must deal with two peculiarities of CAN, which make it unsuitable for direct transmission of Ethernet frames and IP datagrams, namely the limited payload size and the object-based addressing scheme.

The entity that implements CDLC in any CAN node is referred to in the following as *DL-entity*. This is because it offers the typical services that the upper layers (e.g., IP) expect from the data-link layer. From a conceptual point of view, CDLC operates by exchanging DLPDUs between DL-entities, where each DLPDU conveys exactly one PDU of the payload protocol (Ethernet frame, IP datagram, etc.) and is exchanged through one or more messages of the delivery protocol (CAN).

2.3. Approaches for Interconnecting CAN and Intranets

Two different approaches can be devised in order to provide CAN devices with IP connectivity. Either IP is stacked directly above CDLC or the latter has to mimic Ethernet communication. In both cases CDLC permits DLPDUs to be exchanged on the CAN bus. The main difference between these approaches concerns addressing.

Although it may seem counter-intuitive, stacking IP directly on CAN is the most straightforward solution. In the following it will be referred to as *IP over CAN* (IoC). CDLC transfers whole IP datagrams in IoC. In particular, the source and destination IP addresses (IP-SA and IP-DA) are used by IoC nodes and routers to accomplish DLPDU routing. A second solution is to equip CAN nodes with an interface to the upper layers that resembles Ethernet, so that existing IP implementations can be used with minor modifications. Every DL-entity, in this case, must be assigned a globally unique MAC address. This is referred to as *Ethernet over CAN* (EoC).

Working at the DL layer is more complex but achieves greater flexibility. Several cases can be distinguished, depending on the number and type of network interfaces: EoC nodes have only one interface (used to connect to CAN), whereas switches to be used with EoC—here denoted *C-switches* (C-SW)—must have two or more (either CAN or Ethernet). More specifically, a C-SW having only one CAN interface is known as *switchport*, while those provided with CAN interfaces only (at least two) are called *CAN bridges*.

Supporting more than one payload protocol implies that every DLPDU must be tagged with the related protocol. Reasonably, Ethernet, IPv4, and ARP have to be considered, along with a set of network control (NC) messages used for coordinating DL-entities over CAN. However, it should be possible to include other protocols as well, e.g., IPv6. In this paper, IPv4 was explicitly considered, as overheads in IPv6 are noticeably higher and hardly fit the CAN bandwidth. Thus, encapsulating IPv4 in CDLC is probably the most reasonable option at the moment. Nevertheless, this is not a limiting choice for three reasons: 1) the very same EoC/IoC approach can be easily applied to IPv6; 2) IPv4 and IPv6 are perfectly interoperable; 3) the recently introduced CAN with flexible data rate (CAN FD) [21], given its higher throughput, may be fast enough for IPv6.

In the last case, our solution can be taken as the basis for a "wired" version of 6LoWPANs, which achieves inexpensive implementations with reduced wiring costs thanks to the use of bus topologies and supports uninterruptible operation by powering devices through the same cable used for communication, hence enabling the Internet of Things (IoT) in automation environments [22] (also known as industrial IoT).

3. CAN-based Data Link Control

CDLC has to tackle aspects related to both payload size and addressing. Its exact definition is, under many respects, uncorrelated with the network architecture. For instance, a suitable solution is defined in [5]. In this paper a simpler protocol has been developed, which proved to be quite stable and efficient.

3.1. Payload size

One of the main problems of transferring Ethernet frames or IP datagrams over CAN is the very different size of the *maximum transmission unit* (MTU) for the two protocols. For example, an Ethernet frame (including those encapsulating an IP datagram [23]) can be up to 1500 bytes in length, whereas the data field in CAN is able to accommodate 8 bytes at most. A similar problem affects the direct transmission of IP datagrams. Although a fragmentation protocol is defined in IPv4 [24], which permits dealing with the different MTU size of the underlying networks, the minimum fragment size is by far too large for a single CAN message.

A straightforward solution to the MTU problem is to exploit fragmentation at the CDLC level. This means, that every DLPDU is split into a number of smaller fragments, which are sent in sequence over the CAN bus and reassembled at the target node(s). Fragmentation is customarily adopted in many application-layer protocols based on CAN. A popular example are service data objects (SDO), used to transfer object dictionary entries in CANopen [25]. DeviceNet [26] defines its own fragmentation protocol, too. Unlike SDOs, it permits peer-to-peer data transfers, but connections have to be created explicitly in advance.

When transferring Ethernet frames over CAN, frame preamble and interframe gap are not taken into account. Including the frame check sequence (FCS) would permit receivers to check the frame exchange on the whole. However, thanks to its error detection and globalization mechanisms, CAN is deemed reliable enough to avoid using this FCS, as witnessed by the fact that neither CANopen nor DeviceNet use it. In the case of IP the entire datagram is transported, including the header field (20 bytes plus options). Doing so permits existing S/W modules that implement IP to be used directly.

3.2. Addressing

As required by the IP protocol [24], CDLC must support true multi-peer communications through a *node-based* addressing scheme. Such a behavior is directly supported, for instance, by

Ethernet [27]. This means that every DL-entity must be able to explicitly address any other DL-entity connected to the same CAN bus and send it its own DLPDUs. This is in contrast with the *object-based* addressing scheme, specified in the CAN data link layer [28].

Before any messages can be exchanged over CAN, CDLC must therefore assign them suitable message *identifiers*, under the constraint that exactly one producer is allowed for any given identifier, so that arbitration always operates correctly. This implies that each DL-entity should be assigned one or more CAN identifiers for its exclusive use, which implicitly permit distinguishing it on the CAN bus (kind of a source addressing).

Concerning addressing information about the destination DL-entity on the CAN bus, it can be encoded either in the CAN identifier or inside the CAN data field. In the first case the number of allowed DL-entities on each CAN bus is quite low, because of the limited size of standard 11-bit identifiers. This limitation can be loosened noticeably by using extended 29-bit identifiers, but doing so decreases throughput by about 15% because of the related additional overhead. The addressing space could be also enlarged by using the second approach, but in this case frame filtering cannot be exploited on the receiver side so as to reduce the rate at which interrupts are generated—which is usually deemed unacceptable in embedded systems. For this reason, it is strongly suggested that *all* addressing information are stored in the identifier. To this extent two sub-fields are defined in the identifier, whose purpose is to encode the source and destination *CAN node addresses*, denoted CAN-SA and CAN-DA, respectively.

CAN node addresses are much smaller than MAC addresses (6 bytes) and IP addresses (4 bytes in IPv4), which implies that the maximum number of DL-entities allowed on the same bus is typically low. For instance, by reserving half of the standard identifiers in the network to CDLC, 5 bits at most can be devoted to encode each CAN address. This is not a severe drawback, given what the CAN specification recommends [29], that is, to connect a maximum of 30 nodes at a bus rate of 1 Mb/s and maximum bus length (40 m), when using an unshielded twisted pair (UTP) cable and ordinary transceivers. A higher number of nodes can be supported at the physical layer by using high input impedance transceivers but, as suggested above, extended identifiers are required for addressing. Else, distinct CAN buses can be interconnected through CAN bridges.

Whatever the addressing scheme, at least one address must be reserved to broadcast communication for the proper operation of CDLC. Possibly, half of the address space can be reserved to multicast addresses, which means that only 16 CAN nodes are allowed. An individual/global (I/G) bit is used to distinguish between unicast and multicast addresses. In any case, a single pattern, typically 11111_2, is devoted to broadcasts. Assignment of addresses to CAN nodes can be carried out statically, by using dip switches or some other local means (non-volatile memory). Alternatively, CAN addresses can be assigned dynamically (on demand) by using either a centralized or a distributed approach. A thorough description of these aspects is outside the scope of this paper.

3.3. Coexistence with other CAN-based solutions

Most high-level CAN-based protocols (CANopen, DeviceNet, etc.) were not defined taking coexistence with other competing solutions into account. Having EoC/IoC coexist with these protocols on the same bus mostly depends on identifier assignment. So that the new communication paradigm can be supported in existing systems, clashes among identifiers shall be avoided. For the above reason, a universal assignment scheme for CDLC likely does not exist.

In embedded networked systems, where assignment is carried out by the system designer with no particular restrictions—besides those mandated by schedulability analysis to meet timing constraints—the simple solution described in this paper may suffice. It reserves the most significant bit in the CAN identifier field to discriminate between CDLC (recessive) and other messages (dominant). This leaves half of the identifiers available for real-time exchanges and also ensures that the worsening CDLC causes on their latency is bounded.

