Oops! It's Too Late. Your Autonomous Driving System Needs a *Faster* Middleware

Tianze Wu[1], Baofu Wu[2], Sa Wang[3],Liangkai Liu[4], Shaoshan Liu[5], Yungang Bao[6], Weisong Shi[4]

***Abstract*—Autonomous Driving (AD) has entered a period of rapid development in recent years. With the amount of sensors and control logics installed increasing tremendously to guarantee robustness, a big challenge is posed for AD middleware. Both the academia and the industry are eager for an investigation of the performance of middlewares in Autonomous Driving Vehicles (AVs). To fill this gap, we summarize typical communication scenarios of AVs and evaluate different communication mechanisms of three popular open-source middlewares comprehensively. Besides, we construct a benchmark pack named ComP which consists of a perception communication scenario and a group of real AD applications for researchers to assess middleware performance. Our findings provide useful guidelines for researchers and insightful optimization advice for designing middlewares.**

***Index Terms*—Embedded Systems for Robotic and Automation, Software Architecture for Robotic and Automation, Distributed Robot Systems**

I. Introduction

OVER the past decades, the autonomous driving (AD) technique has made rapid and tremendous progress [1]–[4]. However, some people still are profoundly concerned and stay negative towards AD, especially *full* AD[1], becoming a reality. One of the main reasons behind this is the robustness of AVs. It is a big challenge for AVs to transmit and process a large amount of data within stringent deadlines in all the complicated situations. The state-of-the-art devotes much effort to accelerate the data processing part in recent years, including algorithms optimization [5] and specific hardware acceleration [6]. However, there still remains a missing part. How about the communication latency? Since the data processing latency has been dramatically reduced, many researchers and companies speculate that the communication latency may become a new performance bottleneck [7]–[11] in AVs.

Therefore, in this paper, we raise the question: *how about the communication latency of today's open-source middlewares used in AVs?* Can they meet the need of transmission requirement of AVs? As far as we know, many existing works [12]–[14] have explored the performance of middlewares, but none of them could answer the above questions comprehensively.

To answer these questions, we carefully choose three influential and widely used open-source middlewares (ROS1 [15], ROS2 [16], and Cyber [9]) and conduct a bunch of experiments to evaluate their communication performance with different sensors under various scenarios. Meanwhile, we break down the communication process with perf [17] to analyze the root cause of the communication delay. We further provide several guidelines to improve communication performance. Besides, we construct a benchmark pack named ComP [18], which consists of a perception communication scenario and a group of real AD applications. We use it to evaluate the middleware's performance in different platforms. Based on the experimental results, we analyze the communication latency in real AD systems. Our main findings are summarized as follows:

Key findings: (1) At present, communication through shared memory is the optimal choice considering both performance and reliability. (2) Data copy and serialization/deserialization operations are the major sources of communication overhead, the performance penalty from task scheduling is small. (3) Optimization based on the communication characteristics of different scenarios can bring significant benefits. (4) The higher the performance of the platform is, the more likely communication becomes the bottleneck.

In a nutshell, this work makes the following contributions:

- We summarize the typical communication features in AVs and evaluate different communication mechanisms of three popular open-source middlewares comprehensively.
- We construct and opensource a benchmark pack named ComP which consists of a perception communication sce-

Manuscript received: March, 4, 2021; Revised April, 24, 2021; Accepted July, 4, 2021.

This paper was recommended for publication by Editor Tamim Asfour upon evaluation of the Associate Editor and Reviewers' comments. This work was supported in part by Key-Area Research and Development Program of Guangdong Province (NO.2020B010164003), the National Natural Science Foundation of China (Grant No. 62090020), Youth Innovation Promotion Association of Chinese Academy of Sciences (2013073, 2020105), and the Strategic Priority Research Program of Chinese Academy of Sciences (Grant No. XDC05030200).

[1]Tianze Wu is with State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Acedemy of Sciences, and University of Chinese Academy of Sciences `wutianze@ict.ac.cn`

[2]Baofu Wu is with School of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou 310018, China, and Wayne State University, Detroit, MI 48202, United States `baofu.wu@hdu.edu.cn`

[3]Sa Wang is with State Key Laboratory of Computer Architecture, Institute of Computing Technology, University of Chinese Academy of Sciences, and Institute of Computing Technology(Nanjing), Chinese Academy of Sciences `wangsa@ict.ac.cn`

[4]Liangkai Liu and Weisong Shi are with Wayne State University `liangkai@wayne.edu` `weisong@wayne.edu`

[5]Shaoshan Liu is with PerceptIn `shaoshan.liu@perceptin.io`

[6]Yungang Bao is with State Key Laboratory of Computer Architecture, Institute of Computing Technology, University of Chinese Academy of Sciences, and Peng Cheng Laboratory `baoyg@ict.ac.cn`

Digital Object Identifier (DOI): see top of this page.

