Virtual Switch for Integrated Real-Time Systems based on SDN

Hongjie Fang and Roman Obermaisser

University of Siegen, Germany

Abstract— **The development trend from federated to integrated architectures, which promise massive cost reduction through integration of different functions on the same computing platform, calls for the real-time data exchange mechanism of the integrated functions. An integrated architecture means that different functional components are integrated on a computing node, which were formerly allocated to physically separated computing nodes. Based on an integrated architecture, a virtual switch supporting time-space isolation and dynamic configuration has been proposed in our previous work. However, the virtual switch that is Time Sensitive Networking (TSN) enabled is an open research problem.**

In this paper, we propose a virtual switch that is IEEE 802.1Qbv and IEEE 802.1Qci capable according to the TSN standard. We deploy the Linux LXRT/RTAI on a PC platform with 6 physical cores of 3.2 GHz as the execution environment, and implement the virtual switch in a resource dedicated way for the proof-of-concept implementation. The experimental results show the timely deterministic behaviour of the data switching in our proposed virtual switch.

I. INTRODUCTION

Recent advances in the semiconductor industry drive the development trends of the computing platforms to evolve from federated architectures to integrated ones. Extensive research on integrating multiple functions with different criticality levels on a shared platform was carried out in domains like avionic and automotive systems. Corresponding industry standards like ARINC 653 and AUTOSAR were released. In ARINC 653 [26], the functions of different Safety Integrity Levels (SILs) [11] can be integrated based on the APlication/EXecutive (APEX) interface without interfering with each other. The AUTOSAR standard defines a Runtime Environment (RTE) to integrate software components from different manufacturers without specific knowledge of the underlying hardware platforms [30]. Mechanisms like temporal and spatial partitioning are designed to prevent fault propagation between different applications and guarantee the reserved computing resources for each application, so that safety critical applications cannot be affected by applications of lower criticality. From the viewpoint of system development, partitioning is one of the necessary prerequisites for modular certification [23], which enables the integrated applications to be certified to the respective criticality levels.

Another perspective coming along with integrated architectures is the internal dependable communication mechanism between applications within a platform. A dependable communication mechanism should provide temporal and spatial isolation for safety-critical data flows, meanwhile guaranteeing the required bounded latency with low jitter during data transmission. According to the "fault-error-failure" chain discussed in [24], the essential reason of the observable system failures is the system fault. Partitioning mechanism contributes to prevent fault propagation within an integrated platform. In ARINC 653, the defined communication mechanism can be classified into inter-partition and intra-partition communication. The standardised inter-partition communication mechanism is the static message-based virtual channel, which logically links partitions via ports and the implementation is platform specific [33]. The software components within AUTOSAR communicate with each other via the Virtual Functional Bus (VFB) [30], which forms a middleware that provides the infrastructure-independent communication mechanism. Both of these communication mechanisms target static offline configurations, in another word, dynamic reconfiguration during operation is not addressed.

To date, the Software-Defined Networking (SDN) paradigm has widely caused the network designers to rethink the classical data forwarding approach [16]. In the SDN paradigm, the network control plane and the data forwarding plane are separated from each other, which enables the forwarding tables in the data plane to be reconfigured during runtime. 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 [25]. For example, the Open vSwitch [17] instantiates the SDN paradigm based on OpenFlow and enables the data communication between virtual machines in the cloud computing environments in a layered way (i.e. separated control plane and data plane). However, for the safety critical data communication in an integrated environment, bounded latency with low jitter between communicating entities should be taken into account, which is beyond the scope of Open vSwitch.

Ethernet technologies have been successfully adopted in many application areas and safety critical applications are evolving towards more bandwidth and temporal predictability. As the conventional network infrastructures have no capabilities to guarantee bounded latency and low jitter for hard real-time applications, the IEEE Time Sensitive Networking (TSN) task group has introduced several extension protocols to the IEEE 802.1 Ethernet standard, in order to enable the standard Ethernet devices to support both critical and non-critical traffic. The protocols Qbv and Qci in the TSN protocol set regulate the timing control behaviour of ingressing and egressing processes in a physical switching entity, so that identified critical data flows can be guaranteed with respect to deterministic relay.

Inspired by SDN and TSN, we propose a virtual switch residing in an integrated environment to provide deterministic message switching services by leveraging the SDN paradigm and TSN. The major contributions of this paper are as following:

- TSN enabled virtual switches in integrated execution environment
- SDN capable architecture tackling dynamic reconfiguration
- Schedule model and dispatching algorithm with time-triggered control

The remainder of the paper is structured as follows. Section II summarises the related work on virtualized switching based on an integrated architecture. The concept of the virtual switch is described in detail in Section III. Section IV presents the proof-of-concept implementation of the proposed virtual switch and the experimental results are shown and discussed in Section V. This paper ends with a conclusion in Section VI.

II. Related Work

In this section, we analyze the related work regarding the communication for integrated real-time systems. In addition, we discuss the development of virtual switching technology. The major object of this work is to resolve the deterministic data switching problem in an integrated platform, so the related TSN protocols (IEEE 802.1Qbv and IEEE 802.1Qci) are also addressed in this section.

