RDMA-based Sampling Port of ARINC-653

Jong-Bin Lee, Sang-Jae Kim, Wook-Hee Kim, and Hyun-Wook Jin, *Member, IEEE*

***Abstract*—ARINC-653, a standard of avionics software platform, defines the sampling communication port that provides only the latest message while discarding old messages. This sampling port is particularly efficient at transmitting sensing data that reflects the actual state of target system without message queueing delay. In this paper, we implement the ARINC-653 sampling port by exploiting Remote Direct Memory Access (RDMA) over Ethernet that can directly move data to/from remote memory without CPU intervention on the remote node. We propose two different designs to utilize two-sided RDMA and one-sided RDMA operations, respectively. Performance measurement results show that the sampling port over one-sided RDMA provides lower communication latency, stronger temporal partitioning, and better message timeliness than two-sided RDMA and Berkeley sockets based implementations.**

***Index Terms*—ARINC-653, Ethernet, RDMA, sampling port**

I. Introduction

TRANSMITTING sensing data in real-time is critical for precise control of networked embedded systems, such as aerial/ground vehicles and robots, where few to many sensing and control units are connected through embedded networks. Embedded software platforms thus define a messaging mechanism efficient at transmitting sensing data. For instance, ARINC-653 [1], a standard of avionics software platform, defines the sampling communication port that provides only the latest message while discarding old messages. This is useful when a sensor unit sends sensing data to a controller because most controllers are interested in the current situation rather than the past. In addition, ROS2 [2], a robotics software platform, also defines the history QoS policy that can be configured to store only n latest messages.

Emerging networked embedded systems are actively adopting Ethernet technologies to benefit from high bandwidth and enhance real-time performance. For example, Avionics Full-Duplex Switched Ethernet (AFDX) [3] and Audio Video Bridging (AVB) [4] are Ethernet-based networks for aircraft and automobiles, respectively. Although several enhanced features provided by Ethernet have been considered for providing real-time data transmission, there are still useful Ethernet-based features that have not yet been considered in modern embedded systems. Remote Direct Memory Access (RDMA) over Ethernet [5], [6], for example, can directly move data to/from remote memory without CPU intervention on the remote node. There have been significant studies to utilize RDMA for server systems in terms of messaging [7] or data sharing [8], [9], but these cannot be simply applied to embedded systems due to special messaging semantics defined in embedded software platforms.

In this paper, we implement the ARINC-653 sampling port over RDMA standards, such as iWARP [5] and RoCE [6]. Since RDMA over Ethernet defines two-sided (send/recv) and one-sided (write/read) operations, we propose two different designs to utilize both of these. The ARINC-653 sampling port is different from traditional communication endpoints in that its message is shared between processes in the same partition, only the last message is valid, and the message is not consumed by a receiving call. It is challenging to efficiently implement this distinctive communication semantics over RDMA. To tackle this issue, we implement an event-driven network manager for two-sided operations and a synchronization mechanism on messages of arbitrary length for one-sided operations. Existing studies on communications of the ARINC-653 standard did not consider RDMA [10]–[12]. Performance measurement results show that the sampling port over RDMA reduces communication latency and provides stronger temporal partitioning.

II. Background

A. ARINC-653 Inter-Partition Communication

The ARINC-653 standard defines APIs called APplication EXecutive (APEX) that interface between the OS of an avionics computer and the application software [1]. The standard specifies temporal and spatial partitioning that enables the avionics applications to execute independently from each other in terms of CPU and memory resources. This partitioning concept is a key for the Integrated Modular Avionics (IMA) architecture as it provides the resource isolation between applications (i.e., partitions) by controlling utilization timing of CPU resources and by limiting the usage of memory resources. A partition comprises one or more processes that share the resources of the partition but are not visible outside of the partition. The CPU scheduling of partitions is repetitive with a fixed periodicity. The process scheduling can be either periodic or aperiodic.

ARINC-653 also specifies inter-partition communication interfaces. The inter-partition communication channel is a logical link between two or more partitions that may run on different devices. A channel consists of one or more ports. Processes running in the same partition share the communication port. A port can operate in either queueing or sampling mode. In the queueing port, the messages are queued in the internal message queue and passed to processes in the FIFO manner without intentional message loss. On the other hand, in the sampling port, only the last message is saved, overwriting the old messages. The APEX interfaces to send and receive a message through a sampling port are `Write_Sampling_Message` and `Read_Sampling_Message`, respectively.