[1]The automated system takes full control of all driving tasks under all driving conditions that a human driver can manage.

nario and a group of real AD applications for researchers to evaluate middlewares.

- Our findings can provide useful guidelines for AD researchers and insightful optimization advice for designing middlewares.

II. BACKGROUND

A. Real-time requirement for communication in AVs

End-to-end latency is one of the most crucial metrics for building real-time systems, especially for AVs [6]. It dominates the speed of the vehicle's response to the emergency. In AVs, end-to-end latency is the time from when a new event is sensed until the control commands take action. Since the time[2] from the control commands are generated until they actually take action are relatively short compared to the time spent on the AD on-board computer [19], in this paper, we mainly focus on the first *half* time which consists of data transmission time and data processing time.

In recent years, data processing latency has been gradually reduced with the development of hardware accelerators and algorithms [6], [20], [21]. In our experiment, we found that with the help of GPU RTX 2080Ti, the time consumption of common algorithms used in AVs, such as Yolo [22], could be reduced to $\mathbf{10}ms$. So, to avoid becoming a performance bottleneck, we believe that the communication latency should be at least one order of magnitude less than the computation latency, which is around $\mathbf{1}ms$. It should be noted that it is not very reasonable to directly use absolute values, because different scenarios have different end-to-end latency requirements. However, $1ms$ is one order of magnitude lower than the running time of commonly used AD algorithms, and two orders of magnitude lower than the fastest human driver's reaction time ($100ms$ end-to-end latency) [23], which is a good reflection of real-time requirements of communication latency in AVs. In this paper, we directly compare the communication latency with the corresponding computing latency.

B. Middleware in AVs

1) Middleware Architectures: As shown in Fig.1, middleware is a management layer between OS and the upper applications [24]. The basic function of middleware is to bind different services together [24]–[26]. The **communication model** of mainstream middlewares is many-to-many unidirectional data exchange. Applications that produce data become publishers, and applications that consume data become subscribers [27]. Each time a publisher posts new data, the middleware propagates the information to all interested subscribers. The information flow is regulated by Quality of Service (QoS) policies established between the data exchange entities.

Besides the communication function, most middlewares also provide common services and capabilities such as task scheduling, data management outside of what is offered by OS [28]. The usability and programmability provided by middleware have attracted many developers to build their projects based on it. It should be noted that not only in distributed systems, middleware has also been widely used in some complex single-machine systems such as AD systems. Because the modules in single-machine system also need to be managed and communicate with each other, and the middleware just provides these basic functions, which greatly reduces the development cost.

2) Widely used middlewares in AVs: In this paper, we carefully select three well-known middlewares for evaluation.

Robot Operating System (ROS1) has been widely used in AD development. The world's first "all-in-one" open-source AD software, Autoware.AI [20], is based on ROS1. The socket-based Inter-Process Communication (IPC) mechanism of ROS1 brings high compatibility and extensibility [7].

ROS2 is an evolved version of ROS1. ROS2 uses Data Distribution Service (DDS) [29] as its communication foundation, which claims to work well in real-time distributed systems. Apex.AI [30] and Autoware.AI's successor Autoware.Auto both build their own system based on ROS2.

Cyber from Apollo [31] is another influential open-source AD middleware. It is designed specifically for AVs. ROS1 was the underlying middleware of Apollo at first and was replaced by Cyber since Apollo version 3.5. Compared to ROS1 and ROS2, the management and scheduling capabilities of Cyber are enhanced, and we will discuss the effectiveness of these capabilities in the following sections.

3) Underlying communication technologies: As shown in Fig.1, there are three main communication technologies used by middlewares. Socket-based methods are widely used in robots. ROS1 utilizes **TCP** and **UDP**, while ROS2 and Cyber choose **DDS** [29] which aims to enable dependable, high-performance, real-time data exchanges. During our experiments, we find that DDS in ROS2 and Cyber uses UDP to do the communication. The biggest advantage of socket-based methods is high reliability due to its loosely coupled mechanism [7], as the crash of a single module does not cause the entire system to crash. However, socket-based methods often have large overhead.