Guaranteeing coexistence is a bit harder when CDLC has to be implemented in existing CAN-based networks. For instance, the number of identifiers left unused in the predefined connection set of CANopen is not sufficient to provide all nodes (up to 127 per network) with IP connectivity. In this case, switching to the extended identifier format (29 bits) is probably the best option. A satisfactory solution is to reserve the 7 least significant bits in the base identifier to encode CAN-DA, while the remaining 4 bits are set to the function code 1111_2 (not defined in the predefined connection set). In this way, regular CANopen messages are given precedence over CDLC, and the same filtering mask used for them can be exploited. CAN-SA, on which no filtering is carried out, takes 7 bits in the extended identifier, while the remaining 11 bits can be used for other purposes.

3.4. Preserving real-time performance

When different kinds of traffic coexist on the same CAN bus, it is generally impossible to completely avoid mutual blocking without resorting to some forms of coordination, e.g., the time-triggered communication paradigm [30]. Then, it becomes important to understand which effect IoC or EoC communication has on real-time (RT) traffic taking place on the same bus. In the following, it is assumed that CAN identifiers are assigned in such a way that RT messages always have higher priority than CDLC fragments. In other words, they belong to two disjoint sets and any message belonging to the first set has strict precedence over those belonging to the second. This is the case of the prototype CDLC implementation which is presented in Section 6.1.

In general, two sources of blocking can be identified:

1. *Bus-related* blocking, caused by multiple nodes trying to transmit concurrently on the same CAN bus.

2. *Queuing-related* blocking, which may be introduced when the same node generates different kinds of traffic through the same CAN controller.

For what concerns bus-related blocking, it can easily be proved that, if multiple frames compete for transmission in the same arbitration round, the highest-priority message will not suffer from any blocking. Thus, if a RT message competes with any number of CDLC fragments, it will not be delayed at all. Instead, if a RT message is queued for transmission when another message is already being sent on the bus, blocking will occur. The mutual blocking among multiple RT messages was thoroughly analyzed in [31]. Hence, here we focus on the effect of non-RT traffic. When a RT message is queued for transmission while a CDLC fragment is being sent on the bus, the former will be blocked until the next arbitration round, which takes place after the transmission of the fragment has finished. At that point, the RT message will surely win the arbitration against any other CDLC fragment. As a consequence, the worst-case effect of the whole set of messages used by CDLC can be modeled as a single additional RT stream, whose priority is lower than any RT message and whose messages are queued for transmission back to back. Its effects, including chain blocking, can be analyzed as in [31].

Queuing-related blocking may occur when either the CAN controller or the software device driver only support a First-In, First-Out (FIFO) transmission queue rather than a priority-based queue or multiple queues with different relative priorities. In this case, a RT message entering the queue will suffer blocking from any other messages already buffered in that node at that time, regardless of its priority. Scheduling analysis is still possible in this scenario, as shown in [32].

4. Ethernet over CAN

The aim of EoC is to make an application programming interface (API) available on CAN nodes that resembles closely the one provided by IEEE 802 protocols. Whole Ethernet frames are encoded by CDLC in EoC, including destination and source MAC addresses (DA and SA), but except the Preamble and the Frame Check Sequence (FCS).

Similarly to Ethernet, each CAN bus represents a broadcast domain. Therefore, having CDLC sending on the bus the fragments of every Ethernet frame using the broadcast CAN-DA ensures, in theory, proper EoC communication. In fact, receivers can reassemble all incoming DLPDUs and only then decide whether or not they are interested in the encapsulated Ethernet frame by inspecting its (MAC) DA field. However, doing so would cause an excessive interrupt rate, which is typically not acceptable since most CAN nodes have limited processing capabilities. Therefore, some mechanism is needed by nodes in order to discard CDLC fragments related to irrelevant frames, which exploits the frame acceptance filtering function provided by CAN controllers.

4.1. Mapping MAC and CAN addresses

In order to carry out the translation between MAC and CAN addresses a *transparent address resolution process* (TARP) is used by DL-entities (both C-switches and EoC end nodes) that resembles the learning and forwarding mechanism in Ethernet bridges. To this extent, an *extended filtering database* (EFDB) is required in C-switches, which consists of a number of entries. Each entry is related to a remote node and includes several information: its MAC address (MAC_ADDR), the port on which it was last heard (PORT_ID), and, implicitly, the related type (PORT_TYPE)—either Ethernet or CAN. In the case of CAN ports, the CAN address (CAN_ADDR) of the node bound to MAC_ADDR is also stored. Since EoC nodes are provided with a single port (of CAN type), a simplified version of EFDB is maintained by them, whose entries only include information about MAC_ADDR and CAN_ADDR.

TARP operates on each Ethernet frame separately and consists of a forwarding and a learning subprocess. Frames can be either generated by the upper protocol layers (IP) in the same device or received from the network. In the latter case, they can come from either Ethernet ports directly or CAN ports by means of the CDLC protocol.

4.1.1. Forwarding Subprocess

Whenever an Ethernet frame has to be dealt with, the EFDB is searched for an entry whose MAC_ADDR field matches the DA field in the frame. Four outcomes are possible:

a) A match is found with the MAC address specifically assigned to the local DL-entity: the frame is delivered to the upper protocol layers.

b) A match is found in the EFDB for a CAN port: a point-to-point fragmented transfer is carried out through CDLC to the specific destination node on the related CAN bus using the CAN_ADDR information in the entry as CAN-DA.

c) A match is found in the EFDB for an Ethernet port: the frame is relayed directly on that port, as per conventional Ethernet switch rules.

d) No match is found or DA specifies a multicast address: *flooding* takes place, i.e., the frame is relayed on all ports of the switch (both Ethernet and CAN), with the exception of the one on which it was received; the same applies to the unique port of EoC nodes; the broadcast CAN-DA is used by CDLC on CAN ports.

In order to support flooding, CDLC must be able to operate according to an unacknowledged transmission scheme.

4.1.2. Learning Subprocess

This process is used to populate the EFDB and resembles what happens in conventional Ethernet switches. The SA field is inspected in every received Ethernet frame (including those transferred through CDLC) and, depending on the specific condition, the following operations are carried out:

a) An entry already exists in the EFDB whose MAC_ADDR equals SA, but it is associated to a different port: the information in the entry is updated.

b) SA is unknown: a new entry is added to the EFDB where MAC_ADDR=SA; PORT_ID in the entry is set properly and PORT_TYPE as well.

If the frame came from a CAN port, CAN_ADDR in the entry is also set. In any case, the aging time of the entry is reset.

In order to keep EFDB replicas in different DL-entities on the same CAN bus in a coherent state, whenever a frame is relayed by CDLC to a specific destination (unicast CAN-DA) it is preceded by the transmission of a short *NC-MAC_Source_Announce* (*NC-MSA*) network control message, sent with the broadcast CAN-DA. Its purpose is to notify all the other DL-entities that an entry was either created or updated in the local EFDB of the sender, so that the change is propagated to all the other EFDBs. The only information this message has to include—besides CAN-SA—is the SA field of the subsequent Ethernet frame (6 bytes), which fits in the payload of a single CAN message. Upon reception of *NC-MSA*, every DL-entity updates its EFDB using the same rules described above concerning reception of Ethernet frames. Doing so is unnecessary when Ethernet frames are broadcast by CDLC on the CAN bus, since every node ought to read such frames in this case.

In theory, information in EFDB replicas for DL-entities on the same CAN bus might become incoherent following a transmission error that involves a frame or a NC message (because of CAN error globalization, this is unlikely). At any rate, this is not a serious problem: missing the creation of an entry in the EFDB simply delays (temporarily) selective forwarding—and causes a small amount of unnecessary flooding—while missing an update just increases the likelihood that the related entry is removed because of aging.

As for every Ethernet bridge that complies to IEEE 802.1D, the Spanning Tree Protocol (STP and RSTP) must be included in C-switches so as to avoid loops in the network topology.

5. IP over CAN

As a matter of fact, IP is currently layered above the transmission network (Ethernet, WiFi, public data networks, etc.) in almost every high-level non-real-time distributed application. For this reason, a second solution can be devised specifically for this kind of systems, namely IoC, where DLPDUs encode IP datagrams directly. This permits saving 14 bytes with respect to EoC (DA, SA, and ET fields). Most important, unlike EoC MAC addressing is not required: IoC nodes are identified on the whole network using IP addresses. This means that a mechanism is required to translate IP and CAN addresses directly.

Basic IoC implementation is straightforward: IP was conceived to run on different kinds of DL networks and CDLC is just one of them. IoC has to be regarded as a very simple approach (simpler than EoC) that permits CAN nodes to be easily connected to intranets.

5.1. Mapping IP and CAN Addresses

The address space for CAN nodes running IoC complies to the Classless Inter-Domain Routing (CIDR) scheme [33]. A router provided with both CAN and Ethernet interfaces is typically used to interconnect a CAN bus to the intranet, whose internal architecture resembles IoC devices. This means that it can be implemented on inexpensive embedded platforms.