A. AUTOSAR and ARINC 653

In the AUTOSAR standard, the VFB is defined as the communication mechanism to support interconnection between software components and provide separation between applications and the underlying infrastructures. The communication mechanism in AUTOSAR is majorly classified into client-server and sender-receiver paradigms [12], [19]. As researched by the authors in [6], the timing aspect was not addressed due to the primary objective that aims at supporting system integration from the viewpoint of software engineering. The authors also reveal model mismatches that lead to unpredictable timing behaviours of applications. Long, Rongshen, et al. [7] propose extended mapping rules of AUTOSAR runnable entities to optimize the communication timing within a computing node. The major idea is to refine the mapping of runnable entities to the underlying OS tasks, in order to reduce the context switch and consequently the communication delay.

In the ARINC 653 standard, the inter-partition communication is based on statically configured channels between partitions and the implementation of channels is infrastructure-specific. In the SCARLETT project, Bieber et al. [8] proposed the preliminary design of a reconfigurable Integrated Modular Avionics (IMA) platform and defined the switching process tackling the system reconfiguration problem under the identified safety constraints [10]. In the DIANA project, Engel, Christian, et al. in [9] proposed a configuration selection mechanism by exploiting a byzantine agreement algorithm to activate one of the pre-qualified configurations. To the best of our knowledge, the communication within the integrated platforms in the avionic domain is based on switching between pre-defined configurations. The integrated avionic platforms typically come with the Avionics Full-Duplex Ethernet (AFDX) that is specified in ARINC 664 Part 7 [2]. AFDX leverages the concept of Virtual Links (VL) to convey Rate-Constrained (RC) messages while satisfying the timing and bandwidth requirements for distributed avionic applications [5].

B. Virtual Networking

In the research field of virtual networking based on an integrated architecture, the authors in [3] proposed virtual network services on top of time-triggered communication for both safety-critical and non safety-critical applications. The proposed virtual networks leverage offline configured temporal isolation between applications to ensure encapsulation of different communication entities. The proposed virtual network was implemented in a DECOS node computer [4]. Dobrescu, Mihai, et al. validated in [14] that packet processing using a parallel approach (i.e. one core, one task) is better than a pipeline within cores on an multicore platform with respect to predictable performance, which could be improved by contention-aware scheduling. Regarding the management of the virtualized networks, research on adopting SNMP or Netconf were done in recent years [13], [28].

C. Software Switch

Open vSwitch is one of the most famous OpenFlow switches and it is widely adopted in the cloud computing field. In [22], Mian et al. validated that Open vSwitch is secure at the cost of increased round trip times in comparison to non-virtualized execution environments for cloud computing. He and Liang [18] evaluated the security, Quality of Service (QoS) and network performance of Open vSwitch, which showed positive results. In order to guarantee QoS, Akella and Xiong [21] proposed to allocate bandwidth for satisfying QoS requirements of the priority cloud users based on Open vSwitch. In this research, the proposed QoS approach uses a bandwidth and path length based metric and queuing techniques for different users. Other existing software switches (e.g.,VALE switch [15], mSwitch [27]) are mainly concerned about throughput, packet rates, etc. However, for the safety critical data communication in real-time systems, the bounded latency with low jitter between communication entities is an important requirement to be addressed [20].

D. Time Sensitive Networking

TSN enables standard Ethernet infrastructures to converge both safety-critical and non safety-critical traffic. In [36], the authors evaluated the Qbv and Qci sub-protocols of the TSN protocol set in a simulated way and show the determinism with respect to the ingress and egress policing that are defined in the TSN protocols. Raagaard, Michael Lander, et al. in [34] address the dynamic adaptation of TSN enabled networks. They proposed a heuristic algorithm to recalculate the Gate Control Lists (GCL) for the switching entities and leveraged the NETCONF protocol to reconfigure the switching entities. Extensive research on routing and scheduling for the TSN enabled switches were carried out ([37], [35], [32], [31], [29]), which continually drive TSN to be widely adopted in the industry and academia.

III. Virtual Switch

Inspired by the development of TSN and due to the evolving trend from federated architectures to integrated ones, we propose a Qbv and Qci enabled virtual switch residing on an integrated platform, in order to close the research gap of virtual switching guaranteeing bounded delay with low jitter.

A. Requirements of the Virtual Switch

The obligatory functionality of a virtual switch is to provide message switching services. In this section, we analyze the requirements of the virtual switch with respect to the target services.

1) Timing Determinism: The virtual switch aims at providing a communication infrastructure for real-time applications, therefore timing determinism is a prerequisite. From the viewpoint of a switching entity, the timing requirement should apply both for ingress and egress points.

2) Spatial Isolation: For integrated applications within a computing platform, in order to exclude the spatial interference between applications of different safety-criticalities, the virtual switch should be designed to provide applications with dedicated resources to rule out unintended resource interference.

3) Dynamic Reconfiguration: In the above mentioned related work, the networking facilities for integrated applications either guarantee deterministic message switching with offline defined scheduling and routing, or concentrate on services that are not hard real-time capable. In contrast, our virtual switch is online reconfigurable and also able to ensure bounded latency with low jitter.

