FPT2013 Demo Paper

Enhancing Communication On Automotive Networks Using Data Layer Extensions

Shanker Shreejith, Suhaib A. Fahmy
School of Computer Engineering
Nanyang Technological University, Singapore
Email: {shreejit1,sfahmy}@ntu.edu.sg

***Abstract*—Automotive systems function within a distributed computing paradigm consisting of networks of sensors, actuators and processing units. More advanced functions are finding their way into the automotive domain, making computing and networking more complex. While safety, security and determinism are primary concerns for many systems, the networking protocols do not provide extensions to address them and it is left to application designers to tackle these issues. Standardising simple features like time-stamping of messages and health status flags can help improve robustness and mitigate risks associated with replay attacks. It is also possible to integrate further protection, like encryption of messages, addressing the increasing security concerns in this domain. In this paper, we demonstrate a systematic way of accommodating such enhancements within a standard automotive network that retains interoperability with existing systems. We show how such enhancements can be made possible in both software and hardware to help add functionality above the core network specification. We also show that the enhancements incorporated at the hardware layer offers 20× better performance than the software-based approach.**

I. Introduction

Modern vehicles are controlled by distributed embedded computing modules, which implement algorithms for the different functions like drive-by-wire, auto park and others. The different modules perform specific functions/transformations on input data (from sensors or previous modules) and generate outputs which are used by subsequent modules or fed to control hardware (actuators). Data and control exchange occurs over a shared twisted pair medium, over which communication is controlled by protocols like Control Area Networks (CAN) [1], FlexRay [2], and others. Each embedded control unit (ECU) thus incorporates a network interface(s), which implements the protocol specification enabling communication with other devices, in addition to the computing function/logic. Because of the distributed nature of the whole system, enhancements to overall network protocols and features supported by individual ECUs can affect overall system performance significantly.

Any enhancement to the defined communication protocol must be contained within the data segment to ensure interoperability with existing systems. Protocols such as FlexRay and CAN leave the data segment completely to the application designer, and hence enhancements in the data layer are effectively hidden from the protocol, ensuring compatibility with standard network controllers. However, this then requires that these extensions be processed within the software or hardware that manages the ECU computation. This can interfere with the performance of the ECU if this processing burden is significant.

Common off-the-shelf controllers for FlexRay and CAN does not provide add-on features or enhancements like time-stamping, so if required, they must be implemented in ECU software. However, time-stamping messages when they are passed to the network interface is not ideal since these messages may be transmitted over the network at a later time than when the time-stamps were generated. Moreover, because the ECU execution is not synchronised in a traditional system, the software approach incurs additional complexity by requiring that the different computing modules also be synchronised (in addition to the network layer). Adding this functionality would also complicate task management within the ECU and consume higher power, alongside the actual application.

Instead, it is possible to modify the network interface itself to filter and process these extensions. This approach presents a more efficient solution, since time-triggered networks like FlexRay require the network interfaces to be synchronised to a high degree before communication can be established. Furthermore, data segment enhancements like time-stamps and data-segment headers can be easily and accurately handled at the interface, resulting in a better in-vehicle ecosystem. In our implementation of the extensions, we mimic a data-layer protocol structure similar to the existing layered model followed in popular networks like Ethernet, providing larger possibilities for expansion.

In this demonstration, we showcase specific and generic applications of data-layer extensions that can be used to introduce additional features which would otherwise be expensive. The demonstration shows that a synchronised time-stamping of messages can protect the system from replay attacks, which is a commonly exploited security flaw in automotive networks. We also show that by incorporating the *health state* of the computing node into the data-headers of existing communications, we can minimise fault-detection latency and turnaround times for fault-tolerant systems.