Either static or dynamic mapping can be used for translating IP and CAN addresses. Assuming that standard CAN identifiers are in use, as discussed in Section 3.2, *static mapping* supports IoC networks with a `/27` CIDR prefix, which means that 27 of the 32 bits of the IP address represent the *network number* while the 5 least significant bits encode the *host number*. The host number is mapped one-to-one into the layer-2 CAN address. According to the requirements for Internet hosts [34], a host number of all ones (11111_2) is reserved in this case for directed broadcasts and directly corresponds to the broadcast address required by CDLC, as specified in Section 3.2.

Due to the presence of a class of hosts that use a non-standard host number of all zeros for directed broadcast, [34] also recommends that Internet hosts recognize and accept both. Following this recommendation has the side effect of making address 00000_2 unavailable for unicast communication when static mapping is in use. This brings the total number of supported nodes to 30 (if no addresses besides the broadcast address are reserved for multicast communication) or 15 (if they are, according to the approach proposed in Section 3.2). The IP address of an IoC node is obtained by concatenating the routing prefix of the CAN bus and the CAN address of the node. The reverse translation is based on the same principle.

Instead, with *dynamic mapping* IP and CAN addresses are uncorrelated, which means that they are assigned separately—and, possibly, in different stages of system design/deployment.

5.2. Address Resolution Protocol over CAN

A suitable mechanism is required for dynamic address translation in IoC. In particular, a streamlined version of ARP may suffice, referred to as *ARP over CAN* (AoC). As in ARP, AoC messages (request and reply, including gratuitous ones) include 4 fields, namely Sender/Target Hardware Addresses (SHA and THA) for encoding MAC addresses, and Sender/Target Protocol Addresses (SPA and TPA) for IP addresses. A cache is also defined whose entries relate IP and hardware addresses. To retain full compatibility with IoC, also EoC has to use AoC for IP address translation (in the place of ARP). As explained below, this requires negligible changes to implementations.

SPA and SHA in received AoC messages are used by EoC nodes to populate a conventional ARP cache (for managing the IP↔MAC mapping), which is layered above TARP and the related EFDB (which deals with the MAC↔CAN mapping). Unfortunately, IoC nodes do not have a MAC address. To achieve interoperability with EoC, non-globally unique *forged MAC addresses* (FMA) must be defined for them by concatenating a specific (reserved) 3-byte *IoC-prefix* (FORGED_OUI), a 16-bit CAN bus identifier (CANBUS_ID) and the 8-bit CAN address of the IoC node (CAN_ADDR). Thanks to the prefix, FMAs can always be recognized to belong to IoC nodes.

The reserved CANBUS_ID value `0x0000` denotes the *local* CAN bus. The related FMAs are referred to as *locally-unique FMAs* (L-FMA), and their uniqueness is ensured only on the bus to which the nodes are attached.

Each IoC node sets the SHA field in transmitted AoC messages to its own L-FMA. When sending an AoC reply, THA is

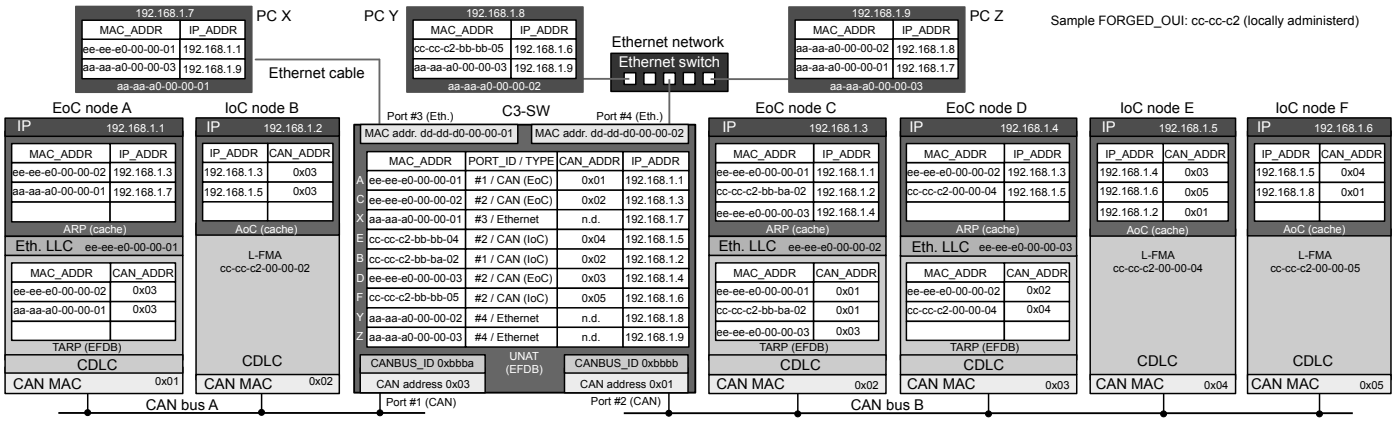

Figure 2: Sample network architecture including IoC, EoC, and Ethernet nodes, along with extended filtering databases and ARP/AoC caches.

set equal to the L-FMA corresponding to the CAN address of the destination on the local bus. This information is found in the local ARP/AoC cache. This implies that, unlike ARP, THA may differ from the SHA value included in the related request.

Concerning the received AoC messages, the values of SHA and THA are typically ignored, since they may refer to real MAC addresses, which cannot be understood by IoC. In fact, in the most general case the sender node can also be of EoC type (or, as described in the next section, a port of a C-switch.) Therefore, its hardware address has to be determined as the CAN-SA field used in the related CAN messages. Practically, before AoC messages are processed and the ARP/AoC cache is updated, the value of SHA is replaced by IoC nodes with the L-FMA derived from the CAN address of the sender (CAN-SA). Similarly, each IoC node replaces THA in received AoC replies with its own L-FMA (which corresponds to CAN-DA in the related CAN messages).

It is worth noting that, unlike IoC, EoC nodes actually make use of the value in the SHA field of the received AoC message. However, before processing replies coming from IoC nodes (which can be easily singled out, as they use L-FMAs in the place of real MACs), EoC nodes replace THA with their own MAC address.

The above arrangement is meant to ease implementations of EoC and IoC protocol stacks. From a practical point of view, in order to translate IP addresses into hardware addresses on both kinds of nodes, it is sufficient to add a thin software layer between ARP and CDLC, which simply carries out format translation between ARP and AoC messages (and vice-versa), including conversions between CAN addresses (CAN-SA and CAN-DA) and the related L-FMAs (stored in SHA and THA). Such an approach permits reusing (most of) the existing ARP implementations, including cache management.

5.3. Layer-3 Enabled Switches

Besides exploiting routers, transmission of IP datagrams inside bridged DL networks that include both Ethernet links and CAN buses could be accomplished also by means of switches. Thanks to the learning and forwarding approach, relaying DLPDUs at the data-link layer is noticeably more flexible than at the network layer. The problem is that the L-FMAs for IoC nodes are not unique across the whole IP subnet (possibly made up of a number of different heterogeneous links interconnected through C-switches), and hence they cannot be used to forward IoC datagrams out of a CAN bus.

A possible solution is to rely on a particular sort of C-switches that operate "above" layer 2. In particular, C-SW behavior has to be extended so that the IP address (IP_ADDR) can be possibly used, in the place of MAC_ADDR, to identify nodes uniquely in the EFDB and carry out datagram forwarding. Such scheme is referred to as *uniform network addressing and traversal* (UNAT) and the related devices are termed *layer-3-enabled C-switches* (C3-SW). UNAT unifies IP and MAC addressing by combining the learning and forwarding approach of TARP and the request-reply paradigm of ARP/AoC. Notably, it does not affect in any way the behavior of the interconnected end nodes (IoC, EoC, or Ethernet ones).

Fig. 5 depicts a sample heterogeneous class C IP subnet (192.168.1.0/24) where devices are connected by means of a switch that implements UNAT. As depicted in the figure, two kinds of entries are foreseen in the EFDB of C3-SWs, depending on the port type (either Ethernet or CAN). Besides the port they refer to (PORT_ID), the following information are stored in each entry:

1. Ethernet: ⟨IP_ADDR, MAC_ADDR⟩;
2. CAN: ⟨IP_ADDR, MAC_ADDR, CAN_ADDR⟩;

IoC end nodes are implicitly distinguished through the OUI in MAC_ADDR, which is set to the IoC-prefix (FORGED_OUI).

5.3.1. Forwarding Subprocess

Besides carrying out all the functions of conventional C-SWs on Ethernet and EoC frames, C3-SWs also act as proxies for IoC datagrams. When an IoC-encoded IP datagram is received (on a CAN port), the destination IP address (IP-DA) is searched in the EFDB. If a match is found with the IP-ADDR field of an entry of the EFDB, the related information is used. Before forwarding takes place on the port identified as PORT_ID, the following actions are carried out, depending on the port type:

- Ethernet: the datagram is embedded in an Ethernet frame where DA is set to MAC_ADDR. Since SA is not available for IoC datagrams, it can be set to either the globally-unique MAC address of the C3-SW or a MAC address purposely forged for CAN-SA so as to be unique on the whole network. The former approach is simpler but it may be weak against IP spoofing attacks [35]. In the latter approach a *network-wide-unique* forged MAC address (N-FMA) is created. To this extent, distinct CANBUS_IDs must be assigned to the CAN buses in the bridged DL network. It is worth noting that only C3-SWs (and not IoC nodes) are aware of CANBUS_IDs. This reduces implementation and network configuration efforts noticeably.