B. Virtual Switch

The virtual switch leverages the mechanisms defined in the TSN Qci and Qbv sub-protocols to achieve the timing policing. Assume that Time-Triggered (TT) traffic is selected for the transmission of safety-critical messages, each TT message arrives at the ingress port of the switch at a specific point in time according to the schedule. The virtual switch checks the incoming time of TT messages based on a predefined time-based Access Control List (ACL) and relays the temporally correct messages to the queues of their egress ports, which are determined based on the routing tables. Regarding the egress policing, the Gate Control List (GCL) defined in the Qbv sub-protocol is specified to control the dequeuing process of each egress port. The ACLs and GCLs within a virtual switch are aligned with each other by taking the relay overhead into account.

C. System Model

For the Qbv and Qci enabled virtual switch, we model the system from the viewpoints of the system architecture and the applications.

1) Architecture Model: As depicted in Fig.1, we define an example architecture model with multiple virtual switches (i.e., VSW1 and VSW2 in Fig.1) residing on an integrated platform. In order to demonstrate the functionalities of the virtual switches, multiple virtual end systems (i.e. VESn in Fig.1) are also defined. The virtual end systems communicate with each other via the virtual switches, which result in two communication scenarios:

- VES $\rightarrow$ VSW $\rightarrow$ VES: this scenario represents the case that the communicating virtual end systems are connected to the same virtual switch.
- VES $\rightarrow$ VSW $\rightarrow$ VSW $\rightarrow$ VES: this scenario represents the case that the communicating virtual end systems are connected to different virtual switches on the same platform.

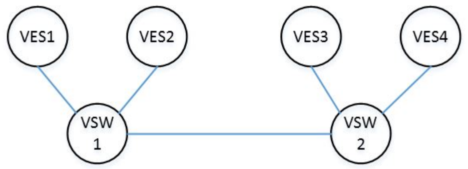


Fig. 1. Architecture Model

2) Application Model: Except from the architecture model, we define the example application model in Fig.2. In this model, the messages (i.e., m1, m2 and m3) between the tasks (i.e., T1, T2, T3 and T4) build up the task dependencies that result in the muster task scheduling in Fig.3. This muster scheduling indicates the general execution order of the tasks, while task activation and execution duration are implementation specific.

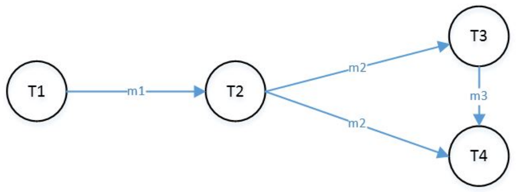


Fig. 2. Application Model

3) Virtual Switch Constraints: According to the example architecture model and the above mentioned requirements, we describe the detailed constraints of the virtual switch as follows.

- Synchronised Start: Within an integrated platform, the virtual switches should have the synchronised time resources and start in a synchronised manner.
- Relay Overhead: The relay overhead for critical messages within a virtual switch should be bounded with low jitter.

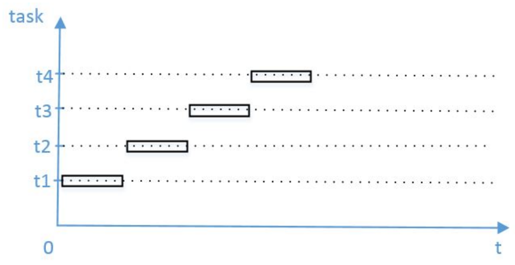


Fig. 3. Task Scheduling

- Exclusive Egressing: Although an egress port owns dedicated queues, passing through the egress port should be exclusive for each message at a specific point in time.
- Egress Queuing: In order to ensure deterministic message switching, the messages of different data flows should be either assigned to different queues, when they are relayed to the same egress port, or the messages should be placed in the same queue in a temporally separated way to avoid messages from different data flows being dispatched together.

4) Schedule Model: From the viewpoint of the integrated platform, the mapping of tasks should be done before discussing the overall schedule problem. An example mapping from the application model to the architecture is as follows:

- T1 $\rightarrow$ VES1
- T2 $\rightarrow$ VES2
- T3 $\rightarrow$ VES3
- T4 $\rightarrow$ VES4

Following the sketched task scheduling in Fig.3 and the above given mapping, the schedule model covering the tasks and virtual switches is shown in Fig.4. In this schedule model, the tasks and virtual switches are activated periodically and the sketched gantt chart presents the schedule within one period.

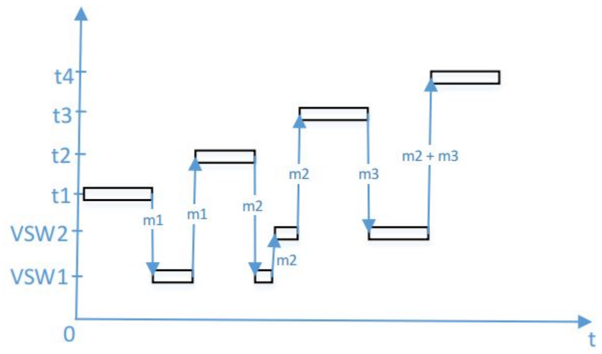
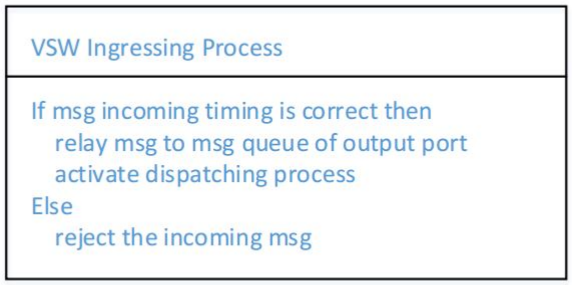