J.-B. Lee, S.-J. Kim, and W.-H. Kim are with the Department of Computer Science & Engineering, Konkuk University, Seoul 05029, South Korea.

H.-W. Jin is with the Department of Computer Science & Engineering, Konkuk University, Seoul 05029, South Korea. e-mail: jinh@konkuk.ac.kr. (corresponding author)

Manuscript received - -, -; revised - -, -.

B. RDMA

RDMA directly moves messages from source to destination without (or with minimum) involving CPUs on the remote node thanks to protocol offloading. RDMA can provide high bandwidth and low latency and save CPU resources as it supports kernel bypassing and zero-copy data movements. RDMA supports two-sided (send/recv) and one-sided (write/read) operations. In two-sided operations, both sender and receiver should call *RDMA-send* and *RDMA-recv* interfaces, respectively, to make message passing. In one-sided operations, only the local task calls either *RDMA-write* that writes the data to the remote buffer or *RDMA-read* that moves the data in the remote buffer into the local buffer, while no corresponding call is invoked on the remote node. iWARP [5] and RoCE [6] are standards for RDMA over Ethernet defined by IETF and IBTA, respectively. iWARP implements RDMA operations over TCP/IP, while RoCEv2 works over UDP/IP.

III. RDMA-BASED SAMPLING PORT

This section describes our design of RDMA-based sampling port. We implement the proposed design in an existing user-level ARINC-653 implementation over Linux [13] by extending its configuration file format and inter-partition communication. Inside the ARINC-653 implementation, we invoke APIs provided by libibverbs [14] to exploit RDMA. This library provides common programming interfaces for iWARP and RoCE. Thus, our sampling ports can run on both standards.

A. Initialization

Communication channels and ports in ARINC-653 are defined statically by the configuration file, which specifies connections, name, maximum message size, direction, etc. We expand the configuration file format to add an option for the type of RDMA operation. If a port is set to use RDMA, we initialize RDMA over Ethernet and create an RDMA queue pair that consists of a send queue, a receive queue, and a completion queue and keep the mapping information between the port name and the RDMA queue pair.

An RDMA-based sampling port has either a TX or RX circular queue according to its direction. Buffers for RDMA operations have to be registered so that the memory region is not swapped out from physical memory and directly accessible from the network controller. Thus, we fill the TX and RX queues with pre-registered buffers during initialization. Since the sampling port delivers the last message to processes, there can be intentional loss for old messages. Details of TX/RX queue management will be described in following subsections.

B. Two-Sided based Sampling Port

The upper data path in Fig. 1 represents two-sided (i.e., RDMA-send and RDMA-recv) based sampling ports. When the sender process calls `Write_Sampling_Message`, the data is copied (T.1) to the rear of the TX queue (represented as `R` in Fig. 1). Although the user data can be directly sent with RDMA without intermediate data copy, the buffer registration for RDMA induces a significant overhead, which is larger

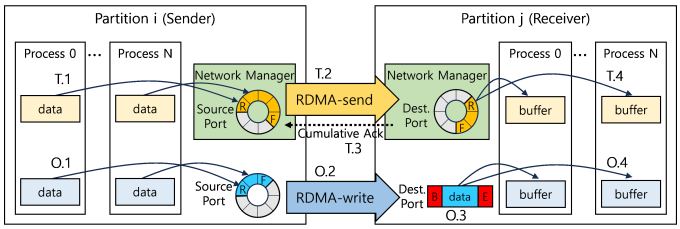


Fig. 1: Data path of RDMA-based ARINC-653 sampling port.

than the data copy overhead for small sensing data. Thus, as described in Sec. III-A, each sampling port manages a circular queue that has pre-registered buffers. The data copied into the TX queue is sent with RDMA-send in FIFO order starting from the front of the TX queue (`F` in Fig. 1) (T.2).

RDMA-recv has to be posted on the receiver side preceding RDMA-send. To guarantee this, the network manager performs flow control by exchanging cumulative acknowledgment (T.3). If there are no pre-posted buffers available on the receiver side, the sender does not post additional RDMA-send operations, but the data in subsequent requests that have not yet been posted for RDMA-send is overwritten by the latest request.

The data in the rear of the RX queue is copied to the destination buffer when a process calls `Read_Sampling_Message` (T.4). Since the sampling port provides the only latest message, we keep the data in the rear even after it is copied to the destination buffer and reuse it until a new message arrives. The buffers that store old data are reclaimed by the network manager. The network manager sends a cumulative acknowledgment that specifies the number of reclaimed buffers.