II. Background and Concept

Each application can be thought of as a combination of interdependent tasks. For instance, in a brake-by-wire application, the different tasks could be sensor data acquisition, sensor data fusion, sensor command validation and actuator command

978-1-4799-2198-0/13/$31.00 ©2013 IEEE

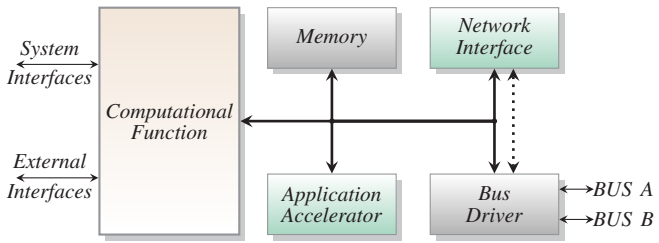

Fig. 1: Typical ECU model.

generation, along with the independent communication tasks to send and receive data. Each ECU comprises a computational core (single or multicore processor), associated resources, and the network interface, as shown in Fig. 1. Tasks are efficiently mapped onto the distributed computing resources (ECUs and networks) to implement the complete application.

FlexRay and CAN are robust serial communication protocols based on time-triggered and event-triggered access mechanisms, respectively. FlexRay uses a concept of slots which provide access to the different ECUs based on a predefined schedule, while CAN uses a priority-based access scheme to the network. In this work, we concentrate on FlexRay-based ECUs, while the principles we demonstrate are equally applicable to CAN and other networks.

The FlexRay protocol defines the FlexRay cycle, which repeats with a configurable periodicity. Each cycle is in turn a repetition of slots of fixed (static) and variable (dynamic) width, which are statically assigned to the participating ECUs by the communication schedule. These different slot mechanisms enable FlexRay to cater to deterministic and volume data transfer over the same bus. In an assigned slot, the ECU transfers data by encapsulating it in a FlexRay frame, as shown in Fig. 2. The complete protocol is usually implemented as a companion chip like the Infineon CIC 310 [3], Freescale S12XF, or integrated with the processing core as slave IP on the same die.

The FlexRay header in the frame captures information specific to the protocol like slot number, cycle number and flags. The application/user has complete control over the data segment, and this is transparent to other devices on the network. However, like CAN and other existing in-vehicle networks, FlexRay messages do not have a notion of time attached to them, which would otherwise enable the application to distinguish between current active data and stale data. The lack of time-tagging is a highly exploited security flaw in in-vehicle systems, whereby an external agent which gains access to the internal network can monitor transactions and replay messages at a later point to disrupt the system [4]. Similarly, there is no standardised scheme for communicating fault status or health state for safety critical systems, without dedicating extra slots for such communication. With the rising number of safety-critical functions and increased automation in modern vehicle, such as with drive-by-wire systems, we believe that a standard approach to incorporate such critical

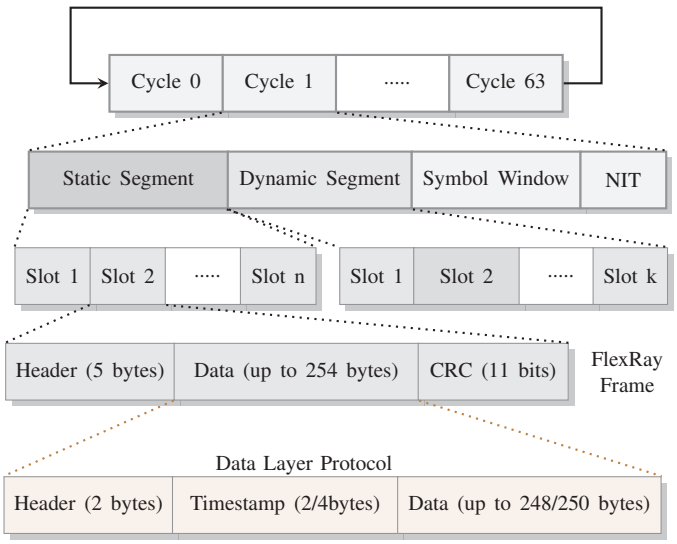

Fig. 2: FlexRay communication model.

information would enhance overall quality of service, without consuming additional communication bandwidth.