Fig. 4. Schedule Model

The discussed communication scenarios (i.e., data flow within a virtual switch, data flow through multiple virtual switches) are mapped in Fig.4. Message m1 and m2 correspond to the above identified scenarios, moreover, m2 and m3 dispatched from VSW2 result in the multiplexing situation at the egress port connected to t4, which calls for the target dispatching algorithm.

D. Dispatching Algorithm

As discussed, based on the proposed schedule model, we present in this subsection the algorithm of ingressing and dispatching process for the virtual switches using time-triggered scheduling.

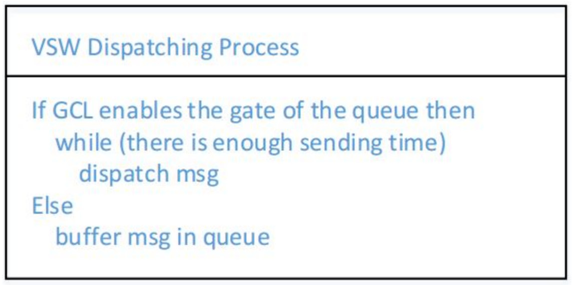


```
VSW Ingressing Process

If msg incoming timing is correct then
  relay msg to msg queue of output port
  activate dispatching process
Else
  reject the incoming msg
```

Fig. 5. Ingressing Process

If an incoming message is safety-critical, as depicted in Fig.5, the incoming timing is checked against the configured ACL. Only the messages arriving in their assigned time slots are relayed to the egress ports. Otherwise the received messages are treated as untimely and abandoned. Right after enqueuing the messages at the egress ports, the dispatching process should be activated.



```
VSW Dispatching Process

If GCL enables the gate of the queue then
  while (there is enough sending time)
    dispatch msg
Else
  buffer msg in queue
```

Fig. 6. Dispatching Process

According to Fig.6, when the dispatching process is called, if the GCL enables the selected queue of a egress port to transport messages, then the selected queue is authorised for data transmission. The other prerequisite for message transmission is that there should be enough time left for the selected queue to dispatch the enqueued messages. Otherwise the messages should be buffered.

IV. Proof of Concept Implementation

In the following section we describe the proof-of-concept implementation of the virtual switch. We discuss details of the employed platform that is multiprocessor computing node. Thereafter we also describe the detailed implementation of the virtual switch and the emphasis is placed on the deterministic message switching by leveraging the TSN mechanisms.

A. Platform

The implementation runs on the PC platform with 16GB RAM and 6 physical cores of 3.2GHz. We deploy on this platform the real-time Linux LXRT/RTAI [1] as the execution environment of the developed virtual switches. In order to provide dedicated computing resources for each virtual switch, we reserve two of the physical cores and run the virtual switches on the isolated cores in a one to one mapping. Therefore the context switches on the dedicated physical cores are avoided, which may cause unpredictable behavior during runtime.

B. Realisation of Virtual Switch

We implement a virtual switch as a kernel module, which can access the RTAI real-time scheduler and services that are implemented as kernel modules. As discussed above, one virtual switch is exclusively assigned to one physical core.

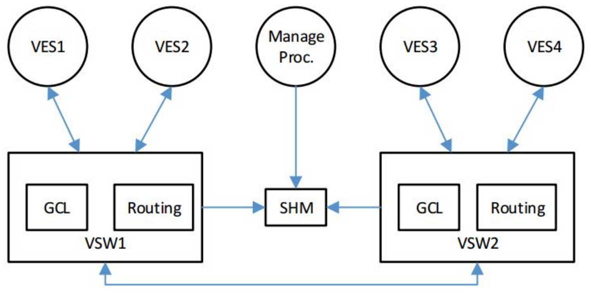


Fig. 7. General Implementation

As depicted in Fig.7, we leverage the real-time fifos provided by RTAI to implement the communication channels between virtual end systems and virtual switches, and the communication channel between virtual switches. The shared memory (i.e., SHM in Fig.7) is used to store the up-to-date configuration parameters (e.g., routing information, GCL, etc.) that is managed by the manager process and consumed by the virtual switches. From the viewpoint of the computing platform, this way contributes to the unified configuration for multiple virtual switches that reside on the same platform.