- CAN: in the case the destination is an IoC node, the IP datagram is relayed on the target CAN bus directly using CDLC and IoC rules. The CAN_ADDR field of the entry is used as CAN-DA while CAN-SA is set to the CAN address of the port of the C3-SW connected to the target bus. Conversely, if the destination is either an Ethernet or an EoC node, MAC addresses in the encapsulated frame are set in the same way as for the Ethernet case. From this point on, forwarding takes place at the data-link level and complies to EoC rules.

Instead, when an Ethernet frame is received—possibly on a CAN port through EoC—the EFDB is searched for either an Ethernet or EoC entry whose MAC_ADDR matches DA. If a match is found, forwarding is carried out using conventional Ethernet/EoC rules. Otherwise, when the frame encapsulates an IP datagram, the EFDB is searched again for an IoC entry whose IP_ADDR matches IP-DA. If it is found, relaying takes place in the same way as for IoC-encoded IP datagram.

Whatever the case, when a match is not found according to the above rules, the frame/datagram should be relayed on all ports of the C3-SW (flooding). Flooding of datagrams on CAN ports takes place as IoC using the broadcast CAN-DA—this is because EoC nodes can be easily enabled to decode IoC datagrams, while the opposite is not true. Such an event is indeed quite rare, since every transmission of a datagram to a host whose hardware address is unknown to the sender has to be preceded by an ARP message exchange, which initializes EFDBs suitably. Likely, it is due to the misalignment of the EFDB and ARP caches on end nodes.

In an attempt to avoid datagram flooding, an AoC request could be broadcast by the C3-SW on all its CAN ports (to ask for the related IP-DA) whenever a match is not found in the EFDB. If the original datagram was encoded as IoC, a similar ARP request is sent on Ethernet ports as well. When the node addressed by IP-DA replies, the EFDB in the C3-SW is coherent again and flooding is no longer necessary.

5.3.2. Learning Subprocess

UNAT ensures complete interoperability between Ethernet, EoC, and IoC in bridged DL networks. The related operations are carried out entirely by C3-SWs, so that implementation of IoC nodes remains as simple as possible. Generally speaking, the EFDB in C3-SWs acts as both filtering database and ARP cache. Unlike TARP in EoC, which obtains information about the position of nodes in the network by analyzing all Ethernet frames, the learning process used by UNAT to update the EFDB relies only on the ARP/AoC messages relayed by the C3-SW, according to the following rules:

- an AoC message is received (on a CAN port): if SHA is related to a L-FMA, the originator is an IoC node. In this case, the C3-SW forges a N-FMA for it and replaces SHA in the message body. At the same time an IoC entry is created/updated in the EFDB as ⟨IP_ADDR=SPA, CAN_ADDR=CAN-SA, MAC_ADDR=N-FMA⟩. Otherwise, the originator is either an EoC node or another C3-SW on behalf of an IoC node, and the entry is set to ⟨IP_ADDR=SPA, CAN_ADDR=CAN-SA, MAC_ADDR=SHA⟩.

- an ARP message is received (on an Ethernet port): an Ethernet entry is created/updated in the EFDB as ⟨IP_ADDR=SPA, MAC_ADDR=SHA⟩.

Generally speaking, ARP messages are forwarded by switches but they are blocked by routers. As a consequence, broadcast transmissions of ARP requests span the whole bridged DL network (including both Ethernet and CAN). In particular, AoC requests are distributed to all nodes on a CAN bus using the broadcast CAN-DA.

Instead, both ARP and AoC replies are delivered as unicast messages. The EFDB in the C3-SW is employed to forward them properly through the network, similarly to what happens to IoC datagrams. Basically, MAC addresses are considered first (either DA in ARP or THA in AoC) and, only in the case a match is not found in the EFDB, the IP address is taken into account (TPA). This is possible since each reply typically follows a specific request, which took care of suitably setting the relevant entries in the traversed C3-SWs.

While forwarding ARP messages from Ethernet to CAN, the C3-SW translates them to AoC. The reverse translation is carried out in the opposite direction.

When an AoC message generated by an IoC node is relayed by a C3-SW, SHA is checked. If it contains an L_FMA, it is converted to an N_FMA (this translation only takes place in the first traversed C3-SW). In the case the AoC message is relayed to Ethernet (encoded as ARP), the N_FMA in SHA is also used for SA. Instead, when the message is relayed (or sent) on CAN, the CAN address of the egress port of the switch (the one on which transmission takes place) is used as CAN-SA.

The effect of AoC messages on end nodes depends on their type. IoC nodes that receive one of such messages simply update their AoC cache with the entry ⟨IP_ADDR=SPA, CAN_ADDR=CAN-SA⟩. Conversely, EoC nodes update their ARP cache as ⟨IP_ADDR=SPA, MAC_ADDR=SHA⟩ and, through TARP, their EFDB as ⟨CAN_ADDR=CAN-SA, MAC_ADDR=SHA⟩.

Besides ARP/AoC caches in end nodes, each ARP/AoC request/response pair sets all the relevant EFDB entries in the traversed C3-SWs in both directions. In this way, all the subse-

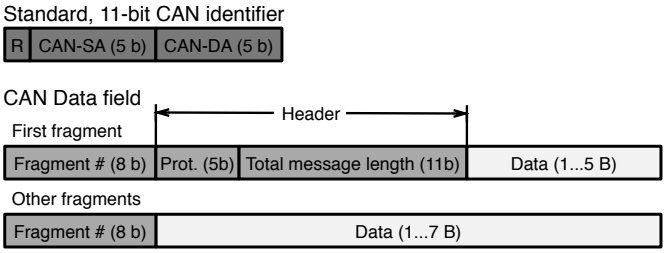

Figure 3: CAN identifiers and frame format in the CDLC protocol.

quent IP datagrams can be correctly forwarded by them to the intended destination.

6. Prototype Design and Implementation

This section discusses how a prototype system has been designed and implemented. First of all, the CDLC protocol implementation, common to both IoC and EoC, is presented in Section 6.1. Due to protocol overhead and implementation complexity considerations, as shown in Section 6.2, the choice for the first prototype fell on IoC, which has then been integrated in the open-source lwIP protocol stack [10], as a network interface driver module, presented in Section 6.3.

6.1. CDLC Protocol Implementation

As shown at the top of Fig. 3, the implementation makes use of standard CAN identifiers to hold the 5-bit CAN-SA and CAN-DA plus one reserved bit, denoted as R in the figure. CAN-SA (the *source* address) comes before CAN-DA, so that in arbitration it implicitly sets the relative priority of frames transmitted by different CAN nodes in case of bus contention. Hence, an appropriate choice of CAN-SA provides a simple way of assigning different, fixed priorities to IP traffic originating from distinct nodes.

Since the prototype implementation's goals were mainly *feasibility* and *performance* evaluation, the static address mapping approach described in Section 5.1 was adopted. There is in fact no doubt that an ARP-like protocol can be implemented with reasonable effort within modern protocol stacks (that already provide ARP themselves). Moreover, address mapping is used only in the very first stage of the communication between peers, and hence, it marginally affects communication performance.

Referring back to Fig. 3, the CDLC protocol transfers an IP datagram as a sequence of *fragments*, as specified in Section 3. All fragments start with a 1-byte *fragment number* at the beginning of the CAN data field. The first fragment has an additional 2-byte *header* containing a 5-bit *protocol identifier* and an 11-bit field containing the *total length* of the IP datagram to be transferred. The protocol identifier is set to 00001_2 for IoC while the other values are reserved for other protocols, namely EoC and network control, supported by CDLC. It is assumed that at most one pending transmission may exist between any two nodes at any given time. For this reason, the fragments bear no explicit information to determine whether or not they pertain to the same datagram.

In all fragments, the rest of the data field holds part of the IP datagram, up to 5 bytes in the first fragment and up to 7 bytes in the others. This corresponds to a maximum IP datagram size of $(2^8 - 1) \cdot 7 + 5 = 1790$ bytes, a value bigger than the Ethernet upper limit [23], that is, 1500 bytes. However, the value has been been capped to 1500 bytes to prevent IP-level fragmentation and reassembly, and hence, multiple fragmentation and reassembly points in the protocol stack when a datagram is routed from a CAN-based network to an Ethernet-based one.