For this demonstration, we incorporate timing and health state into the data segment and explore the applications of such enhancements. This can be thought of as an additional *data layer protocol* (as shown in Fig. 2) similar to the Ethernet specification, comprising a data header and message timestamp followed by the actual application data.The header incorporates information such as sequence number (for multi-cycle data), destination identifiers and health state (flags and consecutive errors). The length of individual elements and granularity of timestamps are configurable, but must have the same value/meaning at all participating nodes.Moreover, such a scheme can co-exist with standard off-the-shelf devices on the same network, since these enhancements are hidden from them. In [5], we show a potential application of such extensions by coupling them with partial reconfigurability of FPGAs for advanced safety critical systems with self-healing capabilities and minimised recovery times. In [6], [7], similar extensions are used for functional validation of the network.

Such features can be incorporated into the network interface or can be handled at the computational logic in software. Off-the-shelf controllers and integrated devices use either the e-Ray IP model from Bosch [8] or the FRCC2100 from Freescale [9], which provide limited scope for incorporating such features at the interface, making a software-based approach the only possible solution for many existing designs. For our demonstration, we have also integrated these features into the datapath of a custom FlexRay controller.With FPGA's finding their way into automotive applications [10], the hardware approach allows us to explore advantages of integrating such extensions within the network interface.

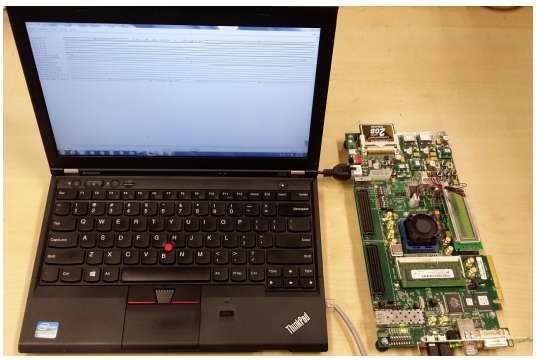

Fig. 3: Demonstration setup with ML605 development board.

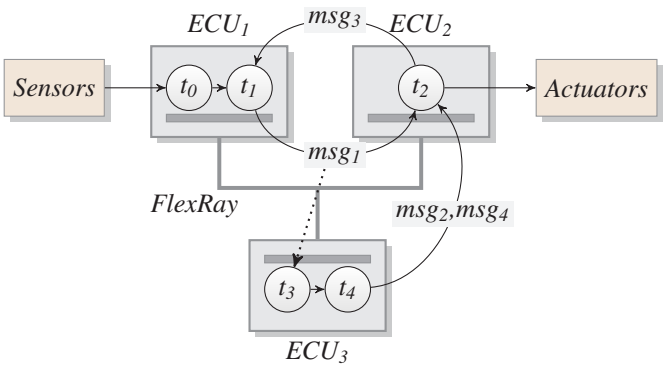

Fig. 4: Demonstration model.

III. Demonstration Overview

The goal of the demonstration platform is to incorporate data segment enhancements to the in-vehicle communication framework and explore applications of this approach. The platform comprises a distributed ECU system on a Xilinx ML605 development board, which uses serial and Ethernet interfaces to a standard desktop/laptop computer (host machine) shown in Fig. 3 (see [7] for further details).

We use a three ECU system to model a simple brake-by-wire application, which uses the on-board I/O modules (switches and LEDs) as the sensor and actuator interfaces. Microblaze processors form the ECU units, with applications (tasks) loaded as software. The individual ECUs are networked using the FlexRay protocol. Resource utilisation on the device is shown in Table I. The host machine presents a graphical interface (GUI) to the user, providing a means to visualise vehicle speed, perform ECU configuration and display updates/status from the ECUs.

A simplified diagram of the ECU communication is shown in Fig. 4. Each ECU performs multiple tasks, indicated by the different t_x and exchanges messages msg_y over the network.

- ECU_1 is the sensor ECU. It reads the sensor data and generates sensor commands (msg_1).
- ECU_2 is the actuator ECU. It receives sensor commands and health messages (msg_2) from ECU_3. If health status is normal, it acknowledges sensor commands (msg_3) and produces actuator commands; else it flags a fault state via (msg_3).
- ECU_3 monitors transactions and generates health status for ECU_1. It also generates replay data (msg_4) when enabled.