Regarding the local scheduling of a virtual switch, the related data structures are shown as follows. From the viewpoint of a virtual switch, each egress port within the virtual switch should have its own schedule, more specifically, the GCL. In order to guarantee low jitter from design, we implement the whole set of GCLs as a static array with pre-defined maximum count of egress ports, instead of dynamically linked list. For the schedule of each egress port, the GCL_duration defines the period of the cyclically enabled queues within this egress port. And this period is subdivided into time intervals (i.e., GCL_item) which correspond to different queue masks. The queues of critical data flows are exclusively enabled to transmit messages, so that one critical data flow is timely isolated from other flows.

```
static struct egress_port_schedule
{
    struct schedule egress_port_GCL[count_egr_port];
};

static struct schedule
{
    unsigned int GCL_duration;
    struct GCL_item GCL[GCL_len];
};

static struct GCL_item
{
    int offset;
    int duration;
    uint8_t queue_mask;
};
```

As we mentioned before, the real-time fifos are leveraged to implement the communication channels between virtual switches, as well as channels between virtual end systems and virtual switches, we implement the egress port to be one-to-one mapped to the fifos. In another word, the queues of an egress port multiplex the fifo attached to this port. Each queue records the number of the stored messages with a predefined maximum count. The way in our implementation to rule out messages of different critical data flows interleaving through the fifo is that each critical data flow owns one dedicated queue.

```
static struct egress_port
{
    int output_fifos_id;
    struct queue all_queues[queue_count];
}egress_port;

static struct queue
{
    int item_count;
    int head_index;
    int tail_index;
    struct queue_item queue[queue_len];
}queue;

static struct queue_item
{
    int msg_len;
    char item[max_msg_len];
}queue_item;
```

Besides the data structures related to the egress ports, each ingressing message is stored in a vsw_ingress_buf. After classifying the stored message, the found egress ports are recorded before activating the relay process. We will discuss more details about the message switching process in the following sections.

```
static struct vsw_ingress_buf
{
    int msg_len;
    uint8_t vsw_rv_msg[max_msg_len];
}vsw_ingress_buf;

static struct matched_egress_port
{
    int count;
    int port_cookie[count_egr_port];
}matched_egress_port;
```

The implemented workflow of the message switching within a virtual switch in depicted in Fig.8. An ingressing port waits for the semaphore that is assigned to the attached fifo, which as discussed connects different communicating entities. Except for the timely synchronised message switching, this implementation also leverages the semaphore as the synchronisation mechanism between communicating entities. The incoming time of a message is checked against the ACL after classification and finding the egress ports. The successfully ingressed critical messages are enqueued in the egress ports and dispatched if the GCL enables the enqueued queue for data transmission.

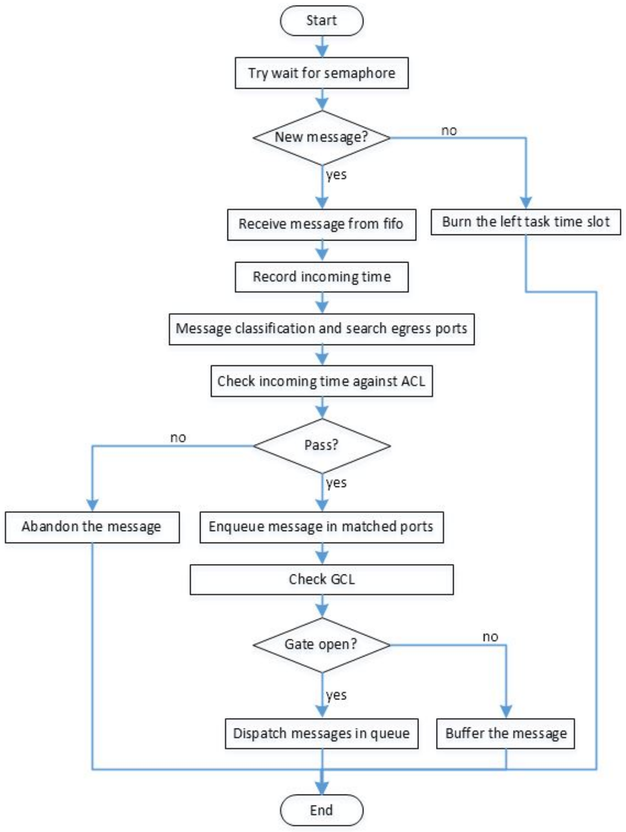

Start
Try wait for semaphore
New message?
no
yes
Receive message from fifo
Burn the left task time slot
Record incoming time
Message classification and search egress ports
Check incoming time against ACL
Pass?
no
yes
Abandon the message
Enqueue message in matched ports
Check GCL
Gate open?
no
yes
Dispatch messages in queue
Buffer the message
End

Fig. 8. Workflow of Virtual Switch

In this implementation, we use the local system clock as the global time for virtual end systems and virtual switches. The synchronised activation of multiple virtual switches is done by adapting the start delay of a latter loaded virtual switch to a predefined delay, so that all virtual switches are activated simultaneously.

C. Temporal & Spatial Isolation

For each virtual switch that owns a dedicated physical core, we implement separate processes to manage the ingressing ports of the connected entities. For example, in VSW1 in Fig.7, we implement two processes to manage the ingressed messages from VES1 and VES2, correspondingly. As shown in Fig.9, each process is run exclusively in a 10 ms period and for the duration of 5 ms, which ensures the temporal isolation between processes in a virtual switch and consequently eliminates race conditions of the processes by design.

