RDMA-Based Deterministic Communication Architecture for Autonomous Driving

Hazem Abaza*†, Abhinaba Habishyashi†, Debayan Roy†, Andrea Bastoni§,
Zain A. H. Hammadeh‡, Shiqing Fan†, Selma Saidi*, and Sergey Tverdyshev†
{*Technische Universität Dortmund*†, *Huawei Munich Research Center**,
Technische Universität München§, *German Aerospace Center*‡}, Germany
{hazem.abaza, selma.saidi}@tu-dortmund.de, abhinaba.habishyashi@mailbox.tu-dresden.de,
{debayan.roy6, shiqing.fan, sergey.tverdyshev}@huawei.com, andrea.bastoni@tum.de, zain.hajhammadeh@dlr.de

***Abstract*—Autonomous driving is a big challenge for next-generation vehicles and requires multiple computationally-intensive deep neural networks (DNNs) to be implemented on distributed automotive platforms. Distributed software—enabling autonomous functionalities—has strict timing requirements, e.g., low and deterministic end-to-end latency. Such timings rely on the communication technologies used in the automotive platform, as much on the computation performance of CPUs, GPUs, TPUs, and FPGAs. Hence, we advocate the use of Remote Direct Memory Access (RDMA) technology—typically used in data centers—in automotive platforms. As shown by our experiments with real hardware, Soft-RoCE (software implementation of RDMA) offers low latency communication because of minimal CPU involvement and reduced memory copies. Simultaneously, we show that the native implementation of RDMA does not support determinism, i.e., there is a high variation in communication delays in the presence of interfering data packets. To mitigate this issue, we propose a multi-layer communication stack comprising a deterministic scheduler on top of the Soft-RoCE layer. Further, we have developed a C++ library that offers easy-to-use communication interfaces for distributed applications while implementing the proposed architecture. Experiments show that our library (i) reduces the end-to-end latency of distributed object detection by nearly 9% while having an implementation overhead of less than 1.5% and (ii) minimizes the effects of other data traffic on the delay in high-priority communication.**

I. Introduction

Enabling increasingly more autonomous driving (AD) or advanced driver assistance system (ADAS) features in next-generation vehicles is one of the key goals of the automotive industry for the upcoming years. AD/ADAS applications heavily rely on deep neural networks (DNNs), e.g., for planning and control, environment perception, and sensor fusion. These DNNs are computationally expensive and typically require customized hardware accelerators for faster processing, which is necessary for AD/ADAS applications. Typically, an end-to-end AD/ADAS application starts with data acquisition from sensors, e.g., cameras and Lidars, and goes all the way to steering, brake, and speed control actuators. In such an application, *data* flows across multiple DNNs deployed on different accelerators which may be distributed in space. AD/ADAS applications have strict timing requirements, e.g., end-to-end latency cannot be more than the typical human reaction time which is around 390 – 600 ms (according to a MIT study [1]). While there is a lot of emphasis on accelerating the computation in DNNs in the best possible way using specially-designed hardware, the *communication delays* between accelerators also play a significant role in determining the final end-to-end latency of the application. Additionally, the amount of data to be transferred between these accelerators can easily exceed tens of megabytes, considering that very precise information about the three-dimensional environment is often contained in such data [2]. To make things worse, the communication network may be shared among multiple data-flow transfers, leading to interference and contention. In this context, this paper focuses on communication technologies to meet the low latency and deterministic timing requirement of AD/ADAS applications.

Andrea Bastoni was supported by the Chair for Cyber-Physical Systems in Production Engineering at TUM and the Alexander von Humboldt Foundation.

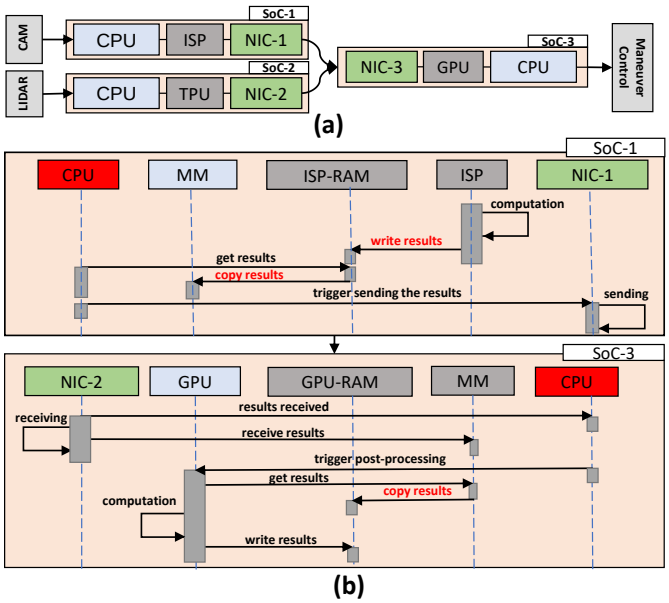


Fig. 1: (a) Platform architecture of an example ADAS application. (b) Sequence diagram showing data transfer between accelerators in different SoCs.

Data flow in current automotive platforms: Figure 1a shows an example ADAS application for pedestrian detection, inspired from [3]. It uses a Lidar point cloud data stream for occupancy grid generation. The camera frames are used for object classification, e.g., to distinguish a pedestrian from

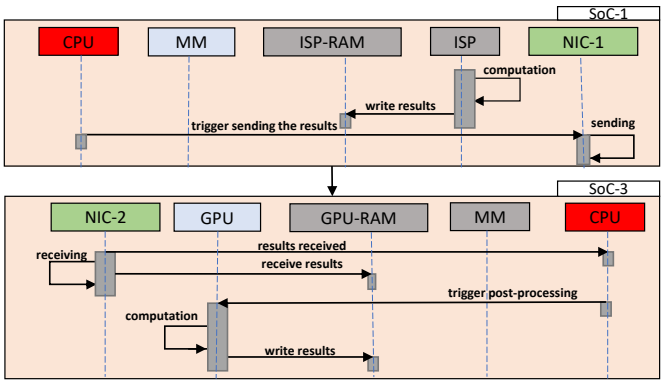


Fig. 2: Sequence diagram showing RDMA-supported data transfer between accelerators in different SoCs

other objects. Here, the camera data is processed on SoC-1 using an Image Signal Processor (ISP), while the point cloud data from Lidar is processed in a Tensor Processing Unit (TPU) on SoC-2. The planning and control decisions are then taken on SoC-3 with the aid of a Graphics Processing Unit (GPU). Hence, the data produced by ISP and TPU, respectively, have to be sent to the GPU in SoC-3. There are two major factors that increase the latency of such data communication between different SoCs [4]: (i) Network access requests from an accelerator to the network interface card (NIC) are performed via the CPU on the same SoC as shown by the sequence diagram in Figure 1b. Processing of such requests consumes millions of CPU cycles for megabytes of data transfer. Also, other workloads running on the CPU can interfere with such requests potentially causing additional delays in the data transfer. (ii) Transferring data between accelerators via CPUs involve unnecessary memory copies as illustrated in Figure 1b, which again delay the communication.

