Virtual Switch Supporting Time-Space Partitioning and Dynamic Configuration for Integrated Train Control and Management Systems

Hongjie Fang and Roman Obermaisser

University of Siegen, Germany

Abstract— **In the railway domain, an execution environment supporting data flows of mixed-criticality applications is an open research problem towards higher integration of future Train Control and Monitoring Systems (TCMS). Compared to the avionic and automotive domains, train inauguration deals with changes of train composition and configuration and is a specific requirement of the railway domain. The statically configured communication channels of other domains need to be extended, in order to cover the requirement of dynamic configuration for train inauguration.**

In this paper, we propose a virtual switch that ensures temporal and spatial partitioning between data flows of the TCMS applications hosted on the same computing node. The switch leverages the Software-Defined Networking (SDN) paradigm to be reconfigurable in order to cover the train inauguration requirement.

I. INTRODUCTION

Extensive research on integrating multiple functions with different criticality levels on a shared platform was carried out in avionic and automotive domains. Corresponding industry standards like ARINC 653 and AUTOSAR were released. In ARINC 653 [16], the functions of class A and class B (according to DO-178C [5]) can be integrated based on the APlication/EXecutive (APEX) interface without interfering with each other. The basic protection mechanism for the integration is temporal and spatial isolation which is designed to prevent faults propagating between different partitions. The safety critical partitions are guaranteed to have reserved computing resources to avoid missing the deadlines.

Due to the specific industry constraints and the long development life-cycles, railway transportation systems have suffered from a limited adoption of novel technological advancements in electronic hardware and software, communication networks and embedded computing. The existing TCMS execution environments are based on a federated architecture which is different to an integrated architecture as used in ARINC 653. Every computing node in a federated architecture hosts one specific function and the communication between computing nodes is based on the Train Communication Network (TCN). According to IEC 61375 [9], the TCN defines a two level network architecture with the Train Backbone (TB) and the Consist Network (CN).

Train inauguration [11] represents the dynamic aspects of the railway domain. Vehicles in the railway domain are free to couple and decouple with each other, therefore causing the need for the train inauguration process. In the existing TCMS, train inauguration process affects only the data communication at TB level. The resulting need for adaptation is resolved by the Train Topology Discovery Protocol (TTDP) [11], which can dynamically manage the train composition related information in the Train Topology Database (TTDB) [17]. The distributed train applications are addressed based on the functionalities using TCN Uniform Resource Identifiers (TCN URIs) defined in IEC 61375 [17]. With the TCN DNS server resolving the TCN URIs to IP addresses based on the topology related information in TTDB, the existing TCMS can resolve the inconsistency caused by train inauguration.

In contrast, the data communication defined in ARINC 653 is statically configured through channels between communication ports. When the fundamental architecture of the TCMS execution environment evolves from a federated architecture to an integrated one, the train inauguration is a major challenge.

For the network architecture and design, SDN has caused significant interest and causes the network designers to rethink the classical data forwarding approach [8]. In the design of SDN, the network forwarding control plane is separated from the data forwarding hardware, which enables the control logic and state to be reconfigured during runtime. On a global network view, the network control plane can be treated as a controlling application that makes the whole network logically centralized [8]. One of the most famous SDN implementations is the Open vSwitch in cloud computing environments [10]. In the cloud computing field, the computing resources are virtualized in order to host multiple virtual devices [10] and Open vSwitch was designed to manage the data communication between virtual machines in a layered way (i.e. control plane and data plane are separated on different layers).

Due to the benefits of SDN, we leverage the SDN paradigm to propose our data communication architecture as a virtual switch residing in the TCMS execution environment based on an integrated architecture, in order to solve the dynamic data flow controlling requirement caused by train inauguration. we extend the temporal and spatial partitioning concept from the application layer to the data switching layer to guarantee the absence of interference between different data flows.

The remainder of the paper is structured as follows. We analyse the state-of-the-art (SOTA) and related work in section II. In section III, we discuss the major requirements for the data communication in the TCMS execution environment based on an integrated architecture. We propose our design

and describe our implementation results in sections IV and V. Section VI discusses the conclusion and future work.

II. Related Work

In this section, we analyse the related work regarding the railway domain. In addition, we discuss the development of SDN technology in fields like cloud computing and virtual switching. Since the major objective of our virtual switch is to solve the dynamic data flow controlling problem caused by train inauguration, we concentrate in this section on the related work of the reconfiguration aspect.