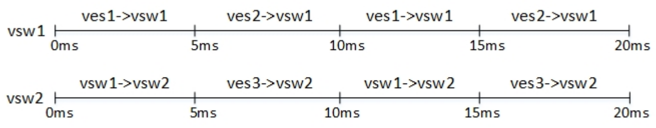


Fig. 9. Task Schedule in Virtual Switch

In the proof-of-concept implementation, assume that each egress port owns 8 queues and two of them are assigned to time-triggered data flows, we use the example GCL (see TABLE I) for all egress ports. For simplification purpose, we assume the frame processing time is neglectable, so that the ACL is identical to the GCL.

TABLE I
EXAMPLE GATE CONTROL LIST

start time	duration	queue mask	remark
$0\mu s$	$30\mu s$	10000000	time-triggered frames
$30\mu s$	$30\mu s$	01000000	time-triggered frames
$60\mu s$	$40\mu s$	00111111	other frames

D. Realisation of Application

Since the virtual end systems are not the emphasis of this paper, we implement the tasks as a simple linux task sending message periodically and waiting for the incoming messages. For the delay measurement purpose, we insert a time stamp of u_int64_t type between the default Ethernet header and the data as shown in Fig. 10.

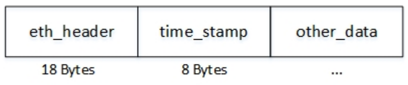


Fig. 10. Example Frame Format

Since T1 in Fig.2 provides the source message of the whole setup, we configure the T1 to send the initial message in a 5 ms period, and the other tasks work in a busy waiting way.

V. EXPERIMENTAL RESULTS AND DISCUSSION

In this experiment, we investigate the determinism of the message switching within the virtual switches. As discussed in section III-C.1, there are two identified communication scenarios in our system setup, which happen either within a virtual switch boundary or involve multiple virtual switches.

We measure the message switching overhead caused by the virtual switches, more specifically, the consumed time from the ingress ports to the egress ports that are connected to the source and sink virtual end systems. The sending virtual end system is configured to send out 1000 messages. The Fig.11 and Fig.12 show the overhead of the local switching within a virtual switch.

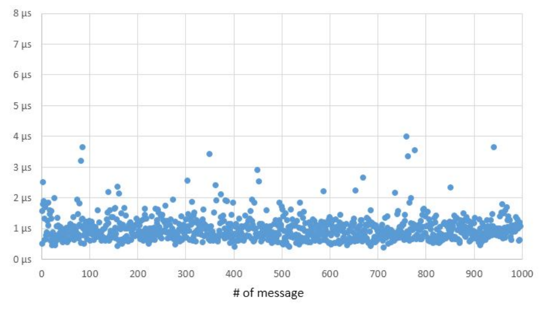


Fig. 11. Overhead for Data Flow between VES1 and VES2

The overall latency caused by the virtual switches for the local message transportation is in the range from 0.5 μs to 2 μs. Several exceptions up to 5 μs could be observed in Fig.11 and Fig.12. In the implementation, we isolate one physical core to run a virtual switch, as discussed in section IV-C. Since timing resource is necessary to enable the process switching, one aspect to mention is that hardware timer interrupts are redirected to other physical cores in this case, which requires the inter-core communication for measuring the overhead that could result in observable jitters. Another aspect is that the jitter caused by the RTAI scheduler could also accumulate to the significant jitters in the measured results.

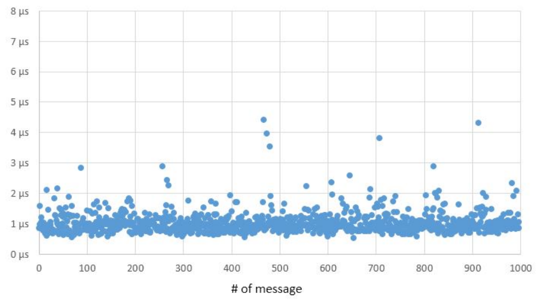


Fig. 12. Overhead for Data Flow between VES3 and VES4

The overhead for data flow between VES2 and VES3/VES4 in Fig.7 is measured one after another within VSW2, therefore the results in Fig.13 and Fig.14 are in the same distribution. The difference of the measured overhead for the same message (depicted in Fig.15) indicates the overhead for enqueuing and dispatching a message, since a message from VES2 is relayed to VES3 and VES4 in order.

As discussed in section IV-C, the ingressing processes of the virtual switches are run in a 10 ms period and each with 5 ms runtime, and the relay/dispatching actions are also finished in the ingressing process context. In our configuration, the process in VSW2 for receiving messages from VSW1 is scheduled to run after the dispatching process in VSW1. In another word, the VSW2 receives messages from VSW1 in about 5 ms, after VSW1 dispatches messages to VSW2. This logical analysis is also confirmed by the results in Fig.13 and Fig.14, which show the switching overhead for a data flow passing through two virtual switches.