Proposed data flow for AD/ADAS applications: To improve the inter-SoC communication latency, CPU involvement and memory copies need to be reduced as much as possible. Remote Direct Memory Access (RDMA) [5] is a technology that can enable direct access of the memory in one SoC (e.g., ISP's RAM) by a processing unit in another SoC (e.g., GPU in SoC-3). RDMA is normally implemented at hardware level (e.g., integrated into a NIC) with limited extension possibilities. Instead, in this paper, we focus on Soft-RoCE, a software implementation of RDMA over a converged Ethernet (RoCE) network [6]. Soft-RoCE is compatible with any Ethernet NIC since it does not use hardware acceleration. Hence, it allows RDMA technology to be integrated in a scalable, portable, and hardware-independent manner. With Soft-RoCE, the CPU is able to execute other workloads, while, on the same SoC, data is being read by a remote accelerator. Also, note that it significantly reduces memory copies. This paper advocates the use of RDMA technology for automotive applications, in particular, data-intensive AD/ADAS applications, because it reduces the latency of transmitting a large amount of data between distributed SoCs compared to communication protocols like Transmission Control Protocol (TCP) [7]. Figure 2 depicts the updated sequence diagram for inter-SoC data transfer in the pedestrian detection application (from Figure 1). Data produced by the ISP on SoC-1 (or TPU on SoC-2) is directly read by the GPU on SoC-3 with a minimal involvement of the CPU on SoC-1 (or the CPU on SoC-2).

Shortcomings of Soft-RoCE for AD/ADAS applications: As shown in Section IV, our experiments highlight shortcomings of the native implementation of Soft-RoCE when used by distributed applications with firm real-time requirements (i.e., where a deadline miss will result in performance degradation). The experiments show nondeterministic timing behaviors when multiple data transfers are performed simultaneously using Soft-RoCE between two SoCs. Specifically: (i) When simultaneous data transfers have the same quality-of-service (QoS) requirements, e.g., static priorities, we have observed that the network bandwidth is shared among such flows arbitrarily, which is not desirable. (ii) There is no notion of priority in Soft-RoCE, i.e., a latency-sensitive data transfer cannot be prioritized over interfering best-effort data packets.

Proposed communication stack: We propose a communication architecture to enable deterministic timing behavior in distributed applications that use Soft-RoCE for communication. We introduce a *Flow Control* layer that manages the calls to the Soft-RoCE layer for sending data. We can implement any scheduling mechanism in this layer with support for different QoS metrics, e.g., data freshness, static priority, and deadline. Note that this architecture does not modify the native implementation of Soft-RoCE. Further, we have developed a communication stack implementing the aforementioned architecture. We have added support for priority-based communication with limited preemption and, hence, it can achieve a lower worst-case and average latency for high-priority data, which is crucial for AD/ADAS applications. Also, the stack hides low-level RDMA implementation details and enables to develop applications using easy-to-use Application Programming Interfaces (APIs) for initialization, handshaking, sending and receiving.

Contributions: Our main contributions are as follows:

- We experimentally show that (i) RDMA can be used to reduce the end-to-end latency of AD/ADAS applications, but that (ii) there are drawbacks when using it for time-sensitive applications.
- We propose a multi-layer communication architecture based on RDMA for deterministic and low-latency data transmission over distributed heterogeneous platforms.
- We implement a lightweight communication stack based on our proposed architecture that provides simple interfaces to be used by distributed applications.
- As a testing use-case, we develop a real distributed AD/ADAS application comprising a YOLO-enabled object detection [8]. Our experiments show that our communication stack (i) reduces the end-to-end latency of the application by nearly 9% and (ii) improves determinism.

Paper organization: In Section II, we provide background on RDMA and show initial results that motivate our work. In

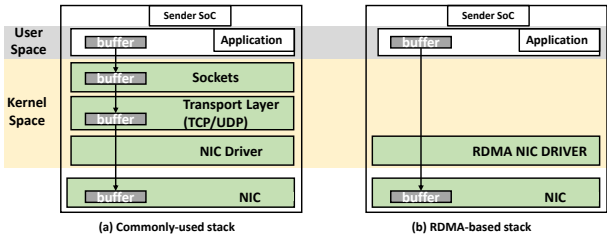


Fig. 3: CPU involvement and memory copies in inter-SoC data transfer with commonly-used stack and RDMA-based stack.

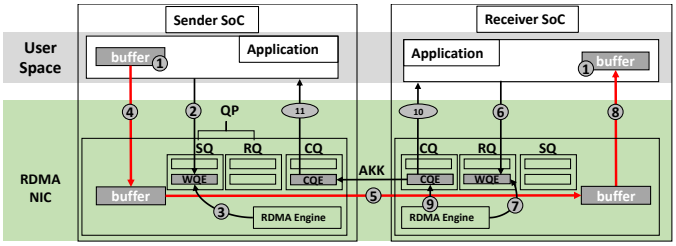


Fig. 4: Different components in an RDMA connection.

Section III, we present our proposed communication stack for low latency and deterministic communication. We describe our experiments and analyze the results in Section IV. Section V discusses the related works. Concluding remarks are provided in Section VI.

II. Remote Direct Memory Access (RDMA)

As mentioned in Section I, RDMA can reduce communication latency for distributed software applications implemented over multiple heterogeneous SoCs. Figure 3 illustrates the main reasons for reduced latency by comparing RDMA against commonly-used communication stacks involving TCP/UDP. In Figure 3a, we see that multiple interventions of the operating system in the CPU are necessary to transfer data through different layers of the conventional communication stack. Also, in the process, data is copied in the buffers of different layers (e.g., in sockets and transport layer) using system calls (e.g., *memcpy()*). Conversely, Figure 3b shows that RDMA sending operation (same will be on the receiving side) bypasses the kernel almost entirely and efficiently transfers the data between the application buffer and the RDMA-supported NIC [9]. Note that this transfer does not involve any system call and is accomplished using direct memory access (DMA) channels. Therefore, we can say that RDMA allows direct access to the application memory (i.e., without involving the operating system and the CPUs on both sending and receiving sides) by remote devices connected in the same network.

A. Communication using an RDMA connection

Figure 4 shows the different components in an RDMA connection and how communication is carried out using them. RDMA communication typically requires a NIC that implements RDMA engine, also called Host Channel Adapter (HCA). In particular, HCA includes all the necessary logic to implement the RDMA protocol. The HCA is placed on a Peripheral Component Interconnect express (PCIe) slot on the SoC and, hence, it can use DMA. To establish an RDMA connection between two SoCs, it is necessary to first reserve and map (or *pin*) memory buffers on the sender and receiver sides and inform the kernel that the registered memory will be used for RDMA communication (1). During the initialization of an RDMA connection, HCA registers are mapped on the memory using which application can directly invoke RDMA transfers, i.e., a fast path is created between the application and the HCA bypassing the kernel. That is, a pair of work queues, called a *Queue Pair* (QP), are generated for communication scheduling on the HCAs at both sending and receiving sides. A QP consists of a *Send Queue* (SQ) and a *Receive Queue* (RQ). Besides the QP, a *Completion Queue* (CQ) is generated to track the completion of a scheduling instruction, also called a *Work Queue Element* (WQE), residing on either of the work queues. The primary content of a WQE is a pointer to the target buffer. In SQ, a WQE contains a pointer to the data that needs to be sent, while in RQ, the pointer in a WQE addresses the buffer where the incoming data has to be placed.