A. ARINC 653 and ARINC 664

ARINC 653 and ARINC 664 [1] were introduced in the avionic domain to define the Integrated Modular Avionics architecture (IMA) for sharing the computing and communication resources of electronic platforms. One of the ultimate objectives of introducing the IMA platforms is to define a reconfigurable system [6]. In the SCARLETT project, Bieber et al. [2] proposed the preliminary design of a reconfigurable IMA platform, which targets to define the reconfiguration scenarios and the corresponding processes to switch between pre-defined configurations under the identified safety constraints [3]. To the best of our knowledge, the research on the reconfiguration of IMA platforms (e.g., partition scheduling, communication scheduling, etc.) is based on switching between pre-defined scheduling plans.

B. TCMS execution environment

In today's execution environments in the railway domain, a TCN is different from an avionic network like ARINC 664 due to the free coupling of vehicles which compose the whole train [4]. From the view point of the whole train, inauguration only affects the TB network, which is composed of CNs connected by Train Backbone Nodes (TBN). All TBNs run the TTDP for the inauguration process and store the train topology related information in the TTDB.

A distributed train application consist of a function leader, function followers and function devices [18] which can be distributed in different consists. In order to address the location transparency of function components, functional addressing was proposed in IEC 61375-2-3 [17] for the inter-consist communication. In more details, a function component is assigned one TCN URI and the resolving from a TCN URI to an IP address is achieved by the TCN DNS server residing within the consist. The TCN DNS server is able to query the TTDB for the up-to-date train topology information and compute the corresponding IP addresses.

C. SDN

SDN gained much attention in both research and industry communities, since SDN brought a revolutionary networking paradigm based on the existing layered programmable network definitions [13]. SDN evolves from the programmable networks and control/data plane separation paradigms, and SDN progressed quickly and successfully thanks to the success of OpenFlow and network operating systems [13]. The major difference between SDN and the classic network infrastructure is that the forwarding tables within the data plane are centrally configurable by the control plane.

Open vSwitch has been one of the most famous OpenFlow switches and it has also been adopted in the cloud computing field. In [15], Mian et al. validated Open vSwitch to be secure at the cost of increased round trip time compared to using non-virtualized execution environments for cloud computing. He and Liang [7] evaluated the security, Quality of Service (QoS) and network performance, which showed positive results. In order to guarantee QoS, Akella and Xiong [12] proposed an approach to allocate bandwidth for satisfying QoS requirements of the priority cloud users based on Open vSwitch. In this research, the QoS approach uses a bandwidth and path length based metric and queuing techniques for different users. However, for the safety critical data communication in mixed-criticality systems (e.g., TCMS execution environment), the bounded latency between communication entities is an important requirement to be addressed [14].

III. Requirements for Virtual switch in TCMS execution environment

In this section, we will discuss the major requirements for the virtual switch in a TCMS execution environment based on an integrated architecture.

A. Temporal and Spatial Partitioning

The proposed virtual switch extends the temporal and spatial isolation concept [16] from the partition level to the virtual switch level, in order to guarantee the absence of interference between different data flows. Different partitions connecting to the virtual switch within one computing node should be partitioned with respect to the execution time, in the way that the partitions are scheduled on a fixed and cyclic basis, so that the non-safety critical applications are not able to affect computing resources of the safety critical applications. The data transportation for each ingress port within the virtual switch should also be temporally separated. With respect to spatial isolation, the ports of the virtual switch belonging to different partitions should have their own dedicated memory resources, in order to prevent fault propagation between data flows.

B. Reconfiguration

In the railway domain, vehicles are required to dynamically couple at the data network level. In this case, configurations of different computing nodes are not fixed, which means it is unrealistic to pre-define data communication configurations. The proposed virtual switch is capable to adapt to the changes of the train topology during runtime, leveraging the existing mechanisms implemented in the railway domain (e.g., TTDB, TCN URI).

IV. System architecture

In this section, we propose our architecture design to address the identified partitioning and reconfiguration requirements. Our design is based on the PikeOS Real Time Operating System (RTOS) [19].

A. General architecture

In the example system architecture in Fig.1, we have 5 resource partitions, which define the spatial isolation between each other, running on top of the PikeOS kernel. The processes within partition 1 to 4 are communicating with each other through the Communication Driver (COM DRV), which encapsulates the whole message sending and receiving process and provide well defined interfaces to the applications. The COM DRV exchanges messages with the DNS server and the Switch Data Plane (SW Data Plane) implemented as PikeOS Kernel Level Drivers [20] in the kernel via statically configured ports. The defined ports are of different functionalities. For the SW Data Plane and the DNS Server, ports of No. 1 to 8 are defined as the message exchanging ports, while the ports named with "CTR" are supposed to be the configuration entry points of these kernel components. Connections between ports are colored differently to identify different data streams and partitions.