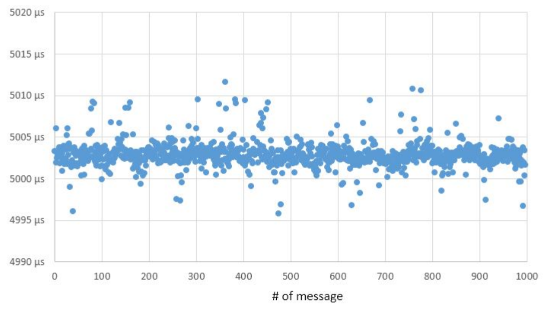


Fig. 13. Overhead for Data Flow between VES2 and VES3

Similar to the measured local delay caused by virtual switches that are depicted in Fig.11 and Fig.12, the majority of the overhead in Fig.13 range from 5000 μs to 5005 μs. Despite the delay caused by schedule (i.e., 5 ms), the left overhead (i.e., 0-5 μs) is implementation related that need to be discussed. As aforementioned, the major reasons for this observable overhead are inter-core communication due to timer on other core and the jitter caused by RTAI scheduler. For this inter virtual switches communication scenario, jitters could accumulate along the message routing path, which result in the wider range of overhead (5 μs) than the intra virtual switch communication (2 μs).

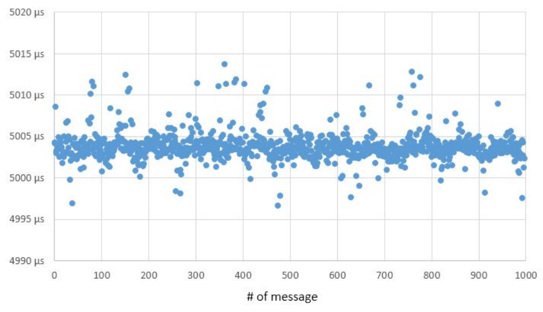


Fig. 14. Overhead for Data Flow between VES2 and VES4

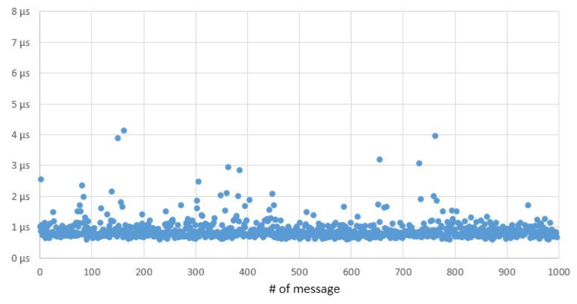


Fig. 15. Overhead for Enqueuing & Dispatching of one Message

VI. Conclusion and Future Work

In this work, we propose the IEEE 802.1Qbv and IEEE 802.1Qci enabled virtual switch for integrated real-time systems residing on single platform and define the model of communication infrastructure with multiple virtual switches.

The corresponding schedule model is defined as the base for the dispatching algorithm in a time-triggered way. The proof-of-concept implementation is done in a resource dedicated way by leveraging the Linux RTAI patch. The experimental results demonstrate the capability of the virtual switch to switch messages in a timely deterministic way.

In this work, the system setup runs on a single computing node. The future work will be the hybrid switching environment with physical and virtual switches to enable a physically networked system, which also brings the challenges in the consistent dynamic configuration of the whole system.

VII. ACKNOWLEDGMENT

This project was funded by the European Regional Development Fund (ERDF).

REFERENCES

[1] Beal, Dave, et al. "RTAI: Real-time application interface." Linux Journal 29.10 (2000).

[2] "AIRCRAFT DATA NETWORK PART 7 AVIONICS FULL DUPLEX SWITCHED ETHERNET (AFDX) NETWORK". ARINC Specification 664p7 (2005).

[3] Obermaisser, Roman, Philipp Peti, and Hermann Kopetz. "Virtual networks in an integrated time-triggered architecture." 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems. IEEE, 2005.

[4] Obermaisser, Roman, and Philipp Peti. "Realization of virtual networks in the DECOS integrated architecture." Proceedings 20th IEEE International Parallel & Distributed Processing Symposium. IEEE, 2006.

[5] Alena, Richard L., et al. "Communications for integrated modular avionics." Aerospace Conference, 2007 IEEE. IEEE, 2007.

[6] Racu, Razvan, et al. "Automotive software integration." Proceedings of the 44th annual Design Automation Conference. ACM, 2007.

[7] Long, Rongshen, et al. "An approach to optimize intra-ecu communication based on mapping of autosar runnable entities." 2009 International Conference on Embedded Software and Systems. IEEE, 2009.

[8] Bieber, Pierre, et al. "Preliminary design of future reconfigurable IMA platforms." ACM Sigbed Review 6.3 (2009): 7.

[9] Engel, Christian, et al. "Enhanced dispatchability of aircrafts using multi-static configurations." Embedded Real Time Software and Systems Congress (ERTS 2010), Toulouse, France. 2010.

[10] Bieber, Pierre, et al. "Preliminary design of future reconfigurable IMA platforms-safety assessment." 27th Congress International Council of the Aeronautical Sciences (ICAS 2010). 2010.

[11] IEC 61508 Edition 2.0: Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems, IEC: Int. Electrotechnical Commission, 2010.

[12] Dafang, Wang, et al. "Communication mechanisms on the virtual functional bus of AUTOSAR." 2010 International Conference on Intelligent Computation Technology and Automation. Vol. 1. IEEE, 2010.