In the demonstration, we look at applications of message timestamps and health states, from the context of the safety-critical brake-by-wire system. The different steps/states in the demonstration are shown in Fig. 5 and are discussed below.

A. Platform Setup and Initialisation

The bitstream file is pre-generated and programmed on the flash memory of the ML605 board. Following power-on, individual ECUs can be chosen and programmed with elf files from the graphical interface. When programmed, the ECUs synchronise over the internal network, and the status is indicated on the graphical interface.

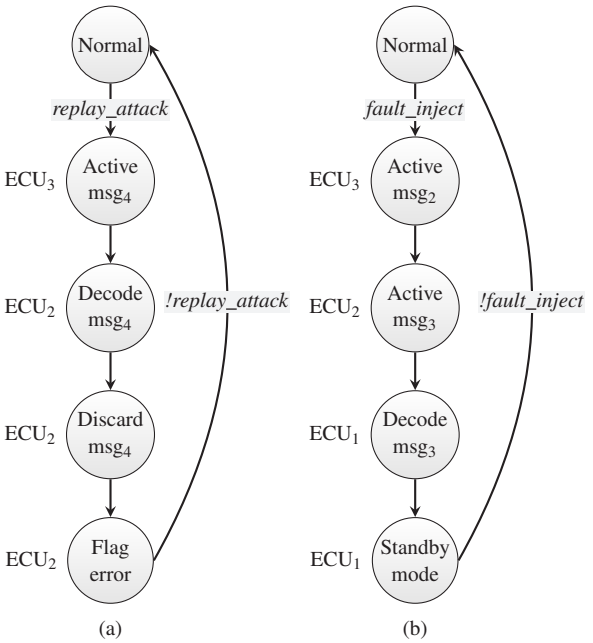

Fig. 5: Demonstration Steps (a) Replay Attack, (b) Fault injection.

B. Timing awareness and Replay Attacks

A replay attack is a simple and effective technique to adversely affect a network. Here, the attacker observes and captures communication on the network and at a later time replays this data on the communication medium. This attack is effective, primarily because the messages on the network do not have a notion of time, and hence the receiver cannot differentiate between live data and replayed data. The *Enable Replay Attack* button generates a replay attack on the system and the following can be observed on the GUI:

- ECU_3 generates valid msg_4, with unsynchronised timestamps.
- ECU_2 rejects msg_4, flags invalid timestamps.
- No change in vehicle speed.

- When *Enable Replay Attack* is disabled, active msg_4 are disabled.

C. Health State and Safety Critical ECUs

Safety critical functions in vehicles incorporate a fall-back mode of operation, which ensures minimal operating functionality in presence of errors. Here, we utilise health flags on msg_3 to indicate a fault in ECU_1. We observe the transition of ECU_1 to fall-back mode, which is the minimal operating state. The *Enable Fault Injection* button triggers a fault on the sensor ECU and the following can be observed on the GUI:

- ECU_3 generates fault msg_2 corresponding to sensor messages.
- ECU_2 flags faulty health state in msg_3.
- ECU_1 switches to fall-back mode, flags less precise operative mode.
- When *Enable Fault Injection* disabled, active msg_2 is disabled.
- ECU_2 flags normal state in msg_3.
- ECU_1 switches back to normal mode, flags normal operative mode.

D. Comparison of Hardware and Software Approaches

In the default setup, such enhancements are added into the software that interfaces with the communication tasks. The same functionality can be enabled in the hardware by selecting *hardware mode* button from the graphical interface, which causes our custom extensions in the FlexRay hardware to be enabled. In this mode, we can observe faster and more deterministic response rates for the above test instances, compared to the software based approach.

From the timing awareness case, the timestamps on GUI show that ECU_2 msg_4 rejection is faster and more deterministic than the software approach. For the fault tolerant scheme, ECU_1 decodes fault flags in msg_3 faster, resulting in reliable turnaround times. Table II shows the response times for hardware and software-based approaches for the different demonstration cases.