We create the network manager as an aperiodic process to handle messages without limitation in the quota allotted for each period. If it is periodic, the network latency can increase significantly depending on the period. At the same time, to prevent excessive usage of CPU resources, the network manager uses event-driven interfaces rather than polling interfaces.

C. One-Sided based Sampling Port

The lower data path in Fig. 1 represents one-sided (i.e., RDMA-write) based sampling ports. Like the two-sided based sampling ports, the data is copied to the TX queue by `Write_Sampling_Message` (O.1). The data copied into the TX queue is sent with RDMA-write (O.2). The port on the receiver side manages only a single pre-registered buffer, to which RDMA-write operations issued on the sender side move data. The data stored in this receive buffer is newest; thus, `Read_Sampling_Message` copies it to the destination buffer (O.4).

Since RDMA-write operations (O.2) and the copy operations in APEX (O.4) can perform concurrently on the same buffer, a race condition can occur. To avoid this, RDMA-write operations attach the *begin* and *end* flags (represented as red blocks in Fig. 1) that have the same value of 8 bytes. The value of these flags are generated by combining the message sequence number and the message size. The message sequence number is managed globally within

the partition and can be accessed synchronously between processes. `Read_Sampling_Message` returns only if the begin and end flags have the same value (O.3); otherwise, it retries the copy operation. As RDMA writes data in increasing address order [7], [9], if the begin and end flags are the same after the data copy, it is guaranteed that there has been no race condition. Although, this design may incur unbounded overhead with a very high transmission rate on the sender side, we do not need an additional daemon, such as network manager, and the flow control. We will analyze the impact of transmission rate on this synchronization scheme in Sec. IV-C.

As an alternative, we can consider using RDMA-read instead of RDMA-write. In this case, the source port manages a single pre-registered buffer, and the receiver reads it with RDMA-read. This design, however, incurs even higher retrial overhead if the begin and end flags are mismatched because remote memory access is required for every retrial. We can also consider using remote atomic operations, such as `FetchAdd` and `CmpSwap`. However, these operations require busy waiting on remote memory. Moreover, most of (if not all) iWARP implementations do not support atomic operations yet.

IV. Performance Evaluation

We measured the performance on two industrial embedded boards equipped with an Intel Core i9-10900 2.80 GHz CPU and an Intel E810-CQDA2 Ethernet network adapter. We installed Ubuntu 20.04. We compare our implementations (i.e., two-sided and one-sided RDMA based sampling ports described in Sec. III) with a traditional Berkeley sockets based implementation. The data path of the sockets based sampling port is similar to that of two-sided based one, but the sockets based implementation uses the Berkeley sockets interface and TCP/IP layers implemented in the Linux kernel.

A. Communication Latency

We analyzed the Round-Trip Time (RTT) between two industrial boards by measuring the time elapsed from sending the message to receiving the reply in a ping-pong fashion. Each industrial board ran a partition that reserved a whole CPU core and had an aperiodic communicating process. Since the sensor data is relatively small, we varied message size up to 4 KB.

As shown in Fig. 2(a). the one-sided RDMA based implementation shows lower round-trip times, up to 46% and 55%, compared with two-sided RDMA and Berkeley sockets based implementations, respectively. This is mainly because the one-sided RDMA based implementation does not generate network events, such as interrupts, on the remote side and can make a progress of communication regardless of peer's progress. Accordingly, it does not require the network manager. The Berkeley sockets based implementation shows worse latency than the others because it performs an additional data copy at the sockets layer and has a complicate bottom half called *net_rx_softirq* to handle network events.

Fig. 2(b) plots the empirical Cumulative Distribution Function (CDF) of RTT of 16-byte message. Overall, all three different implementations show relatively stable numbers. The one-sided RDMA based implementation reported the least worst-case RTT compared with the others.

B. Temporal Partitioning

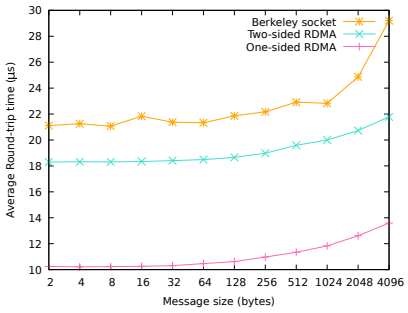
As we have described in Sec. II-A, providing temporal partitioning that guarantees CPU resources for each partition while preventing resource interference between partitions is a major function of ARINC-653. In order to see if the communication on a partition affects another partition running on the same CPU core, we measured the execution time of a computing partition that runs together with a communicating partition. The computing partition includes a periodic process that calculates a simple equation. The communicating partition receives messages via a sampling port from a remote partition that sends 4-byte messages with 1 ms or 10 ms interval. The network interrupt handler and bottom half also ran on the core running the communicating partition.