[13] Daitx, Fabio Fabian, Rafael Pereira Esteves, and Lisandro Zambenedetti Granville. "On the use of SNMP as a management interface for virtual networks." 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops. IEEE, 2011.

[14] Dobrescu, Mihai, Katerina Argyraki, and Sylvia Ratnasamy. "Toward predictable performance in software packet-processing platforms." Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12). 2012.

[15] Rizzo, Luigi, and Giuseppe Lettieri. "Vale, a switched ethernet for virtual machines." Proceedings of the 8th international conference on Emerging networking experiments and technologies. ACM, 2012.

[16] Levin, Dan, et al. "Logically centralized?: state distribution trade-offs in software defined networks." Proceedings of the first workshop on Hot topics in software defined networks. ACM, 2012.

[17] He, Zongjian, and Guanqing Liang. "Research and evaluation of network virtualization in cloud computing environment." Networking and Distributed Computing (ICNDC), 2012 Third International Conference on. IEEE, 2012.

[18] He, Zongjian, and Guanqing Liang. "Research and evaluation of network virtualization in cloud computing environment." Networking and Distributed Computing (ICNDC), 2012 Third International Conference on. IEEE, 2012.

[19] Chen, Hao, et al. "Research on Client/Server Communication Mechanism in AUTOSAR System." 2013 IEEE 11th International Conference on Dependable, Autonomic and Secure Computing. IEEE, 2013.

[20] Obermaisser, Roman, and Donatus Weber. Architectures for mixed-criticality systems based on networked multi-core chips. Emerging Technology and Factory Automation (ETFA), 2014 IEEE. IEEE, 2014.

[21] Akella, Anand V., and Kaiqi Xiong. "Quality of service (QoS)-guaranteed network resource allocation via software defined networking (SDN)." Dependable, Autonomic and Secure Computing (DASC), 2014 IEEE 12th International Conference on. IEEE, 2014.

[22] Mian, Adnan Noor, et al. "Effects of virtualization on network and processor performance using open vswitch and xen server." Utility and Cloud Computing (UCC), 2014 IEEE/ACM 7th International Conference on. IEEE, 2014.

[23] Obermaisser, Roman, and Donatus Weber. "Architectures for mixed-criticality systems based on networked multi-core chips." Emerging Technology and Factory Automation (ETFA), 2014 IEEE. IEEE, 2014.

[24] Laprie, Jean-Claude. "Dependable computing: Concepts, limits, challenges." Special issue of the 25th international symposium on fault-tolerant computing. 1995.

[25] Jarraya, Yosr, Taous Madi, and Mourad Debbabi. "A survey and a layered taxonomy of software-defined networking." IEEE Communications Surveys & Tutorials 16.4 (2014): 1955-1980.

[26] Airlines Electronic Engineering Committee. Avionics application software standard interface part 1-required services. ARINC Document ARINC Specification 653P1-4, Aeronautical Radio, Inc., Maryland, 2015.

[27] Honda, Michio, et al. "mSwitch: a highly-scalable, modular software switch." Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research. ACM, 2015.

[28] Santos, Paulo Roberto da Paz Ferraz, Rafael Pereira Esteves, and Lisandro Zambenedetti Granville. "Evaluating SNMP, NETCONF, and RESTful web services for router virtualization management." 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM). IEEE, 2015.

[29] Drr, Frank, and Naresh Ganesh Nayak. "No-wait packet scheduling for IEEE time-sensitive networks (TSN)." Proceedings of the 24th International Conference on Real-Time Networks and Systems. ACM, 2016.

[30] Safe4RAIL Project, "D2.1 Report on state-of-the-art of 'functional distribution architecture' frameworks and solutions", 2016.

[31] Craciunas, Silviu S., et al. "Scheduling real-time communication in IEEE 802.1 Qbv time sensitive networks." Proceedings of the 24th International Conference on Real-Time Networks and Systems. ACM, 2016.

[32] Craciunas, Silviu S., R. Serna Oliver, and TTTech Computertechnik AG. "An overview of scheduling mechanisms for time-sensitive networks." Proceedings of the Real-time summer school Lcole dt Temps Rel (ETR) (2017).

[33] Fang, Hongjie, and Roman Obermaisser. "Execution Environment for Mixed-Criticality Train Applications Based on an Integrated Architecture." 2017 International Conference on Promising Electronic Technologies (ICPET). IEEE, 2017.

[34] Raagaard, Michael Lander, et al. "Runtime reconfiguration of time-sensitive networking (tsn) schedules for fog computing." 2017 IEEE Fog World Congress (FWC). IEEE, 2017.

[35] Nayak, Naresh Ganesh, Frank Drr, and Kurt Rothermel. "Routing algorithms for IEEE802. 1Qbv networks." ACM SIGBED Review 15.3 (2018): 13-18.

[36] Pahlevan, Maryam, and Roman Obermaisser. "Evaluation of time-triggered traffic in time-sensitive networks using the opnet simulation framework." 2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP). IEEE, 2018.

[37] Nayak, Naresh Ganesh, Frank Drr, and Kurt Rothermel. "Incremental flow scheduling and routing in time-sensitive software-defined networks." IEEE Transactions on Industrial Informatics 14.5 (2018): 2066-2075.