In the design of CDLC, it has been assumed that hosts are able to reliably receive frames from a CAN bus working at full utilization. This assumption is reasonable given the computing power nowadays available even in very low-cost microcontrollers, like the one adopted for the implementation [36], and it simplifies protocol design. Namely, the protocol does not comprise any flow control mechanism because the CAN bus caps the maximum incoming fragment rate by itself.

Furthermore, neither end-to-end acknowledgment nor retransmission mechanisms have been included in CDLC, in order to simplify broadcast communication. A fragment loss, due to a bus error or transient overload, leads the receiving side to discard the whole IP datagram it belongs to. In this situation, some transport protocols, like TCP, will retransmit the datagram while others, like UDP, leave this duty to the application layer. From this point of view, CDLC behaves like an Ethernet-based link. Actually, the CAN bus (upon which CDLC is layered) does provide extra features with respect to Ethernet, for instance automatic fragment retransmission upon detectable bus errors, which are able to lower the probability of fragment loss. As a result, in CDLC, the frame transfer sequence becomes extremely straightforward:

- On the transmitting side, the fragments belonging to a certain IP datagram are transmitted in sequence by means of a single transmit queue, so that the CAN controller does not reorder them.
- On the receiving side, the reassembly automaton assumes that all fragments coming from the same source node shall be received in order. Any mismatch between the expected and actual fragment numbers leads the receiver to discard the current datagram and wait for the first fragment of the next one.

It must also be noted that, when considering the IP protocol, the absolute minimum datagram size that will ever be encountered is 20 bytes, that is, the minimum size of the IP header. Given the frame format shown in Fig. 3, the number of fragments needed to transfer such a datagram using IoC is 4. Therefore, it is easy to prove that the minimum number of consecutive fragment losses that leads CDLC to erroneously reassemble together fragments belonging to different datagrams is 4, too. Given the extremely low probability of this error scenario, the protocol does not include any countermeasure against it.

6.2. IoC and EoC Protocol Overhead

The overhead introduced by IoC and EoC approaches can be evaluated by considering the number of bytes n'_{CAN}, which is

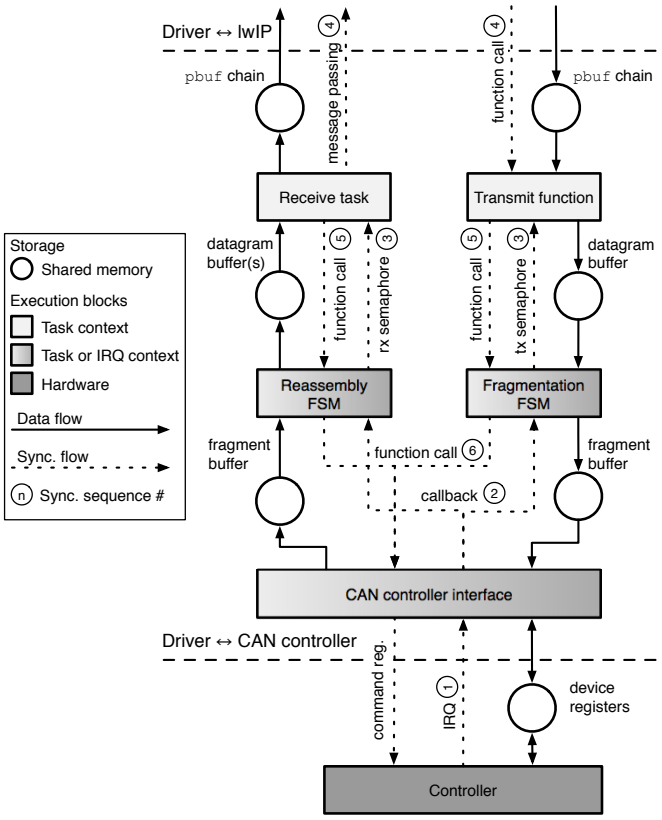

Figure 4: Internal structure of the lwIP CAN interface.

needed to transfer a TCP segment of n bytes through the CAN interface:

$$n'_{\text{CAN}} \simeq (n+h) \cdot \frac{8}{7} \cdot \frac{114}{64} = 2.04\,(n+h) \qquad (1)$$

In the equation above, h represents the total size of the protocol headers needed by TCP, IP, and possibly Ethernet (if EoC is used). More specifically, $h = 40$ for IoC (considering 20 B for the IP header plus 20 B for the TCP header). For EoC, $h = 54$ because DLPDUs transferred on the CAN bus also include a 14 B Ethernet header. The additional overhead imposed by EoC (through a higher value of h) is the same for other transport protocols and this justifies the choice of IoC for the prototype implementation, in particular CAN has a rather limited bandwidth.

In equation (1), $8/7$ represents the approximate overhead introduced by CDLC through the presence of a 1-byte fragment number for every 7 data bytes, except in the first and last fragments. Namely, the first fragment also contains a 2 B CDLC header besides the fragment number while the last fragment may contain less than 7 data bytes. It should be remarked that the approximation is still accurate because the minimum $n + h$ is 40 B (namely, for an empty TCP segment in IoC), which corresponds to 6 fragments.

Last, $114/64$ approximates the average overhead of CAN framing, CRC, IFS, and bit stuffing for random payloads [37]. The worst case can be modeled in a similar way, considering that the maximum number of bits needed to transfer a 64-bit payload is 132, as shown in [37].

As mentioned in Section 2.3, another reason to choose IoC for the prototype is that several open-source protocol stacks directly support the integration of a network interface driver for IP datagrams, and it is the only additional component required. On the contrary, the implementation of EoC would be more complex because, as discussed in Section 4, it requires the implementation of additional protocols, typical of an Ethernet bridge, for instance the Spanning Tree Protocol (STP and RSTP). This would likely increase the memory requirements, which is a resource embedded systems (in particular, low-end ones) are generally not so wealthy of.

It is worth noting that EoC is a peculiar solution aimed at providing Ethernet communication above CAN. It should be mainly used in those cases in which the network and transport layers are not based on the TCP/IP protocol suites (very few at present, indeed). In addition, taking into account the protocol overhead, complexity of implementation, as well as memory requirement, the first prototype is developed based on IoC, which is shown in the following sections together with its performance measurement.

6.3. *Integration of IoC in a TCP/IP Protocol Stack*

The internal structure of the IoC driver for the lwIP protocol stack is shown in Fig. 4. The driver ↔ lwIP interface is specified by lwIP. On the transmit side (right part of the figure), a function call, ④ in the figure, is used to request a datagram transmission. If the transmit path is idle, the transmit function immediately starts the transmission by means of function call ⑤ to the lower layer. Otherwise, synchronization between the two layers is achieved by means of semaphore primitive ③.

On the receive side (left part of the figure), a message passing interface ④ allows the driver to "push" incoming datagrams into the protocol stack upon notification that a datagram has been completely received ③. After that, a downward function call ⑤ synchronizes with the receive path and enables it to reuse the datagram buffer for reception. Since lwIP interactions must be performed from a task context, a separate *receive task* is in charge of them.

Datagrams are held in a shared `pbuf` chain (a linked list of dynamically-allocated memory buffers defined by lwIP) on each side. On the transmit side, `pbuf` chains are allocated by lwIP itself and passed to the interface driver. On the other hand, the receive task is responsible of pre-allocating `pbuf` chains and filling them with incoming IP datagrams.

The driver ↔ CAN controller interface is defined by hardware. Downward synchronization and data transfer are implemented by device register access (the CAN controller in use is not DMA-capable), while upward synchronization is provided by interrupt requests ①. A *CAN controller interface* module provides device-independent access to these hardware features.

CDLC, by itself, is implemented by means of two Finite State Machines (FSMs) shown in the middle of Fig. 4. They can be invoked by the transmit function (from a task context) and the receive task, as well as the CAN controller interface through a

callback ② (from an IRQ context, when a fragment has been received or the controller is ready to accept a new fragment for transmission). Synchronization with the transmit function and the receive task takes place by means of two semaphores ③, located at the boundary between the IRQ and task contexts. Function calls ⑥ to the CAN controller interface are used to interact and synchronize with hardware.

The preliminary implementation leaves room for further optimizations for what concerns memory management. As shown in Fig. 4, IP datagrams being transmitted are currently copied multiple times along the transmit path, and the same happens on the receive path. For this reason, the performance results described in Section 7 can likely be improved further.

For what concerns code complexity, the preliminary implementation consists of about 5500 lines of C code (of which 3000 are architecture-independent and 2500 are architecture-dependent). When evaluating the effort needed to port IoC to a new architecture, only the second value shall be considered, because the architecture-independent code must only be *recompiled*, without any modifications. The architecture-dependent code consists essentially of an appropriate device driver for the CAN controller.

7. IoC Performance Evaluation

7.1. Experimental Setup

In the prototype, IoC has been implemented on a hardware development board based on the NXP LPC1768 microcontroller [36]. The combination of the lwIP protocol stack and this microcontroller is convenient for performance comparison because it is in widespread use and its performance, at least for what concerns Ethernet-based networks, has been investigated in previous work [38].