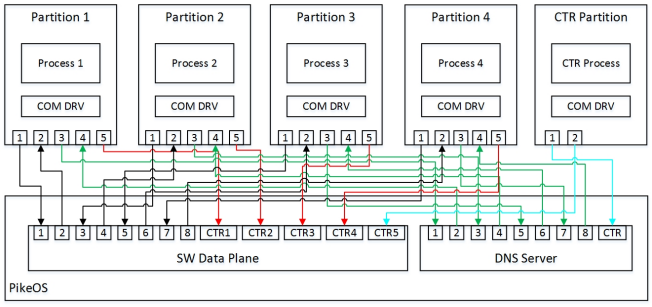

Fig. 1. General Architecture

B. Resource partition and time partition

The resource partition and time partition concepts are introduced in PikeOS [21] in order to guarantee non interference with respect to system resources between different applications. A resource partition represents a TCMS application and is assigned to a time partition that owns statically allocated time windows within the fixed cyclic time frame. The PikeOS scheduler dispatches the time partitions in a time-driven manner to meet the timing requirements of real-time applications. The priority based preemptive schedule applies to the threads within resource partitions assigned to the same time partition.

C. Data plane and control plane

According to the SDN paradigm, we define the SW Data Plane as the data plane of the virtual switch and the CTR Partition as the control component. The DNS server in Fig.1 is a mock up instance and extended to map a TCN URI to the destination ports. Before sending a message, the sending partition will query the DNS server with the destination TCN URI for the destination ports within the SW Data Plane. Thereafter it will configure the path within the SW Data Plane, and then send out the messages. The SW Data Plane is capable for buffering the received messages in a dedicated buffer of each ingress port (e.g., port 1, 3. etc.), which ensures the spatial separation between data flows.

The message transport within the data plane is triggered by the CTR partition during its assigned time windows. We avoid moving the messages to the destination ports right after receiving in the ingress ports, so that the destination ports are not exposed to a data race situation, in which a non-safety critical message can delay a safety critical one. The order of ports, from which the SW Data Plane moves the messages to the destination ports, can be configured by the CTR partition.

V. Implementation and Results

In this section, we discuss the implementation and analyse the experimental results of the prototype. For supporting hard real time communication, all the related components should have bounded Worst Case Execution Times (WCET). We discuss the relevant data structure and the scheduling of our implementation, in order to show the deterministic behavior of the proposed virtual switch.

A. Relevant data structures

1) SW Data Plane: The major data structure in the SW Data Plane consists of three parts.

SW Data Structure

```
unsigned char port_bitmap[MAX_NUM_PORT]
BUFFER_OF_PORT buffer_array[MAX_NUM_PORT]
unsigned char port_control_list[MAX_NUM_PORT]
```

BUFFER_OF_PORT

```
int num_of_data_in_buffer
char port_buffer[BUFF_SIZE]
```

Fig. 2. The Major Data Structures in SW Data Plane

As shown in Fig.2, the first part is the array port_bitmap, which is used for each ingress port to store the bitmap of the destination ports. A partition needs to configure this bitmap before it sends out a message. The second part is the buffer_array for each ingress port to buffer the received messages before the SW Data Plane moves them to the destination ports. When the SW Data Plane is triggered to transport the messages from the ingress ports to the egress ports, the bitmap port_control_list is used to define the priorities of the ingress ports.

2) DNS Server: Fig.3 shows the major data structures used in the DNS server.

The DNS Server stores the mapping of TCN URIs, IP addresses and port cookies in the uri_ip_port_mapping_array to resolve the queries from the sending partitions. The port_buffer array is defined for each port to record the query results. Every port within the DNS Server and the SW Data Plane is assigned a unique cookie. The query results can contain more than one destination port cookie, in case one partition needs to send out a multi-cast message.

```
DNS Server Data Structure
PORT_BUFFER_STRUCTURE port_buffer[NUM_of_PARTITION]
URI_IP_PORT uri_ip_port_mapping_array[NUM_of_PARTITION]

PORT_BUFFER_STRUCTURE
int NUM_of_IP
char DES_URI[LENGTH_of_URI]
vm_sockaddr_ip4_t IP_Address[NUM_of_PARTITION]
unsigned long long port_cookie[NUM_of_PARTITION]

URI_IP_PORT
char URI[LENGTH_of_URI]
vm_sockaddr_ip4_t IP_Address
unsigned long long port_cookie
```

Fig. 3. The Major Data Structure in DNS Server

B. Scheduling

In the general architecture in Fig.1, we have 5 resource partitions. In the implementation, Partition 1 and Partition 2 are the senders and correspondingly, Partition 3 and Partition 4 are the receivers.