Fig. 3 shows CDFs for the execution time of the computing partition. *None* in these graphs represents the case where the computing partition runs without communicating partition; thus, the closer the measured value is to *None*, the more it means that there was no resource interference. We can observe that the one-sided RDMA based implementation almost does not induce resource interference because it does not require local CPU resources to place data received into the RX buffer. Therefore, it can provide stronger temporal partitioning than the others. Two-sided RDMA and Berkeley sockets based implementations, however, generate network events on the receiving side when messages arrive. The interrupt service routine is invoked in both implementations, and the corresponding bottom half is additionally performed. Since these event handling routines are performed with a high priority at the operating system level, the computing partition gets delayed even if the partition is not corresponding to the communication. The resource interference reduces as the sending interval increases (from 1 ms to 10 ms).

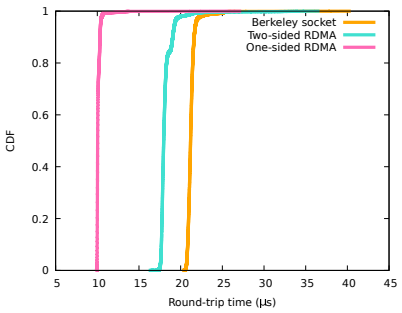
C. Message Timeliness

We also analyzed the end-to-end timeliness of periodic messages. In these experiments, the sender periodically transmitted messages that contained the timestamp through a sampling port, and the aperiodic receiver polled a new message arrival and reported differences between previous and new timestamps received. Thus, if the receiver obtained every message in time without overwriting/discarding messages in the sampling port, the interval of received timestamps at the receiving end should be equal to the transmission interval.

Fig. 4 shows actual transmission intervals on the sender and observed intervals on the receiver in the x and y axes, respectively (y-axis is a log scale). The Berkeley sockets based implementation shows significantly larger observed intervals than actual transmission intervals for small transmission intervals (less than 25 μs). That is, the Berkeley sockets based implementation fails to provide messages transmitted at small intervals to the application in a timely manner. In Linux, if IP packets arrive in a very high rate, packets are handled by the lower-priority daemon process called *ksoftirqd* instead of the higher-priority bottom half to avoid resource starvation of user processes. Thus, packet processing gets delayed, resulting in message overwriting in the sampling port. As shown in

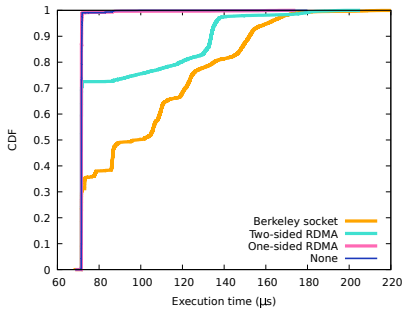


(a) Round-trip time

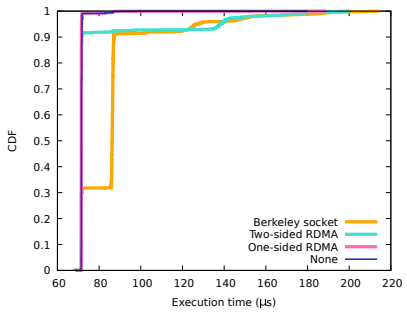


(b) CDF of RTT of 16-byte message

Fig. 2: Round-round-trip communication latency

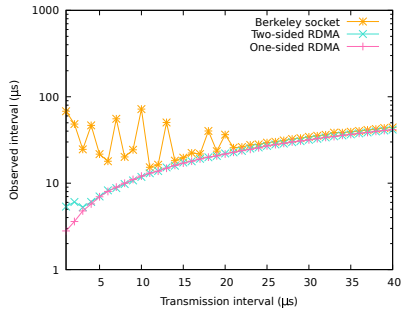


(a) 1 ms transmission interval

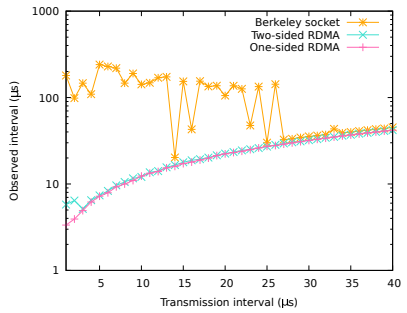


(b) 10 ms transmission interval

Fig. 3: Impact of background communication on foreground computation



(a) 16-byte message



(b) 4096-byte message

Fig. 4: Transmission intervals vs. observed intervals on the receiver

Fig. 4(b), a larger message size aggravates such message overwriting and increases observed intervals even more.