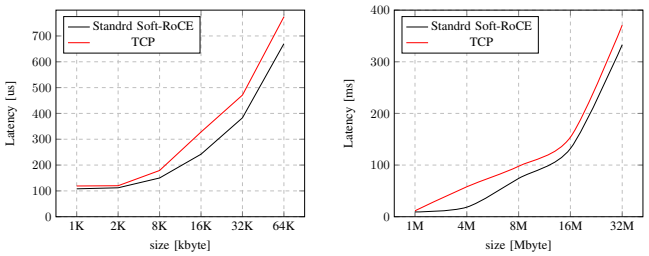
When an application initiates an RDMA send operation, a WQE is created and placed on the SQ in the HCA (2). The HCA polls the QP and, hence, gets the WQE (3). Once the HCA gets a WQE, it processes the WQE and fetches the data from the memory region specified in the WQE to the HCA buffer using DMA (4). The HCA then creates and sends a data packet comprising the data, the SoC address, the RDMA connection identifier, among other information (5).

Simultaneously, at the receiving side, the application creates and places a WQE on the RQ in the HCA (6). Now, when the receiver HCA receives a data packet and identifies the RDMA connection, it checks the corresponding RQ for a WQE (7). If a WQE is available, the HCA puts the data in the memory region specified in the WQE using DMA, otherwise, it rejects the packet (8). If the data transfer is completed successfully, the HCA will put a completion queue element (CQE) on the CQ (9). The application polls the CQ to check if the data is received so that it can continue processing the data (10).

On successful completion, the receiver HCA also sends an acknowledgement to the sender HCA. Once the sender HCA receives the acknowledgement, it also puts a CQE on the CQ at the sending side. As the application polls the CQ also at the sender side, it gets notified that the data is sent successfully (11).

B. RDMA over Converged Ethernet (RoCE)

RDMA semantics of InfiniBand was adapted to run over Ethernet and the corresponding specification (RoCE version 1 or RoCE_v1) was released by InfiniBand Trade Association (IBTA) in April 2010 [10]. RoCE_v1 uses standard Ethernet-based services at the data link layer. RoCE_v1 uses Layer 2 (L2) information and supports packet routing only within an L2 subnet. Later, in 2014, IBTA revised RoCE_v1 and released RoCE version 2 (RoCE_v2) that supports routing of data packets on the network layer [11]. Packet routing across different sub-networks is possible because RoCE_v2



(a) smaller data sizes (b) larger data sizes

Fig. 5: Soft-RoCE vs TCP in terms of communication latency.

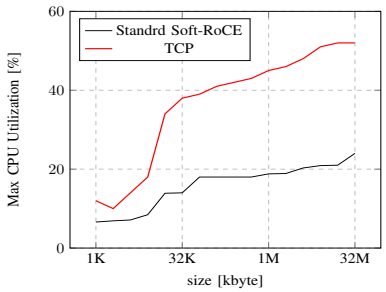


Fig. 6: Maximum CPU utilization with Soft-RoCE and TCP.

uses Layer 3 (L3) information. A global routing header (GRH) is used by the network layer to route RoCE_v2 data packets, which is similar to IPv6 addressing.

Typically, RDMA transfer are carried out using a dedicated hardware RDMA engine, as explained in Section II-A. This increases the dependencies on external hardware and the associated proprietary software. However, these dependencies can be avoided by using Soft-RoCE. Soft-RoCE is a complete software implementation of the RDMA principles that makes RoCE_v2 protocol available for any Ethernet-based network interconnect [6]. It is an open-source Github community project, with contributions from IBM, Mellanox and System Fabric Works and its implementation is available as a Linux kernel module. Soft-RoCE avoids system calls and enables zero copy on the sending side, while it needs only one copy on the receiving side. The reason for this one copy is that the RDMA connection has to be identified for the received data before it can be copied to the corresponding pinned memory buffer. The performance of Soft-RoCE is comparable to RoCE_v2 [6] while it offers more flexibility and allows a complete RDMA implementation over any NIC. Soft-RoCE enables more efficient data transfers compared to the default Ethernet protocol stack, as shown in Section II-C.

C. Low latency communication using Soft-RoCE

To quantitatively assess the benefits of using RDMA with respect to the standard communication stack, we perform experiments where we transfer data between two Intel x86_64 processors running Linux version 5.13.0-51-generic as the operating system. We compare communication latency for the default communication stack (which uses TCP to transfer data) and Soft-RoCE. We use qperf [12] to measure the communication latency. We vary exponentially the data size from 1 Kbyte to 32 MB and for each data size, we perform 150 data transfers. Figure 5 shows the average communication latency with TCP and Soft-RoCE for different data sizes. We can clearly see that Soft-RoCE performs better than TCP. In specific cases the communication latency can be reduced by 36%, e.g., for 4 MB data, Soft-RoCE offers a latency of 37 ms while it is 58 ms with TCP.

We also measure the maximum CPU utilization on the sender side for each data size across all runs both with TCP and Soft-RoCE. The results are shown in Figure 6. We can see that the maximum CPU utilization is much lower with Soft-RoCE in comparison to TCP. For more than 1 MB data, Soft-RoCE can reduce the absolute value by more than 25%, e.g., the max CPU utilization with Soft-RoCE is 20.3% for 4 MB data while it is 48.1% with TCP. These results emphasizes the fact the Soft-RoCE can effectively reduce the CPU load and the CPU can use this time to execute other workloads.

D. Nondeterministic behavior of Soft-RoCE

Considering that multiple applications are running simultaneously on a many-SoC automotive platform, several of them may want to send data over the same physical network link. For such scenarios, multiple RDMA connections can be created between two SoCs [13]. We have experimentally verified that such implementations can also be accomplished using Soft-RoCE. In such implementations, each RDMA connection can be used by an application to transfer specific data (or a series of data produced by the same periodic/sporadic task). For example, in the pedestrian detection application (discussed in Section I), if we have both ISP and TPU on SoC-1 and the GPU on SoC-2, the results of ISP and TPU can be sent to the GPU using two different RDMA connections created between SoC-1 and SoC-2. Each RDMA connection is associated with its own pinned memory buffer, data channels, and queues (discussed in Section II).

While using multiple RDMA connections between two SoCs, we have identified the following major drawbacks that prevent the usage of Soft-RoCE for AD/ADAS applications: (i) When two time-critical applications use different RDMA connections to send their data, we have observed that data packets are transferred in an arbitrary order to the receiver SoC. In certain cases, one application might have to wait for an arbitrarily long time before its data is transferred. In AD/ADAS applications, a deterministic ordering of packets, e.g., first-in first out (FIFO), enable performing a worst-case analysis to determine an upper bound on the communication delay [14], which helps to provide performance guarantees. (ii) When a time-critical AD/ADAS application sends data in parallel with a best-effort application, there is no guarantee that the former will be prioritized for RDMA communication. In particular, we have observed that Soft-RoCE does not have any notion of priority for communication scheduling. This means that a critical data packet might be delayed by a non-deterministic amount of time while a non-critical data packet is transferred by Soft-RoCE, which is again not desirable.