The scheduling of the implementation is depicted in Fig.4. We define a cyclic time frame of 40ms and each of the time windows is set to 10ms, so that despite of system overhead, each sending or receiving process can be finished within one time window.

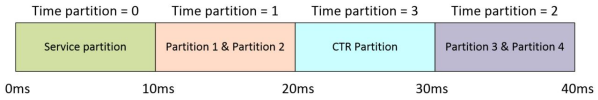

Fig. 4. Scheduling Scheme

C. Reconfiguration

In the example system architecture, we assume that the train topology related information is updated periodically and the DNS Server keeps the up-to-date mapping of the TCN URIs, IP addresses and the assigned ports within the SW Data Plane. The reconfiguration of the SW Data Plane is done by each COM DRV before sending out a message. In more detail, if one process requests to send out a message with a destination TCN URI, the COM DRV will query the DNS Server for the destination port of this message, and then configure the SW Data Plane through the connected CTR port.

D. System setup

In our implementation, we use the QEMU emulator to emulate a generic X86 platform for the generated ROM image, which integrates the PikeOS kernel, kernel drivers, system extensions and the user applications. Our emulation runs on the PC hardware with two cores of 2.3 GHz and 24 GB memory.

E. Proof-of-concept results

In this implementation, the process of conveying one message from one partition to another one is depicted in Fig.5.

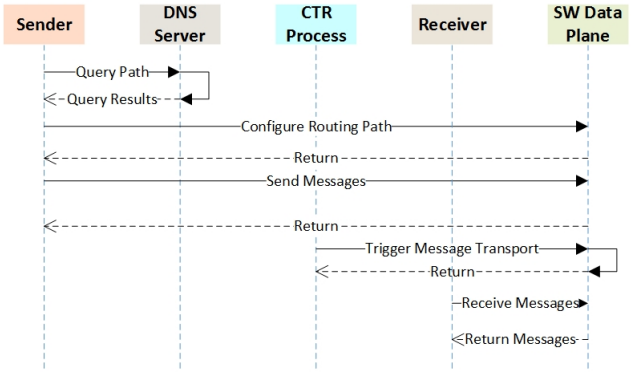

Fig. 5. Process of Data Communication

In this scenario, we measured the consumed time for a partition to send and receive a message. In our implementation, if one sending partition sends more than one message within one time window, there will be no context switching for the sending thread and the sending delay will be dramatically decreased. In order to make the test scenario as close as possible to the worst case scenario and based on the thread model of PikeOS, we ensured that each sending partition sends at most once within one time window.

In Fig.6, the results consist of the consumed time for Partition 1 and Partition 2 to send 100 messages. The measured sending process starts from sending out the query to the DNS Server and ends at the successful return from sending out a message. This figure also depicts the results of measuring the delay for both partitions to configure the SW Data Plane, before sending out the messages. On average, the consumed time for configuring the SW Data Plane takes more than 50% of the sending delay.

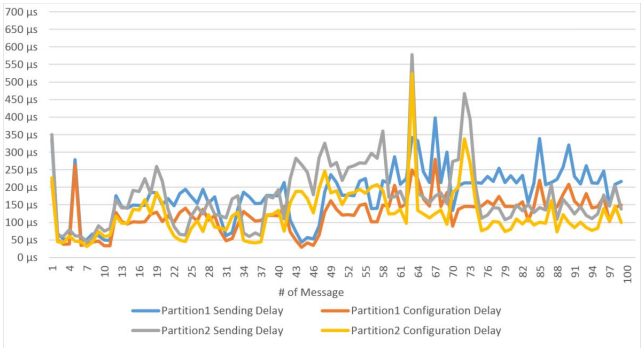

Fig. 6. Sending and Configuration Delay

Similar to Fig.6, the results in Fig.7 depict the receiving delay for Partition 3 and Partition 4. The receiving process covers two steps: sending out the receive command and receiving the expected message. On average, receiving a message takes about 100 microseconds less than sending a