The one-sided RDMA based implementation showed almost the same observed intervals with actual transmission intervals for all experiments. This means that retried copy operations described in Sec. III-C hardly occur even for very high data transmission rate, and their overhead is not significant. The two-sided RDMA based implementation, however, reported about 6 times higher observed intervals than transmission intervals for very small transmission intervals (less than 3 μs). This is mainly due to network event handling performed by the interrupt service routine in the device driver and the bottom half (*tasklet_softirq* and the subsequent tasklet) in the kernel.

V. Conclusions and Future Work

In this paper, we proposed two-sided and one-sided RDMA based designs for the ARINC-653 sampling port. To the best of our knowledge, our study is the first to utilize RDMA for inter-partition communication of ARINC-653. Performance measurement results showed that the one-sided RDMA based implementation reduced the round-trip time, up to 46% and 55%, compared with two-sided RDMA and Berkeley sockets based implementations, respectively. In addition, we showed that the one-sided RDMA based implementation could provide stronger temporal partitioning and better message timeliness. As future work, we plan to implement an RDMA based queueing port of ARINC-653.

Acknowledgments

This research was supported by the Component Localization Program (C200010).

References

[1] *Avionics Application Software Standard Interface Part 1 – Required Services*. ARINC Industry Activities. Available: https://www.sae.org/standards/content/arinc653p1-5.

[2] S. Macenski, T. Foote, B. Gerkey, C. Lalancette and W. Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, 2022.

[3] *Aircraft Data Network Part 1 – Systems Concepts and Overview*. ARINC Industry Activities. Available: https:https://www.sae.org/standards/content/arinc664p1-2.

[4] *IEEE 802.1BA - Audio Video Bridging (AVB) Systems*. IEEE. Available: https://www.ieee802.org/1/pages/802.1ba.html.

[5] R. Recio, B. Metzler, P. Culley, J. Hilland and D. Garcia, *A Remote Direct Memory Access Protocol Specification*, RFC 5040, IETF, 2007.

[6] *Supplement to InfiniBand Architecture Specification Volume 1 Release 1.2.2 Annex A17: RoCEv2 (IP Routable RoCE)*. IBTA. Available: https://www.infinibandta.org/.

[7] J. Liu, J. Wu, S. P. Kini, P. Wyckoff and D. K. Panda, "High performance RDMA-based MPI implementation over InfiniBand," in *Proc. the 17th ICS*, New York, NY, USA, 2003, pp. 295-304.

[8] S. -H. Kim, H. -R. Chuang, R. Lyerly, P. Olivier, C. Min and B. Ravindran, "DeX: Scaling Applications Beyond Machine Boundaries," in *Proc. the 40th IEEE ICDCS*, Singapore, 2020, pp. 864–876.

[9] A. Dragojevi, D. Narayanan, O. Hodson and M. Castro, "FaRM: Fast Remote Memory," in *Proc. the 11th USENIX NSDI*, USA, 2014, pp. 401-414.

[10] S.-H. Lee, S. Han, and H.-W. Jin, "A Configurable, Extensible Implementation of Inter-Partition Communication for Integrated Modular Avionics," in *Proc. IEEE RTCSA*, Korea, 2012, pp. 453–458.

[11] H. Prez and J. J. Gutirrez, "Handling Heterogeneous Partitioned Systems through ARINC-653 and DDS," *Computer Standards & Interfaces*, vol. 50, pp. 258-268, 2017.

[12] J. Chen, K. Chen, C. Du, and Y. Liu, "Design and Implementation of a Virtual ARINC 653 Simulation Platform," *Simulation*, vol. 97, no. 6, pp. 427–436, 2021.

[13] H.-C. Jo, J.-K. Park, H.-W. Jin, H.-S. Yoon and S. H. Lee, "Portable and Configurable Implementation of ARINC-653 Temporal Partitioning for Small Civilian UAVs," *IEEE Access*, vol. 7, pp. 142478–142487, 2019.

[14] *RDMA Core Userspace Libraries and Daemons*. Available: https://github.com/linux-rdma/rdma-core.