In Section IV, we show experimental results supporting our claims on the aforementioned nondeterministic behavior

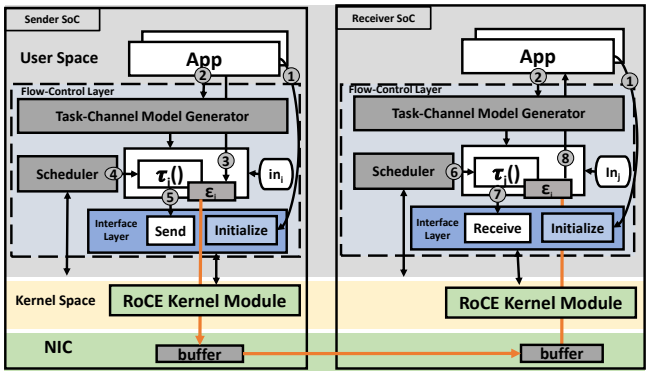


Fig. 7: Deterministic Soft-RoCE communication stack.

exhibited by Soft-RoCE.

III. RDMA-Based AD/ADAS Communication Stack

In this section, we describe the proposed multi-layer communication architecture and its initial implementation, as shown in Figure 7. In particular, we add two upper layers over a default Soft-RoCE implementation. (i) We add a *flow control* layer that manages RDMA operations performed by the applications. In particular, we can add different scheduling policies to send and receive data in this layer. (ii) The flow control layer uses easy-to-use APIs provided by the *interface layer* to carry out RDMA communication. This layer wraps the APIs in Libibverbs which is a user space library comprising 36 or more ibverbs (Infiniband verbs) to interact with the Soft-RoCE kernel module. This layer hides the complex details of ibverbs from application/middleware developers for sending and receiving data. Also, it allows us to replace the implementation of the flow control layer very easily. Note that application developers can directly use the APIs provided by this layer without using the flow control layer if they want to bypass the scheduler.

A. Interface layer

This layer comprises a user space C++ library that provides three simple APIs, namely *Initialize*, *Send*, and *Recieve*.

1) *Initialize* API is invoked to set up an RDMA connection. It initializes the state of different RDMA components, e.g., QP and CQ states. It registers the memory to be used for communication. It defines the connection parameters (e.g., local and remote IP addresses) to identify the remote side and establish a connection with it. The arguments to this API are (i) a pointer to the memory area to be registered and (ii) the IP address of the remote SoC with the associated port number. Note that *Initialize* API has to be invoked on both sending and receiving sides.
2) *Send* API is invoked on an SoC to send a data. It creates and posts a work request on the SQ of the QP. it defines an error object using which details of the error can be propagated to the application level on an unsuccessful completion of the work request. The arguments to *Send* API are (i) the IP address of the receiver SoC and (ii) the size of and the pointer to the data to be transferred.
3) For a successful communication, the receiver SoC also needs to invoke the *Receive* API to receive the data. Similar to the *Send* API, (a) it creates and posts a work request on the SQ of the QP and (b) defines an error object to notify the application of any error in the communication. The arguments to *Receive* API are (i) the IP address of the sender SoC, (ii) the size of the data to be received and (iii) the pointer to the memory where the data will be stored.

The aforementioned APIs help the application developer to integrate Soft-RoCE in their application without the need to be acquainted with the ibverbs and the associated challenges to use them appropriately for efficient communication, e.g., maintaining QP states, creating work requests, and defining memory protection domains. Each API in this layer encapsulates several ibverbs[1] and provides simple interfaces that any application developer can interpret correctly, e.g., pointers to data, data sizes, and IP addresses.

B. Flow control layer

The main function of this layer is to schedule work requests across different RDMA connections according to a desired policy. To realize this functionality, we have used Tasking Framework [15] which is an open-source multi-threading platform based on the Task/Channel model introduced in [16]. It provides abstract classes with virtual methods to create applications as directed acyclic graphs (DAGs) of *tasks* (or functionalities) and *channels* (or message queues between communicating tasks).

In the context of RDMA communication, we use (i) channels to implement RDMA buffers and (ii) tasks to implement RDMA send and receive operations. Further, the activation of a task (τ_i) can be controlled by configuring the task input(s) (in_i). The flow control layer offers *time-driven* and *event-driven* activation of tasks. In a time-driven activation, the task invoking an RDMA send/receive operation is triggered by a periodical signal generated a timer. Hence, we can control the rate at which data is sent/received irrespective of the rate at which data is produced. On the other hand, RDMA operation can also be carried out (or the corresponding task can be activated) in response to an event, e.g., data is pushed into the channel by the application. Using such an event-driven activation, data can also be sent/received sporadically. Both time- and event-driven activation are useful in AD/ADAS applications. Also, when multiple inputs are defined for a task, they can be combined, e.g., by *AND* or *OR* operation, to activate the task. That is, a task can be triggered when either of the input signals is available or it can be triggered only when all input signals have arrived. The above task activation options in the flow control layer allow our proposed stack to be used for applications with different timing characteristics and requirements.

[1] ibverb details ommitted for the sake of readability and space constraint

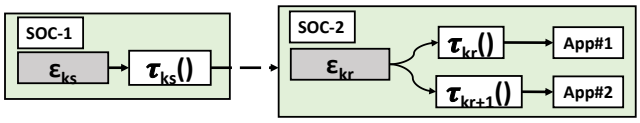


Fig. 8: One channel feeding data to multiple consumer tasks.

Using our proposed stack, a number of threads are created to carry out different RDMA operations. These threads can be scheduled according to a policy. In the flow control layer, we have implemented a static fixed-priority scheduler. Note that when tasks have equal priority, the scheduler dispatches them using a FIFO policy. Unlike the default implementation of Soft-RoCE, using a real-time scheduling policy (as described above) our proposed stack can guarantee (i) in-order packet delivery for applications with equal priority and (ii) lower worst-case and average latency for high-priority data packets.

Figure 7 shows how (i) applications interact with the flow control layer and (ii) the flow control layer uses the APIs in the interface layer. After an RDMA connection is established using the *initialize* API of the interface layer (1), the application have to invoke the task/channel generator of the flow control layer both at sending and receiving sides to define theirs channel(s) (ϵ) and task(s) (τ) (2). Now when the data to be sent is available at the sending application, it is directly pushed to its associated channel using the push() functionality that is implemented in the flow control layer (3). push() starts a chain of function calls: activate() and queue() that queues the task in the ready queue of the priority-based scheduler (4). Once the sender task is scheduled for execution, the scheduler calls the perform() function which calls the *Send* API of the interface layer (5). Simultaneously, on the receiver side, the RDMA operations are managed in the same way except that the receiver task is activated and queued by the scheduler with an empty channel (6). When the receiver task is scheduled for execution, the scheduler calls the perform() function that calls the *Receive* API of the interface layer (7). If the receive operation is successful, the data is then pulled by the application, and the operation is completed (8).