The key idea here is that, the software-based approach requires the data to be transferred through the network controller, into the processor, before being processed in software. For the hardware-based approach, the extensions on the network controller process this information as the messages arrive, initiating an interrupt when necessary, hence resulting in much smaller latencies and better determinism. Such a low latency approach is important in designing high performance network units like gateways and FlexRay switches [11] (between multiple sub-networks) that can process messages and make decisions on the fly as opposed to a store and forward scheme.

TABLE I: Resource utilisation on XC6VLX240T.

Function	LUTs	FFs	BRAMs	DSPs
ECU_1	10638	6984	48	5
ECU_2	10601	6890	48	5
ECU_3	10618	6886	32	5
Debug Logic	11629	8150	39	5
Total	43486	28820	167	20
(%)	28.8%	9.6%	40%	2%

TABLE II: Software and hardware performance comparison.

Enhancement	Processing	Software	Hardware
Timing awareness	Timestamp accuracy (Tx)	64 us (2 slots min)	100 ns
	Timestamp processing	3780 ns	180 ns
Health monitoring	Msg processing	3550 ns	180 ns
	Turnaround (standby)	3650 ns	280 ns

IV. Conclusion

In this paper, we demonstrated how in-vehicle communication networks like FlexRay can be extended by using extensions in a data-layer above the protocol layer. The demonstration highlights the use of such enhancements in different scenarios like health state communication and replay attack protection, without requiring additional communication slots/bandwidth. We also show that such data-layer protocols offer faster and much more deterministic performance when integrated into the network controller as opposed to integration in software within the ECUs. We believe that this approach can be adapted for other legacy and evolving in-vehicles networks standards like CAN and time-triggered Ethernet, and will be an important tool in improving safety, reliability and determinism of in-vehicle communication.

References

[1] *CAN Specification, Version 2.0*, R. Bosch GmBh, Std., 1991.

[2] *FlexRay Communications System, Protocol Specification Version 2.1 Revision A*, FlexRay Consortium Std., December 2005.

[3] *SAK-CIC310-OSMX2HT, FlexRay Communication Controller Data Sheet*, Infineon Technologies AG, June 2007.

[4] I. Rouf, R. Miller, H. Mustafa, T. Taylor, S. Oh, W. Xu, M. Gruteser, W. Trappe, and I. Seskar, "Security and privacy vulnerabilities of in-car wireless networks: a tire pressure monitoring system case study," in *Proc. of the USENIX conference on Security*, 2010.

[5] S. Shreejith, K. Vipin, S. A. Fahmy, and M. Lukasiewycz, "An Approach for Redundancy in FlexRay Networks Using FPGA Partial Reconfiguration," in *Proc. of the Design, Automation and Test in Europe (DATE) Conference*, 2013, p. 721 to 724.

[6] J. Sobotka and J. Novak, "FlexRay controller with special testing capabilities," in *in Proc. of the Conference on Applied Electronics (AE)*, 2012, p. 269 to 272.

[7] S. Shreejith, S. A. Fahmy, and M. Lukasiewycz, "Accelerating Validation of Time-Triggered Automotive Systems on FPGAs," in *Proc. of the International Conference on Field Programmable Technology (FPT)*, 2013.

[8] *Product Information : E-Ray IP Module*, Robert Bosch GmbH, July 2009.

[9] *FRCC2100 : Product Brochure, Freescale FlexRay Communications Controller Core*, IPextreme, Inc.

[10] S. Shreejith, S. Fahmy, and M. Lukasiewycz, "Reconfigurable Computing in Next-Generation Automotive Networks," *Embedded Systems Letters, IEEE*, vol. 5, no. 1, p. 12 to 15, 2013.

[11] P. Milbredt, B. Vermeulen, G. Tabanoglu, and M. Lukasiewycz, "Switched FlexRay: Increasing the Effective Bandwidth and Safety of FlexRay Networks," in *Proc. of the Conference on Emerging Technologies and Factory Automation (ETFA)*, 2010.