Then, **IntrA-Process (IAP)** and **SHared Memory (SHM)** are presented. Communication between different threads (inter-thread) in the same process (intra-process) is efficient because it realizes zero-copy data transport. ROS1 nodelet and ROS2 Intra-Process communication are two representatives of IAP methods. SHM is the most efficient way of IPC [7] methods since it also avoids data copy operations. However, SHM still requires serialization and deserialization operations. Cyber provides an SHM method based on a shared buffer with ring structure association with its publishers and subscribers [9]. We can see that these methods are different tradeoffs between performance and reliability [32].

TABLE I
TYPICAL SENSORS IN AVS.

Sensor	IMU	CV	LidarL	LidarH	Radar
Frequency/Hz	200	50	20	20	20
Message Size/Byte	1k	4M	4M	8M	10k

[2]This time includes Control Area Network (CAN) bus transmits control commands to the vehicle's actuator and the actuators to start reacting.

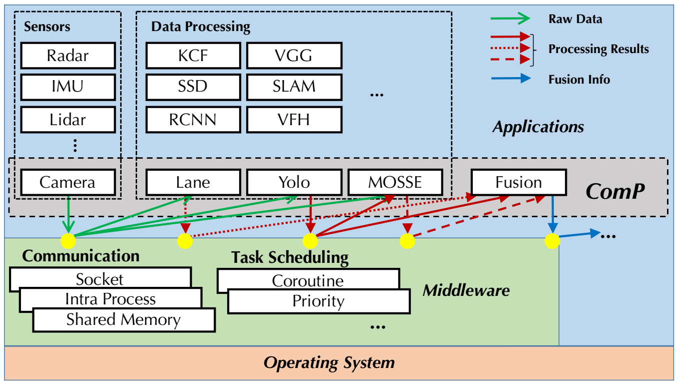

Fig. 1. In a broad sense, middleware is a management layer between OS and the upper applications.

C. Characteristics of Sensors in AVs

Different sensors in AVs have different posting frequencies and message sizes. We summarize the parameters of some typical AD sensors in Tab.I according to products commonly used in AVs [31], [33]–[35]. **Inertial Measurement Unit (IMU)** consists of gyroscopes and accelerometers. It has an updated value every $5ms$ and is used as a complementary sensor to GPS. **Computer Vision (CV)** ability is obtained by cameras, the size of one high-resolution picture can be $4MB$, and cameras usually work at 30-50FPS. Lidar has a more extensive sensing range and is less affected by a lousy environment than cameras. Lidar with different scanning densities produces data in different sizes, **Lidar Light (LidarL)** and **Lidar Heavy (LidarH)** in Tab.I approximately represent two possible data frame sizes of 32 and 64 line Lidar. Radar uses electromagnetic waves for scanning. It typically operates at 10-20FPS and generates around $100KB$ of data per second.

D. Two Typical Communication scenarios in AVs

When the messages generated by a component are transmitted to more than one processing component, we refer to this scenario as the **1-N** scenario. 1-N scenario is common in AVs. For example, the Lidar data could be used for avoiding obstacles, building HD map and localization [35] respectively.

Another scenario we talk about is data fusion, and we call it the **N-1** scenario. Take the high-level sensor fusion [36] in AVs as an example. When data processing is over, sensor fusion techniques are used for the multi-sensor collaboration to correct errors and aggregate results [21].

III. Methodology

A. Measurement metric

Latency is our primary measurement metric. In local cases, we record the elapsed time from when the data is released from one node to another node. And in distributed cases, we record the round-trip time between the time the node sends a message and the time it receives a return message. The release time is recorded right before the publish function is called, and the receive time is recorded right as the callback function is called. The time counter function we used in ROS1 and ROS2 is **clock_gettime**, while the function in Cyber is **high_resolution_clock**. They both provide nanosecond accuracy and their overhead could be ignored (getting current time accounts for 4.23% or less of the communication latency). Each experiment runs for 200 seconds, and we discard the first 200 samples to mitigate initialization effects.

B. Experiment design

1) Micro experiments: We first test the basic characteristics of these communication methods through micro experiments. The experimental parameters are set according to typical sensors' characteristics to show whether these methods could handle common communication tasks in AVs. And by allocating different resources and giving different QoS settings, we depict the contributory factors to the communication performance. In addition to simple point-to-point scenario, we analyze the performance and fairness of different methods in complex scenarios (1-N, N-1). Further, we explore two interesting ways that could improve the performance. By breaking down the communication process with perf and reading the source code, we analyze the reasons for these experimental results and try to find the bottlenecks.