C. Real-time extensions

1) Multi-rate DAGs: In automotive systems, we can find applications that comprise tasks running at different frequencies. Such applications are often modeled using multi-rate DAGs [17]. In such DAGs, we can easily find a producer task that feeds data to two or more consumer tasks that run with different frequencies. Consider an example where frames captured by the camera are usually available at 30 Hz. They can be fed to pedestrian detection tasks also at 30 Hz. At the same time, for lane departure detection, they can be fed at 100 Hz. Now, when the consumer tasks are in the same SoC, multiple memory regions are pinned typically to receive the same data at different rates. However, our communication stack allows us to implement a set of tasks $\{\tau_K, \tau_{K+1}, .., \tau_{Kn}\}$ that consume data from the same channel ϵ_{cam} without introducing additional memory copies. Hence, with our proposed communication stack, it is very easy to implement realistic distributed automotive systems where applications run at different rates.

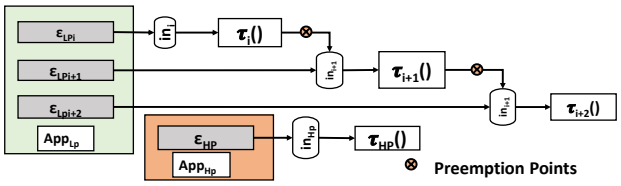


Fig. 9: Fixed-point preemption in an RDMA communication.

The conventional RDMA operation to serve these flows will require the user to pin another buffer on SOC-1 and copy the data between the buffers before creating parallel flows of data. However, as shown in Figure 8, we can use our stack, to implement a set of tasks on the reciever side $\{\tau_{Kr}, \tau_{Kr+1}, .., \tau_{Krn}\}$ that consume the same data channel ϵ_{ks} on the sender side without introducing extra copy in the driving functionality flow. through the task-channel model, our proposal can support DAGs that have different firing conditions and even different priorities in order to meet high-level driving application end-to-end timing requirements.

2) Fixed-point preemption: For mixed-criticality automotive applications sharing the same physical network link for communication, we may encounter a case where a best-effort application is blocking the transmission of data packets of a time-sensitive application. That is, a large amount of data may be sent by a best-effort application while a time-sensitive application waits to send its data. This is because Tasking Framework does not support preemption. Also, such a scenario will lead to a substantially longer communication latency for a critical application which might cause unacceptable performance degradation.

To address this problem, we introduced a preemption feature in RDMA communication. This preemption policy is particularly useful where multiple applications use the same RDMA port to send/receive data and have different priorities. In essence, our flow control layer supports fixed-point preemption [18], i.e., scheduling decisions can be taken at one or more fixed points while sending the whole data. To allow such a fixed-point preemption, we use a multi-channel and multi-task implementation for sending low-priority and large-sized data. In this implementation, τ_{LP}—the task implementing the RDMA sending functionality for a low-priority appplication—is split into a set of tasks $\tau_{LPi}, \tau_{LPi+1}, .., \tau_{LPn}\}$. Each task takes data from one channel and there is a fixed set of channels $\{\epsilon_{LPi}, \epsilon_{LPi+1}, .., \epsilon_{LPn}\}$ instead of one channel ϵ_{LP}. Figure 9 shows an example of how we can implement preemption-enabled RDMA communication. Here, the sum of the sizes of these partitioned channels is equal to the size of the preemption-unaware channel that can hold the whole data to be sent. Each of these tasks are non-preemptively scheduled in a certain order given by the design. At each preemption point, the scheduler checks the availability of other high-priority RDMA functionalities (or communication tasks), reducing its blocking time due to low-priority RDMA functionalities. Due

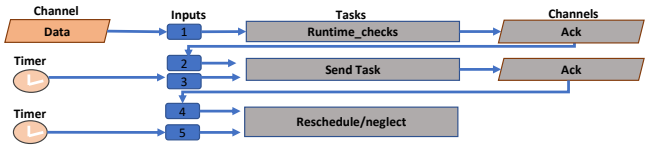


Fig. 10: Sending data via the flow control layer.

to the fixed preemption points, τ_{HP} (in Figure 9) can be executed before τ_0, after τ_0 or after τ_1 based on when it is ready to run. This ensures reduced and deterministic latency for high-priority RDMA communication. For such a preemption policy one can apply worst-case latency analysis similar to the worst-case response time analysis known in real-time system literature [18]. To apply such a technique to estimate the communication latency is a future work.

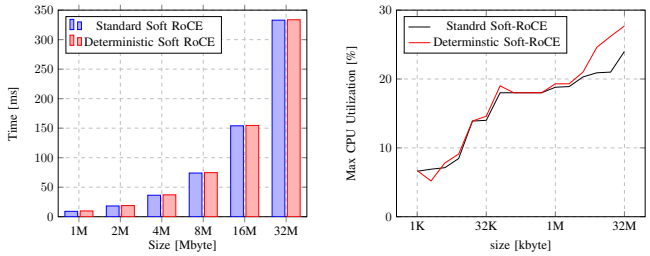
D. Support for more QoS metrics

Figure 10 illustrates how a more robust RDMA send operation can be carried out using our proposed stack. Here, an RDMA send operation involves three steps as follows: (i) to perform runtime checks before starting data transfer, (ii) to send data (i.e., calling the *Send* API in the interface layer), and (iii) to receive the acknowledgment for successful completion or reschedule the operation if acknowledge is not received.

Once data is pushed by the application to the channel on the sending side, it can trigger a task to perform runtime checks. These checks can be related to different QoS requirements, e.g., data freshness or security-related. However, the more checks we perform, the more the communication overhead, i.e., the communication latency will increase. Once these checks are finished, a task is triggered to send data. This task runs based on the scheduling policy configured in the flow control layer. Note that we can also configure this task to wait for a timer signal activated periodically. Such a periodic activation will allow us to implement time-triggered communication over Soft-RoCE, which may be desirable in many safety-critical applications. Further, as mentioned in Section II-A, once an RDMA communication is completed successfully, there is a WQE in the CQ. We can also use this information from Soft-RoCE layer to activate a task in the flow control layer. Simultaneously, we can configure the same task to get triggered by a timer signal generated after a pre-configured amount of time from the activation of the data send task. If the task is first triggered because of the notification for successful completion, then the timer is inactivated. However, if the task is triggered by the timer signal then the whole flow can be restarted or a warning can be generated based on the design requirements.

IV. Evaluation

Hardware setup: We emulate a MPSoC platform with internal RDMA connections by creating a two-node setup where each node resembles an SoC. Each node comprises a x86_64 Intel CPU and a RTX 2080 NVIDIA GPU. Both of them are equipped with an RTL8111 NIC and are connected to the same



(a) Communication latency (b) Maximum CPU Utilization

Fig. 11: Overheads of Deterministic Soft-RoCE compared to Standard Soft-RoCE

8-port Gigabit Ethernet switch. Linux 5.13.0-51-generic kernel runs on both nodes.