In the experiments, two test programs set up a unidirectional data exchange, using the TCP protocol. This kind of exchange is representative of application-layer protocols in which data predominantly flows in one direction (like HTTP and FTP). Since the effect of IoC traffic on real-time performance has already been analyzed previously in Section 3.4, in the following we will just focus on the measurement of the average data transfer rate, which is a main performance index for non real-time traffic. Data transfer takes place through a single connection. The maximum number of connections the system may support has not been evaluated because it mainly depends on the available memory in lwIP and the way it is configured rather than the kind of link. In any case, all connections to the same node share the available bandwidth.

The total amount of data to be exchanged is fixed and set to 1 MB in all experiments. Data is passed to the protocol stack in chunks of d bytes each time. The value of d is configurable and it directly affects the interaction overheads with the protocol stack, that is, the number of calls needed to transfer a given total amount of data. As shown in the following, d may affect TCP segmentation indirectly, in particular the possible values of TCP segment size, denoted as n. However, this is an internal choice of the protocol stack, for example lwIP or any other protocol stack in use.

Table 2: Measured Link-Level Performance, TCP Traffic.

d (B)	1	4	16	64	256	1024
r_{CAN} (kB/s)	0.78	2.91	8.90	18.02	24.76	26.08
r_{ETH} (kB/s)	5.75	22.75	88.45	332.61	1088.03	1284.71

The first round of experiments was aimed at evaluating how effective IoC is, with respect to a standard Ethernet-based connection. To this purpose, two LPC1768-based boards, hosting the test programs, are directly connected by means of either a CAN or an Ethernet link and the average data transfer rates through them are measured.

A second set of experiments was aimed at determining communication performance when the IP *forwarding* mechanism is in use between a CAN-based and an Ethernet-based network segment. To this purpose, the same test programs were used to exchange data between an Ethernet-equipped Linux PC and an LPC1768 board implementing IoC, with the interposition of another LPC1768 board working as a router based on lwIP, as shown in Fig. 1a. This test configuration also provided the opportunity to confirm the interoperability of the router with the Linux protocol stack and assess the impact of the data-originating protocol stack on communication performance.

7.2. Link-Level Performance

For what concerns the first round of experiments, the *measured average* TCP data transfer rates through an IoC (denoted as r_{CAN}) or a standard Ethernet link (denoted as r_{ETH}), are listed in Table 2 and plotted in Fig. 5 as a function of the data chunk size d, a parameter which is specified at the application layer. To make the comparison easier, the measured average data transfer rate is plotted with reference to two calculated values, namely the *raw speed* of the physical link (which is denoted as s in the following) and the *approximate maximum* transfer rate (which is denoted as u in the following). In all experiments, the raw speed of CAN and Ethernet are $s_{\mathrm{CAN}} = 500\,\mathrm{kb/s}$ and $s_{\mathrm{ETH}} = 100\,\mathrm{Mb/s}$, which correspond to $62.5\,\mathrm{kB/s}$ and $12500\,\mathrm{kB/s}$ respectively, as shown in the figure. It is worth mentioning that, for CAN, 500 kb/s is the highest bit rate in common use. The approximate maximum transfer rate u is calculated by taking into account data-link, network, and transport-layer protocol overheads as:

$$u_{\mathrm{CAN}} = s_{\mathrm{CAN}} \cdot \frac{n}{n'_{\mathrm{CAN}}} \quad \text{and} \quad u_{\mathrm{ETH}} = s_{\mathrm{ETH}} \cdot \frac{n}{n'_{\mathrm{ETH}}} \tag{2}$$

where n'_{CAN} and n'_{ETH} represent the total number of bytes to be transmitted on the link in order to transfer a TCP segment of n bytes through IoC and Ethernet, respectively. Neglecting non-piggybacked ACK traffic in overhead calculations, n'_{CAN} can be approximated by using (1) with $h = 40$ since IoC is in use:

$$n'_{\mathrm{CAN}} \simeq (n + 40) \cdot \frac{8}{7} \cdot \frac{114}{64} = 2.04\,n + 81.43 \tag{3}$$

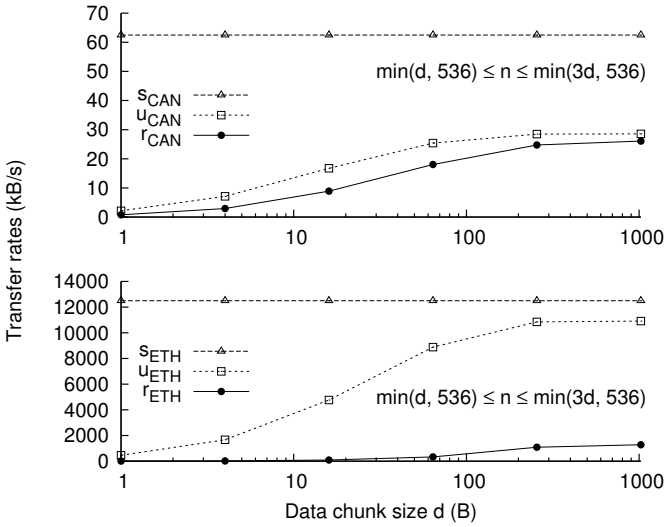

Figure 5: Link-Level Performance, IoC vs. IP over Ethernet, TCP Traffic.

Similarly, the number of bytes n'_{ETH} needed to transfer the same segment through Ethernet is:

$$n'_{\text{ETH}} = \begin{cases} 84 & n < 6 \\ n + 40 + 38 = n + 78 & n \geq 6 \end{cases} \quad (4)$$

In both cases, the overhead of 40 B is due to IP and TCP headers. Instead, in (4) the additional overhead of 38 B takes into account the combined size of the Ethernet header, FCS, Preamble, and Interframe gap [27]. Finally, (4) also considers the minimum total Ethernet frame length constraint of 84 B.

For a certain data chunk size d, the range of the TCP segment size n observed experimentally is $\min(d, 536) \leq n \leq \min(3d, 536)$. As already mentioned in Section 7.1, the variability of n mainly depends on the segmentation algorithms of the lwIP protocol stack. Regardless of d, lwIP has been configured to limit the value of n to 536 B to save memory. This upper limit (namely, 536 B) is also the minimum legal value set forth in the IP protocol specification. For a certain d, the calculation of u_{CAN} and u_{ETH} has been approximated by excess using the largest value of n observed.

The results show that the two kinds of link behave in two very different ways. On one hand, as shown in Fig. 5, the CAN-based link is able to achieve nearly fully utilization as the measured average data transfer rate r_{CAN} is quite close to the approximate maximum transfer rate u_{CAN}. Instead, this is not the case for the Ethernet-based link because software overheads prevail. On the other hand, the gap between u and s is bigger for the CAN-based link than the Ethernet-based link. This is because the network overhead plays a more important role in the former one, as can also be seen from (3) and (4). Those results are due to two different phenomena:

1. The value of d affects performance in two ways. First of all, smaller values of d correspond to smaller values of n used by lwIP. In turn, this lowers the performance according to (2)–(4). What's more, d is inversely proportional to the number of lwIP calls needed to transfer a given total amount of data. Hence, smaller values of d introduce a higher software overhead due to the inter-task communication between the test programs and the protocol stacks.

2. Across the two kinds of link, the same value of d corresponds to two very different ratios between software overheads and communication times. In fact, software overheads depend on d (and also on the CPU speed) but are independent on the link type, at least above the data-link layer. Instead, communication times mainly depend on two link-dependent properties, namely, link speed and data-link overheads (including CDLC overheads in the case of CAN).

Table 3: Forwarding Performance, TCP Traffic.

d (B)	1	4	16	64	256	1024
$r_{\text{C}\to\text{E}}$ (kB/s)	0.65	2.49	7.97	17.80	24.07	26.43
$r_{\text{E}\to\text{C}}$ (kB/s)	25.18	26.12	26.36	26.50	26.46	26.44

Their combined effect is rather counterintuitive because the speed advantage of Ethernet-based versus CAN-based communication, namely $r_{\text{ETH}}/r_{\text{CAN}}$, when carrying TCP traffic is much lower than the raw ratio between their link speeds, that is, $s_{\text{ETH}}/s_{\text{CAN}} = 200$. In fact, as can be derived from Table 2, the ratio between data transfer rates only goes from about 7.4 times (when $d = 1$) up to about 49 times (when $d = 1024$).

7.3. Forwarding Performance

The results of the second round of experiments, aimed at determining the performance of an integrated network involving the IP *forwarding* mechanism between a CAN-based and an Ethernet-based network segment, are shown in Table 3 and Fig. 6. The first row of the table pertains to the experiments in which data originate from the lwIP-based IoC end node and flow from the CAN-based to the Ethernet-based segment, while the second corresponds to data that originate from the Linux protocol stack and flow in the opposite direction. In Fig. 6, the values of s and u have been calculated with respect to CAN, which is the slower segment.