2) With ComP in different platforms: Besides micro evaluations, we test the performance of Cyber on different platforms with our benchmark pack ComP. Based on experimental results running on different platforms and using different methods, we reveal the current relationship between transmission latency and processing latency and further discuss some problems encountered in real systems.

C. Design of ComP

1) Application selection: To highlight the interdependence of components in AD systems, we choose the perception scenario which is responsible for detecting the surrounding environment and tracking obstacles. Inspired by [6], we select one typical algorithm for each component to ensure that ComP can be easily deployed. For **object detection**, we use YOLOv4 [22] which is one of the state-of-the-art object detectors based on deep learning and achieves a balance between performance and speed. For **object tracking**, we use Minimum Output Sum of Squared Error (MOSSE) Filter [37] which runs fast and has acceptable accuracy. For **lane detection**, our OpenCV [38] based approach first conducts the contour and geometric analysis to extract the relevant lane line information, then performs the pixel scan. Finally, the result is generated by line fitting.

2) How ComP works: The processing pipeline of ComP is shown in Fig.1. First, raw data (image) from Ford Multi-AV Seasonal Dataset [39] is published by a Camera component. Then images are transmitted to three processing components. Component *Lane* is for detecting lane lines, and its results are sent to *Fusion* directly; Component *Yolo* is for identifying objects, and two components utilize its results; Component *MOSSE* is for vehicle tracking, and it needs object coordinate info to initialize the objects to be tracked once it loses the target. Results of these components are finally fused in *Fusion*.

TABLE II
DEFAULT QOS CONFIGURATION OF DDS.

History Policy	Depth	Reliability Policy	Durability Policy
KEEP_LAST	1	RELIABLE	VOLATILE

D. Experimental Setup

The experiments are conducted on two different hardware settings, namely a server PC, with two Intel Xeon E5-2630 v4 CPUs, 32 GB RAM, and one 2080Ti Nvidia GPU. In addition, we use a Jetson AGX Xavier with 32 GB RAM as the embedded platform. Docker container virtualization technology [40] is used to adjust resources, and the performance loss caused by Docker is acceptable [41]. The operating system is Ubuntu 18.04, and the kernel version is 4.15.0. ROS1 version is Melodic, ROS2 version is Foxy, and Cyber version is 5.5.0. FastDDS is the DDS implementation in ROS2 and Cyber; the ROS2 FastDDS version is rmw_fastrtps, while the Cyber FastDDS version is 1.5.0. The default DDS QoS policy used in our experiments is shown in Tab.II, and the default queue length of publisher and subscriber is 1.

IV. RESULTS OF MICRO EXPERIMENTS

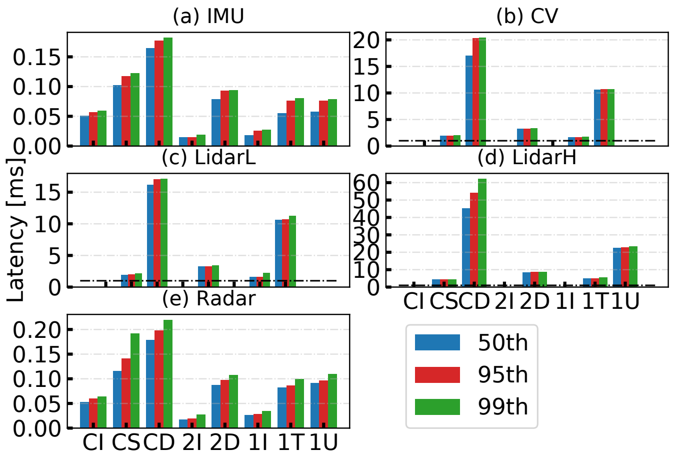

Fig. 2. **Abbreviation: Cyber IAP(CI); Cyber SHM(CS); Cyber DDS(CD); ROS2 IAP(2I); ROS2 DDS(2D); ROS1 IAP(1I); ROS1 TCP(1T); ROS1 UDP(1U).** 50th-, 95th-, 99th- percentile latency of different methods when transferring different kind of sensor data. Evaluation was performed on the server PC. In figures b, c, d, we indicate the positions of $1ms$ latency with black dotted lines.

A. Basic characteristics

Fig.2 shows the respective latency characteristics of different communication methods when transferring different kinds of sensor data. We observe a gap between the median latency and the long-tail (95th-, 99th- percentile) latency of each method. Therefore, long-tail latency should be used to evaluate the performance if stringent predictability is needed.