Different communication stacks: To transfer data between the two computation nodes, we use three different communication stacks. (i) *TCP*: We can use the default Ethernet protocol stack with TCP/IP. (ii) *Standard Soft-RoCE*: We can use the native Soft-RoCE implementation extended with our simple-to-use APIs (for initialization, handshake, send and receive). (iii) *Deterministic Soft-RoCE*: We can use the full multi-layer communication stack (including the scheduler layer) that we have implemented. While we have compared (i) and (ii) in Section II-C, in this section, we mainly compare (ii) and (iii).

Communication overheads: We assess the overheads added by Deterministic Soft-RoCE compared to Standard Soft-RoCE in terms of communication latency and maximum CPU utilization. We measure the overheads for sending data packets of varying sizes from one node to another. (i) We have observed that the increase in the communication latency stays constant around 670 us. We have seen a similar overhead for smaller packets where this increase is significant, e.g., Deterministic Soft-RoCE increases the latency of sending 64 KB data by approximately 100%. Nevertheless, as shown in Figure 11a, for larger packets with more than 1 MB data, the overhead is less than 7.5%. Later, in this section, we will see that this is an acceptable cost to pay for sending large-sized safety-critical data in AD/ADAS applications considering that Deterministic Soft-RoCE will reduce the worst-case communication latency significantly in the presence of interfering best-effort data packets. (ii) We have measured the maximum CPU utilization on the sender side while sending each data packet. For each data size, we have considered 150 different data transfers and have noted the maximum CPU utilization among them. Figure 11b shows the maximum CPU utilization for sending data of different sizes both with Deterministic Soft-RoCE and Standard Soft-RoCE. It is clear that with Deterministic Soft-RoCE CPU utilization is higher, i.e., more computation is required by the CPU. For data size up to 4 MB, the difference in the CPU utilization is less than 1% in absolute value, which is negligible. For data size more than 4 MB, we see around 5 – 6% absolute increase in CPU utilization. Nevertheless, the CPU utilization is still significantly lower than when using the standard TCP stack, as observed in Figure 6.

Packet order: We have created three tasks on one node that

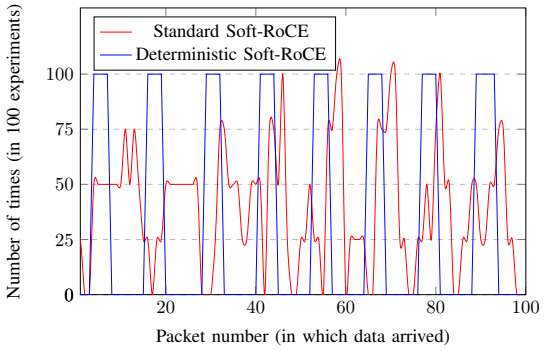


Fig. 12: Packets delivered in order with Deterministic Soft-RoCE and out of order with Standard Soft-RoCE.

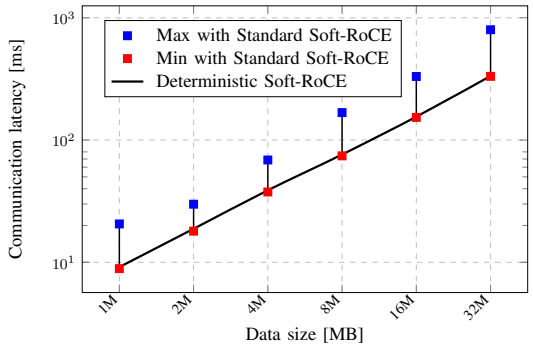


Fig. 13: Latency for a high-priority data packet.

are sending data in a round-robin fashion. In each turn, a task sends 4 data packets consecutively. For each sender task, there is a task on another node receiving the data. We repeat the experiment 100 times. Let us assume here that the data transfers have equal priority, i.e., a high priority. For such an assumption, we expect the data to be sent as per the first-in-first-out (FIFO) policy. We use Wireshark [19] to observe when packets are transferred on the network for one task and note down the order. Figure 12 shows the number of times (in 100 experiments) data sent by the observed task appeared on the network as the $n-$th packet, where $0 \leq n \leq 100$ (we only consider the first 100 packets in each experiment). With Deterministic Soft-RoCE, we have seen that the packets always appear in the network in the same order as they are sent. This is shown by the solid blue line in the figure where 4 consecutive packets are sent at a regular interval by the task under study across all experiments. The line drops to 0 when packets from other tasks are being sent. Conversely, with Standard Soft-RoCE, packets appear on the network out of order, as shown in the figure. Overall with Standard Soft-RoCE, for the observed task, only 30% of the packets are received in the same order as they are sent. This nondeterministic behavior of Standard Soft-RoCE prevents it to be used for time-sensitive applications because a packet can be delayed for an arbitrary amount of time, especially when multiple tasks are communicating over the same network link.

Communication latency: In this experiment, we have three tasks sending data with different priorities (i.e., high, medium and low) on one node. On the receiving side, we have a consumer task for each data. We vary the data size exponentially from 1 MB to 32 MB and carry out 100 runs for each data size. In Figure 13, we show the when Standard Soft-RoCE is used for the data transfers, we see that there is a large variation in the communication latency of the high-priority packet. The maximum latency can easily be more than two times of the minimum latency, e.g., for 1 MB, we observe a maximum latency of 20.6 ms and a minimum latency of 8.9 ms. This clearly shows that Standard Soft-RoCE is not suitable for sending latency-sensitive (or high-priority) data packets. Conversely, Deterministic Soft-RoCE respects the QoS requirement of a packet (e.g., priority in this case). Hence, in Figure 13, we can see that the communication latency of the high-priority packet remains constant for each data size when Deterministic Soft-RoCE is used. Also, the communication latency for the high-priority packet is nearly equal to the minimum latency observed when Standard Soft-RoCE is used. These results show that our proposed communication stack has the potential to be used for AD/ADAS applications with firm real-time requirements.

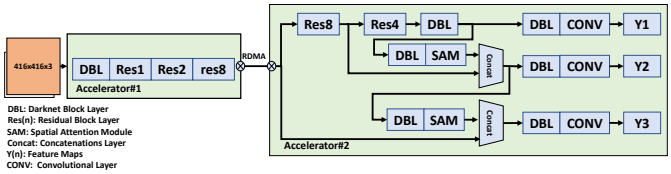


Fig. 14: Partitioned YOLO representing a distributed AD/ADAS application where each partition can run on a different GPU.

Distributed automotive application: In this experiment, we want to quantitatively assess the benefits of using Soft-RoCE towards reducing the end-to-end latency of AD/ADAS applications. In this context, the most important challenge is that, to the best of our knowledge, there is no available automotive benchmark that runs on accelerators distributed over multiple SoCs. Nevertheless, it is not difficult to imagine that such implementations will be common in autonomous vehicles where, for example, object detection and tracking is followed by motion planning that is further followed by vehicle control and each of these algorithms can be accelerated using specialized processors, e.g., [20]. Hence, we have developed a benchmark to resemble an AD/ADAS application, which is also an engineering contribution of this paper.