With respect to the first set of experiments, part of the TCP processing has been moved onto the PC where much higher computing power is available. For instance, when data flows from the Ethernet-based to the CAN-based segment, transmit-side TCP processing is moved onto the PC. In addition, due to the unidirectional data flow, the receive-side TCP processing still performed by lwIP is limited to passing incoming data to the application and sending acknowledgments back. Hence, overall TCP processing is mainly performed by the Linux protocol stack rather than lwIP. On the other hand, when data flows in the opposite direction, receive-side TCP processing is moved onto the PC. Even if IP forwarding is involved in both cases regarding the first set of experiments, it affects communication performance lightly since it doesn't involve the transport layer and no memory-to-memory copies are needed.

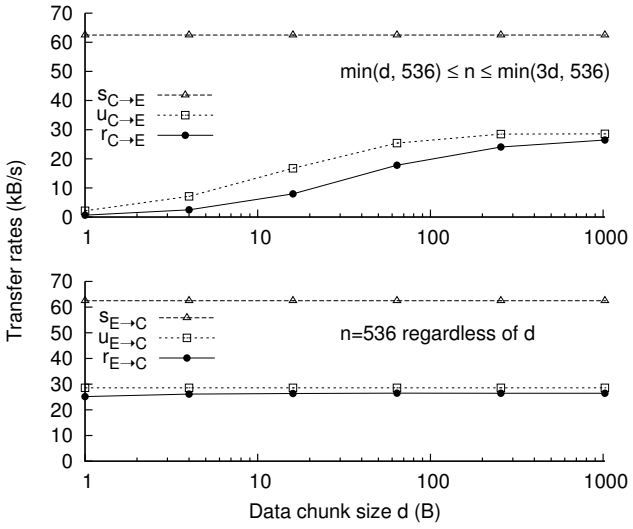

Figure 6: Forwarding Performance, TCP Traffic.

When lwIP is sending data, the dependency of $r_{C\to E}$ on d (also through n) is pronounced because it still includes both phenomena discussed in Section 7.2.

On the contrary, when the Linux protocol stack is sending data, the dependency is less pronounced as the average transfer rate $r_{E\to C}$ is always close to $u_{E\to C}$. This is because the Linux protocol stack keeps the TCP segment size constant at $n = 536$. However, as shown in Table 3, for small values of d, there is still a slight performance loss. This is because the protocol and link level overheads are now constant as n is fixed, while the dependency of inter-task communication overhead on d is still the same as before. What's more, since Linux aggressively gathers user data to build larger TCP segment, especially for small values of d, this introduces an additional source of overhead. But the last two kinds of overhead do not lead to dramatic performance loss because of the PC CPU speed advantage.

With respect to the previous set of experiments (Table 2), the data transfer rate $r_{E\to C}$ is almost always greater than the CAN-only transfer rate r_{CAN}. This is because transmit TCP processing on Linux plus lwIP forwarding is more efficient than transmit TCP processing on lwIP, taking into account that lwIP forwarding is lightweight while the PC CPU is at least one order of magnitude faster than the LPC1768.

Instead, the situation when data flows from the CAN-based segment to the Ethernet-based segment is more complex. The data transfer rate $r_{C\to E}$ is mainly bounded by TCP processing on the transmit side (still performed by lwIP) and communication overheads that depend on d through n. Hence, the general behavior of $r_{C\to E}$ is still the same as r_{CAN}. The slight speed advantage of the router configuration when $d = 1024$, and its disadvantage for smaller values of d, can be justified by the balance between the overhead incurred in receive-side TCP processing and the forwarding overhead. In fact, with respect to r_{CAN}, the receive-side TCP processing is much more efficient on the PC rather than on the LPC1768, while the forwarding overhead is inversely proportional to n and becomes higher for smaller values of d.

It is worth mentioning that the integration of the CAN-based segment and the Ethernet-based segment did not hinder the performance because the end-to-end data transfer rates $r_{C\to E}$ and $r_{E\to C}$ are still close to what a direct CAN link can provide, that is r_{CAN}.

Table 4: Memory Footprint of IoC Implementation (bytes)

Category	Text	Rodata	Data	Bss
Native/sockets API	11720	1521	0	132
Infrastructure	5832	1292	0	21
Memory management	7392	2278	0	12261
TCP protocol	19268	2088	52	53
UDP protocol	2104	102	0	0
IP protocol	7680	747	0	32
CDLC and CAN driver	3764	822	0	4
Total	57760	8850	52	12503

7.4. Memory Footprint

Besides evaluating IoC performance, as discussed in Sections 7.2 and 7.3, the experimental system presented in Section 7.1 was also used to measure the overall memory footprint of IoC starting from the linker's memory map of the executable image under test. This is useful to determine how demanding the IoC implementation is with respect to the overall memory resources available on a typical low-cost microcontroller, such as the LPC1768. In turn, this may affect the practical feasibility and convenience of the proposed approach.

Footprint data concerning an end-node IoC implementation is summarized in Table 7.4. In the table, the overall footprint is divided vertically into categories, corresponding to different protocol stack modules. Horizontally, the footprint of individual modules is broken down into *segments*, allocated in different parts of microcontroller's memory.

Namely, the *text* segment contains executable machine code and usually resides in non-volatile flash memory. The *rodata* segment, which contains immutable (i.e., read-only) data, is allocated in the same way. On the other hand, the *data* segment contains initialized read-write variables and must be allocated twice, once in non-volatile memory (that keeps their initial values) and once in RAM (that contains the working copies). Finally, the *bss* segment holds uninitialized read-write variables, which are allocated in RAM.

For what concerns categories, names are self-explanatory for the most part. The *infrastructure* category comprises all low-level protocol stack modules (for instance, the operating system adaptation layer), which are mandatory for lwIP operation but are not protocol-specific.

As shown in the table, the overall flash memory requirement does not exceed 66 KB (57760 + 8850 + 52 = 66662 B), while the total RAM is less than 13 KB (12503 + 52 = 12555 B). Considering that the microcontroller is equipped with 512 KB of flash memory and 64 KB of RAM, this correspond to less than 13% and 20% of its total memory resources, respectively.

Furthermore, the RAM footprint can easily be further reduced by tailoring the lwIP memory pools configuration to the

target application. In fact, the configuration used for the experiments is conservative. It supports and allocates memory for up to 5 simultaneous TCP connections to the IoC node, even though they may or may not be needed in practice. Other open-source protocol stacks, for instance μIP [10], feature an even smaller memory footprint, but still keep a similar internal structure. Hence, IoC can easily be adapted to work on them, too.

8. Conclusions

In this paper, a network architecture for the seamless integration of CAN in IP-based Intranets has been proposed. Rather than defining a specific solution or protocol, attention has been focused on sketching a generic framework for achieving integration. At the same time, the inherent benefits and shortcomings stemming from design choices like the communication layer at which integration takes place (Ethernet versus IP), and how to support interoperability between those choices, have been highlighted.

Experimental results derived from a prototype implementation of one of the proposed designs show that it is not only feasible, but its TCP data transfer performance on a popular class of embedded microcontrollers compares very favourably with respect to the raw ratio between CAN versus Ethernet link speeds. The preliminary implementation also consumes only a small fraction of the overall memory resources embedded in the microcontroller under study, thus confirming the practical feasibility of the proposed approach.

The prototype implementation of the other proposed design, namely EoC, is also in the schedule so as to complete the example and provide necessary performance data for reference before the proposed network architecture would be adopted in industry.

References

[1] P. Gaj, J. Jasperneite, M. Felser, Computer communication within industrial distributed environment—a survey, IEEE Trans. Ind. Informat. 9 (1) (2013) 182–189. doi:10.1109/TII.2012.2209668.

[2] D. Ingram, P. Schaub, R. Taylor, D. Campbell, Performance analysis of IEC 61850 sampled value process bus networks, IEEE Trans. Ind. Informat. 9 (3) (2013) 1445–1454. doi:10.1109/TII.2012.2228874.

[3] M. Rahmani, K. Tappayuthpijarn, B. Krebs, E. Steinbach, R. Bogenberger, Traffic shaping for resource-efficient in-vehicle communication, IEEE Trans. Ind. Informat. 5 (4) (2009) 414–428. doi:10.1109/TII.2009.2019127.

[4] J. Taube, H. Beikirch, M. Voss, Alternative powerline communications interfaces for the use in arbitrary networks, in: Proc. 5th IEEE Intl. Workshop on Factory Communication Systems (WFCS), 2004, pp. 327–330. doi:10.1109/WFCS.2004.1377738.

[5] P. Cach, P. Fiedler, Internet Draft – IP over CAN, Brno University of Technology, 2001, expires: Sept. 2001, available at http://mirror.physik-pool.tu-berlin.de/pub/ietf/ietf-tools.html/draft-cafi-can-ip-00.html.