We started with an object detection algorithm based on YOLO [8], the state-of-the-art family of DNN architectures and models, pre-trained on the COCO dataset [21]. The DNN has 106 layers with fully convolutional underlying architecture and provides a very high accuracy in object detection. Considering that we were looking for at least two neural networks (NNs) where the result of one is fed into the second, we partitioned YOLO into two NNs. Here, we have modified YOLO so that it can be easily partitioned through parametrization. It is also possible to obtain more than two NNs from YOLO. An example partitioned YOLO is shown in Figure 14 where the input of the first NN is a scaled image

(or a camera frame). Note that splitting DNNs for accelerated training is known [22] and RDMA can even be used in such setups to improve the training throughput. However, here, we have used partitioned YOLO for inference.

We run two NNs obtained from YOLO in two different GPUs attached to different nodes. The output of the first NN will be used as input by the second NN which we transfer using Standard Soft-RoCE. For the partitions shown in Figure 14, 5.64 MB data is transferred using Standard Soft-RoCE. We measure the computation and communication time for object detection using distributed YOLO. The computation times in the two nodes add up to 100 ms and the communication latency is 50 ms, i.e., the end-to-end latency for object detection is 150 ms. Further, we transfer the same amount of data using TCP where we get a communication latency of 65 ms. Thus, with TCP, we can say that the end-to-end latency of object detection is 165 ms. That is, with Standard Soft-RoCE, we can reduce the end-to-end latency by 9.1%. If we consider Deterministic Soft-RoCE that has an overhead of 0.67 ms, the reduction in the end-to-end latency is 8.7%. These results emphasizes the fact that reduction in the communication latency can improve the end-to-end latency of AD/ADAS applications significantly. Note that here we have not considered the impact of Deterministic Soft-RoCE in reducing the worst-case latency in comparison to TCP and Standard Soft-RoCE.

V. Related Works

RDMA technology has been widely adopted in data centers over Ethernet networks (i.e., RoCE) to tackle the challenges in data-intensive applications, e.g., big data analytics and online gaming [23] [24] [25]. Several works have evaluated the performance of RoCE in comparison to default Ethernet communication stacks with TCP/IP, e.g., [26], [27]. These works clearly show that RoCE reduces the CPU load (related to network communication) significantly compared to TCP and, at the same time, offers lower communication latency. RoCE communication has been traditionally established over RDMA-capable NICs, i.e., the main focus has been to build such hardware, e.g., [28]. Although Soft-RoCE has been developed to enable hardware-independent RDMA communication, it has not received much attention for industrial use. Considering that Soft-RoCE offers more flexibility and similar performance with respect to hardware-based RoCE [6], we advocate its use in the automotive domain.

The primary research focus in the context of RoCE has been to control packet congestion in the network switches that can lead to packet losses. To avoid packet loss, the default technique in RDMA is to use Go-Back-N protocol where only the packets following and including the lost packet are re-sent [29]. However, this protocol suffers from lower throughput and, hence, is improved to IRN where only the lost packet is re-transmitted, thereby allowing out-of-order packet delivery [30], [31]. Further, to avoid buffer overflows in network switches and NICs, priority flow control (PFC) mechanisms have been proposed [32], where sender devices are notified hop-by-hop to pause/resume sending packets based on the states of the buffers in the receiver devices. PFC is typically implemented at the port level which might lead to poor performance, e.g., unfairness and victim flow, with respect to individual data flows [33], [34]. For flow-level congestion control, quantized congestion notification (QCN) is proposed where a data flow is addressed by its identifier in addition to the MAC address [35]. Data center QCN (DCQCN) is also proposed to ensure fairness in bandwidth allocation [33]. Different algorithms are proposed for DCQCN, e.g., [36]. Note that the problem setting in these works is very different from ours. In a typical automotive setting with AD/ADAS applications, the static analysis shall be performed to estimate buffer size so that time-critical packets are not lost [37]. Our goal is to minimize the latency of high-priority data packets and, at the same time, it is acceptable to be unfair to best-effort packets. Note that these works assume that packets are sent based on the default RDMA protocol unless there is congestion, however, we want to prioritize packets even when there is no congestion and the sending side already knows the priority of the packets. These existing techniques cannot be directly applied to solve the problem at hand and they are mostly implemented on hardware. [38] is the closest to our work where a hardware priority queue is proposed in the Queue Pair. However, our communication architecture is more flexible as we can easily add different scheduling mechanisms and QoS metrics to the software.

Full-duplex switched Ethernet [39] and BroadR-Reach [40] standards have enabled the use of Ethernet in safety-critical automotive systems. Express traffic or high-priority traffic is recommended in IEEE 802.3br [41], an amendment to Ethernet protocol. Further, the Time-Sensitive Networking (TSN) Task Group has proposed several amendments (e.g., time synchronization [42], bandwidth reservation [43], and queueing and forward of time-sensitive frames [44]) to the Ethernet protocol to support time-sensitive communication. However, to the best of our knowledge, none of the implemented TSN protocol stack supports RDMA communication. This work is the first step to carry out RDMA communication for time-sensitive automotive applications. Our proposed communication stack is developed keeping in mind a possible future extension to integrate TSN with RDMA.

In the automotive domain, to the best of our knowledge, RDMA technology has only been used in Mobile Data lake (MDLake) to collect vehicle data from all loggers [45]. Again, hardware-enabled RoCE_v2 is used for high-bandwidth communication.However, we propose to use Soft-RoCE for inter-SoC communication in distributed AD/ADAS applications. We have performed real-world experiments to evaluate the performance of our proposed communication stack involving Soft-RoCE in terms of latency and determinism which are critical requirements of AD/ADAS applications.

VI. Conclusion

In this paper, we propose to use Soft-RoCE for AD/ADAS applications. We have shown that it can reduce communication

latency significantly compared to a default Ethernet communication stack over TCP. Further, we have demonstrated non-deterministic timing behavior in communication over standard Soft-RoCE. At the same time, we have developed a multi-layer communication stack and we experimentally show that using the stack, we can carry out deterministic and low-latency transfer of high-priority data. In the future, we would like to extend the stack with support for TSN communication, in particular, different traffic classes. We want to provide easy interfaces for Data Distribution Service (DDS) over Soft-RoCE. Also, we can implement the proposed stacks on different SoCs and validate its real-time performance.

REFERENCES

[1] B. Wolfe, B. Seppelt, B. Mehler, B. Reimer, and R. Rosenholtz, "Rapid holistic perception and evasion of road hazards," *Journal of Experimental Psychology: General*, vol. 149, 2020.

[2] M. Bui, L. Chang, H. Liu, Q. Zhao, and G. Chen, "Comparative study of 3d point cloud compression methods," in *IEEE International Conference on Big Data (Big Data)*, 2021.

[3] X. Han, J. Lu, Y. Tai, and C. Zhao, "A real-time LIDAR and vision based pedestrian detection system for unmanned ground vehicles," in *Asian Conference on Pattern Recognition (ACPR)*, 2015.

[4] M. Silberstein, "OmniX: An Accelerator-Centric OS for Omni-Programmable Systems," in *Workshop on Hot Topics in Operating Systems (HotOS)*, 2017.

[5] R. Recio *et al.*, "RFC 5040: A remote direct memory access protocol specification," *Internet Standards (IETF)*, 2007.

[6] The RoCE Initiative of the InfiniBand Trade Association (IBTA), "Soft-RoCE: RDMA transport in a software implementation," 2015.

[7] G. Kaur, M. Kumar, and M. Bala, "Performance Evaluation of Soft-RoCE over 1 Gigabit Ethernet," *IOSR Journal of Computer Engineering*, vol. 15, 2013.

[8] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv*, 2018.

[9] "NividaDocs." [Online]. Available: https://docs.nvidia.com

[10] InfiniBand Trade Association *et al.*, "Supplement to InfiniBand Architecture Specification 1.2.1 Annex A16," 2010.

[11] InfiniBand Trade Association *et al.*, "Supplement to InfiniBand Architecture Specification 1.2.1 Annex A17," 2014.

[12] "qperf." [Online]. Available: https://linux.die.net/man/1/qperf

[13] T. Ziegler, V. Leis, and C. Binnig, "RDMA communciation patterns," *Datenbank-Spektrum*, vol. 20, 2020.

[14] G. C. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*. Springer, 2011.

[15] Z. A. Haj Hammadeh, T. Franz, O. Maibaum, A. Gerndt, and D. Lüdtke, "Event-driven multithreading execution platform for real-time on-board software systems," in *Proceedings of the 15th OSPERT*, 2019.

[16] I. Foster, *Designing and building parallel programs: Concepts and tools for parallel software engineering*. Addison-Wesley Longman Publishing Co., Inc., 1995.

[17] M. Verucchi, M. Theile, M. Caccamo, and M. Bertogna, "Latency-aware generation of single-rate DAGs from multi-rate task sets," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2020.

[18] R. J. Bril, J. J. Lukkien, and W. F. Verhaegh, "Worst-case response time analysis of real-time tasks under fixed-priority scheduling with deferred preemption," *Real-Time Systems*, vol. 42, 2009.

[19] "wireshark." [Online]. Available: https://www.wireshark.org/

[20] Y. Li, S. E. Li, X. Jia, S. Zeng, and Y. Wang, "FPGA accelerated model predictive control for autonomous driving," *Journal of Intelligent and Connected Vehicles*, 2022.

[21] T.-Y. Lin *et al.*, "Microsoft coco: Common objects in context," in *European conference on computer vision*, 2014.

[22] Y. Huang, Y. Cheng, A. Bapna, O. Firat, M. X. Chen, D. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu, and Z. Chen, "GPipe: Efficient training of giant neural networks Using pipeline parallelism," in *International Conference on Neural Information Processing Systems*, 2019.

[23] C. Guo, "RDMA in data centers: Looking back and looking forward," *Keynote at APNet*, 2017.

[24] M. Beck and M. Kagan, "Performance evaluation of the RDMA over ethernet (RoCE) standard in enterprise data centers infrastructure," in *Workshop on Data Center-Converged and Virtual Ethernet Switching*, 2011.

[25] T. Hoefler *et al.*, "Datacenter Ethernet and RDMA: Issues at Hyperscale," *arXiv preprint arXiv:2302.03337*, 2023.

[26] Y. Wan, D. Feng, F. Wang, L. Ming, and Y. Xie, "An In-Depth Analysis of TCP and RDMA Performance on Modern Server Platform," in *IEEE International Conference on Networking, Architecture, and Storage*, 2012.

[27] P. Balaji, H. V. Shah, and D. K. Panda, "Sockets vs RDMA interface over 10-gigabit networks: An in-depth analysis of the memory traffic bottleneck," in *Workshop on RDMA: Applications, Implementations, and Technologies (RAIT)*, 2004.

[28] Y. Yuan, J. Huang, Y. Sun, T. Wang, J. Nelson, D. R. K. Ports, Y. Wang, R. Wang, C. Tai, and N. S. Kim, "Rambda: Rdma-driven acceleration framework for memory-intensive µs-scale datacenter applications," in *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2023.

[29] Y. Wang, K. Liu, C. Tian, B. Bai, and G. Zhang, "Error Recovery of RDMA Packets in Data Center Networks," in *International Conference on Computer Communication and Networks (ICCCN)*, 2019.

[30] R. Mittal, A. Shpiner, A. Panda, E. Zahavi, A. Krishnamurthy, S. Ratnasamy, and S. Shenker, "Revisiting network support for RDMA," in *ACM Special Interest Group on Data Communication*, 2018.

[31] Q. Meng and F. Ren, "Lightning: A Practical Building Block for RDMA Transport Control," in *IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*, 2021.

[32] I. S. Association, "IEEE Standard for Local and metropolitan area networks–Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks–Amendment 17: Priority-based Flow Control (IEEE. 802.11Qbb," 2011.

[33] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel, M. H. Yahia, and M. Zhang, "Congestion control for large-scale rdma deployments," *ACM SIGCOMM Computer Communication Review*, vol. 45, 2015.

[34] C. Guo, H. Wu, Z. Deng, G. Soni, J. Ye, J. Padhye, and M. Lipshteyn, "RDMA over commodity Ethernet at scale," in *ACM SIGCOMM Conference*, 2016.

[35] I. S. Association, "IEEE Standard for Local and Metropolitan Area Networks– Virtual Bridged Local Area Networks Amendment 13: Congestion Notification (IEEE. 802.1Qau," 2010.

[36] T. Wang, H. Kan, Q. Sun, S. Xiao, and S. Wang, "Congestion detection and link control via feedback in RDMA transmission," in *IEEE International Conference on Service Science (ICSS)*, 2022.

[37] M. Benazouz and A. Munier-Kordon, "Cyclo-Static DataFlow Phases Scheduling Optimization for Buffer Sizes Minimization," in *ACM International Workshop on Software and Compilers for Embedded Systems*, 2013.

[38] L. Rosa, W. Song, L. Foschini, A. Corradi, and K. Birman, "DerechoDDS: Strongly consistent data distribution for mission-critical applications," in *IEEE Military Communications Conference*, 2021.

[39] Aeronautical Radio, Incorporated, "ARINC 664 P7 – AIRCRAFT DATA NETWORK PART 7 AVIONICS FULL-DUPLEX SWITCHED ETHERNET NETWORK," 2009.

[40] Broadcom Corporation, "BroadR-Reach R Physical Layer Transceiver Specification for Automotive Applications V3.0," 2014.

[41] IEEE Standards Association, "IEEE Standard for Ethernet Amendment 5: Specification and Management Parameters for Interspersing Express Traffic (IEEE 802.3br-2016)," 2016.

[42] IEEE Standards Association, "IEEE Standard for Local and Metropolitan Area Networks – Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks (IEEE 802.1AS2010)," 2010.

[43] IEEE Standards Association, "IEEE Standard for Local and Metropolitan Area Networks - Virtual Bridged Local Area Networks - Amendment: 9: Stream Reservation Protocol (IEEE 802.1Qat-2010)," 2010.

[44] IEEE Standards Association, "IEEE Standard for Local and Metropolitan Area Networks—Virtual Bridged Local Area Networks - Amendment: Forwarding and Queuing Enhancements for Time-Sensitive Streams (IEEE 802.1Qav-2009)," 2009.

[45] b-plus Group, "MDLake 100G data sheet," 2022. [Online]. Available: https://www.b-plus.com/fileadmin/data_storage/100G_EN_v1.0.pdf