[6] M. Ditze, R. Bernhardi, G. Kämper, P. Altenbernd, Porting the Internet Protocol to the Controller Area Network, in: Proc. 2nd Intl. Workshop on Real-Time LANs in the Internet Age, 2003, pp. 1–4.

[7] P. Lindgren, S. Aittamaa, J. Eriksson, IP over CAN, transparent vehicular to infrastructure access, in: Proc. 5th IEEE Consumer Communications and Networking Conference (CCNC), 2008, pp. 758–759. doi:10.1109/ccnc08.2007.175.

[8] M. Johanson, L. Karlsson, T. Risch, Relaying Controller Area Network frames over wireless internetworks for automotive testing applications, in: Proc. 4th Intl. Conference on Systems and Networks Communications (ICSNC), 2009, pp. 1–5. doi:10.1109/ICSNC.2009.68.

[9] IEEE Std 802.1AB™-2009, Standard for local and metropolitan area networks—Station and media access control connectivity discovery, IEEE, 2009.

[10] A. Dunkels, Full TCP/IP for 8-bit architectures, in: Proc. 1st Intl. Conference on Mobile Applications, Systems and Services (MOBISYS), 2003.

[11] T. Sauter, S. Soucek, W. Kastner, D. Dietrich, The evolution of factory and building automation, IEEE Industrial Electronics Magazine 5 (3) (2011) 35–48. doi:10.1109/MIE.2011.942175.

[12] T. Sauter, M. Lobashov, How to access factory floor information using internet technologies and gateways, IEEE Trans. Ind. Informat. 7 (4) (2011) 699–712. doi:10.1109/TII.2011.2166788.

[13] J. Beran, P. Fiedler, F. Zezulka, Virtual automation networks, IEEE Industrial Electronics Magazine 4 (3) (2010) 20–27. doi:10.1109/MIE.2010.937930.

[14] Esd electronic system design GmbH, NTCAN Part 1: Structure, Function and C/C++ API Application Developers Manual (2013).

[15] Esd electronic system design GmbH, ELLSI Manual — EtherCAN Low Level Socket Interface (2014).

[16] HMS Industrial Networks AB., User Manual — Anybus® X-gateway (2014).

[17] J.-L. Scharbarg, M. Boyer, C. Fraboul, CAN-Ethernet architectures for real-time applications, in: Proc. 10th IEEE Intl. Conference on Emerging Technologies and Factory Automation (ETFA), Vol. 2, 2005, pp. 245–252. doi:10.1109/ETFA.2005.1612687.

[18] A. Nacer, K. Jaffres-Runser, J.-L. Scharbarg, C. Fraboul, Strategies for the interconnection of CAN buses through an Ethernet switch, in: Proc. 8th IEEE Intl. Symposium on Industrial Embedded Systems (SIES), 2013, pp. 77–80. doi:10.1109/SIES.2013.6601474.

[19] X. Li, J.-L. Scharbarg, C. Fraboul, Worst-case delay analysis on a real-time heterogeneous network, in: Proc. 7th IEEE Intl. Symposium on Industrial Embedded Systems (SIES), 2012, pp. 11–20. doi:10.1109/SIES.2012.6356565.

[20] T. Pulgar, J.-L. Scharbarg, K. Jaffres-Runser, C. Fraboul, Extending CAN over the air: An interconnection study with IEEE802.11, in: Proc. 18th IEEE Intl. Conference on Emerging Technologies Factory Automation (ETFA), 2013, pp. 1–8. doi:10.1109/ETFA.2013.6648051.

[21] Robert Bosch GmbH, CAN with Flexible Data-Rate Specification Version 1.0 (2012).

[22] Z. De Zhou, R. Valerdi, S. Zhou, L. Wang, Guest editorial: Special section on IoT, IEEE Trans. Ind. Informat. 10 (2) (2014) 1413–1416. doi:10.1109/TII.2014.2316734.

[23] C. Hornig, A Standard for the Transmission of IP Datagrams over Ethernet Networks, RFC 894, Symbolics Cambridge Research Center, 1984.

[24] J. Postel (Ed.), Internet Protocol — DARPA Internet Program Protocol Specification, RFC 791, USC/Information Sciences Institute, 1981.

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

[26] IEC, Low-voltage switchgear and controlgear – Controller-device interfaces (CDIs) – Part 3: DeviceNet, 2.0 Edition, IEC 62026-3 (2008).

[27] IEEE Std 802.3™-2008, Standard for Information technology—Telecommunications and information exchange between systems—Local and metropolitan area networks—Specific requirements Part 3: Carrier sense multiple access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications, IEEE, 2008.

[28] ISO, ISO 11898-1 – Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signalling, International Organization for Standardization (2003).

[29] ISO, ISO 11898-2 – Road vehicles – Controller area network (CAN) – Part 2: High-speed medium access unit, International Organization for Standardization (2003).

[30] ISO, ISO 11898-4 – Road vehicles – Controller area network (CAN) – Part 4: Time-triggered communication, International Organization for Standardization (2004).

[31] R. Davis, A. Burns, R. Bril, J. Lukkien, Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised, Real-Time Systems 35 (3) (2007) 239–272.

[32] R. I. Davis, S. Kollmann, V. Pollex, F. Slomka, Schedulability analysis

for Controller Area Network (CAN) with FIFO queues priority queues and gateways, Real-Time Systems 49 (1) (2013) 73–116. doi:10.1007/s11241-012-9167-8.

[33] Y. Rekhter, T. Li (Eds.), An Architecture for IP Address Allocation with CIDR, RFC 1518, 1993.

[34] R. Braden (Ed.), Requirements for Internet Hosts – Communication Layers, RFC 1122, Internet Engineering Task Force, 1989.

[35] M. Oh, Y.-G. Kim, S. Hong, S. Cha, ASA: Agent-based secure ARP cache management, IET Communications 6 (7) (2012) 685–693. doi:10.1049/iet-com.2011.0566.

[36] NXP B.V., LPC17XX User manual, UM10360 rev. 2 (Aug. 2010).

[37] G. Cena, I. Cibrario Bertolotti, T. Hu, A. Valenzano, Performance comparison of mechanisms to reduce bit stuffing jitters in Controller Area Networks, in: Proc. 17th IEEE Intl. Conference on Emerging Technologies and Factory Automation (ETFA), 2012, pp. 1–8. doi:10.1109/ETFA.2012.6489559.

[38] I. Cibrario Bertolotti, T. Hu, Real-time performance of an open-source protocol stack for low-cost, embedded systems, in: Proc. 16th IEEE Intl. Conference on Emerging Technologies and Factory Automation (ETFA), 2011, pp. 1–8.

Gianluca Cena has been Director of Research with the Istituto di Elettronica e di Ingegneria dell'Informazione e delle Telecomunicazioni (IEIIT) of the National Research Council of Italy (CNR) since 2005. His current research interests include industrial communication systems, wired and wireless networks, and real-time protocols.

Gianluca Cena served as Program Co-Chairman for the 2006 and 2008 editions of the IEEE Workshop on Factory Communication Systems, and took part to the organization of several other IEEE Conferences. He is Senior Member of IEEE and has been Associate Editor of the IEEE Transactions on Industrial Informatics since 2009.

Ivan Cibrario Bertolotti has been a Researcher with the National Research Council of Italy (CNR) since 1996. Currently, he is with the Istituto di Elettronica e di Ingegneria dell'Informazione e delle Telecomunicazioni (IEIIT), Torino, Italy. He co-authored a book on real-time, embedded operating systems and regularly serves as a technical referee for several primary international conferences and journals. Ivan Cibrario Bertolotti has been an IEEE member since 2006. His current research interests include real-time operating system design and implementation, industrial communication systems and protocols, and formal methods for vulnerability and dependability analysis of distributed systems.

Tingting Hu has been a Research Fellow with the National Research Council of Italy (CNR) since 2010. Currently, she is with the Istituto di Elettronica e di Ingegneria dell'Informazione e delle Telecomunicazioni (IEIIT), Torino, Italy, and she also holds a PhD degree in Control and Computer Engineering from Politecnico di Torino, Torino, Italy. Her primary research interests concern the design and implementation of real-time operating systems and communication protocols. Tingting Hu is an IEEE member since 2011 and serves as technical referee for several primary conferences in her research area.

Adriano Valenzano is Director of Research with the National Research Council of Italy (CNR). He is currently with Istituto di Elettronica e di Ingegneria dell'Informazione e delle Telecomunicazioni (IEIIT), Torino, Italy, where he is responsible for researches concerning industrial computer networks and systems. He has co-authored about 200 refereed journal and conference papers in the area of computer engineering. Since 2007 he has been serving as an Associate Editor for the IEEE Transactions on Industrial Informatics. Adriano Valenzano is a Senior Member of the IEEE. He is also vice-president of the Piedmont chapter of the Italian National Association for Automation (ANIPLA).