By comparing scenarios (Fig.2 (b) and (c)) with different frequencies and the same message size, we can see that the publishing frequency is not an essential factor in transmission latency. [13] even found that latency decreases with increasing frequency, which might be due to energy-saving features or scheduling scheme.

However, by comparing scenarios (Fig.2 (c), (d), and (e)) with the same frequency and different message sizes, it is clear that message size is an essential determinant of transmission latency especially in socket-based methods. By breaking down the communication process, we find that the reason is because the data copy and serialization/deserialization operations are the major sources of communication overhead. As a result, when the message size is small ((a), (e) in Fig.2), all methods can satisfy the latency requirement. Besides, we can see from Fig.2 that only IAP-type methods are hardly affected by the message size since they avoid all data copy and serialization/deserialization operations. It should be noted that the performance of ROS2 IAP has exceeded ROS1 IAP. This discovery updates the observation in [12], which says that the performance of ROS1 IAP is better than ROS2 IAP at that time. SHM avoids copying data but still requires serialization/deserialization, so its performance is not as good as IAP, but is much better than socket-based methods.

For socket-based methods, we could see that Cyber DDS is not as good as ROS2 DDS. That's because the overhead of serialization/deserialization operations of Cyber are much bigger than ROS2. We believe that FastDDS [27] is optimized specifically for ROS2, which greatly reduces the overhead of serialization/deserialization. Another interesting finding is that although ROS2 DDS and ROS1 UDP both invoke UDP to communicate, there is a wide performance gap between them. By analyzing their call stacks, we find that there are two main reasons for this gap: 1. The overhead of ROS1 UDP's call stack to UDP (sys_writev) is much larger than that of ROS2 DDS (sys_send_msg). 2. The maximum UDP datagram size supported by ROS1 is $1500bytes$, while ROS2 DDS can use $64KB$, so ROS1 UDP has large message splitting and merging overhead when transmitting big message.

Finding 1: Data copy and serialization/deserialization operations are the major sources of communication overhead.

As shown in Fig.3 (a), since the experiments were carried out locally, communication delay was positively correlated with CPU utilization under same frequency. However, providing more CPU resources does not necessarily reduce latency. We evaluate the latency of these methods with different CPU resource constraints. Take Cyber DDS in Fig.3 (b) as an example, latency decreases as CPU resource goes from insufficient to sufficient and does not change much once the CPU resource is enough. So in the other experiments in this article, we make sure that CPU resources are sufficient and that latency is not affected.

To summarize, communication through shared memory is the optimal choice for both performance (latency is close to $1ms$) and reliability (do not need to be in the same process) at present. In the one-to-one scenario, we rank the performance of all these methods in this order: **ROS2 IAP >ROS1 IAP >Cyber IAP >Cyber SHM >ROS1 TCP >ROS2 DDS >ROS1 UDP >Cyber DDS.**

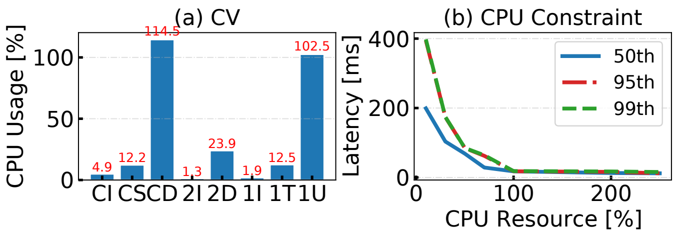

Fig. 3. (a) shows the CPU usages of different methods when transferring CV data in server PC. (b) shows the latency of Cyber DDS with different CPU resource allocations when transferring CV data.

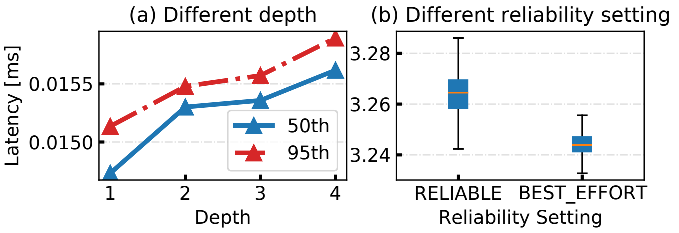

Fig. 4. (a) is an example of ROS2 IAP transferring IMU data with different publisher and subscriber depths. (b) is an example of ROS2 DDS transferring CV data with different reliability settings.

B. QoS settings

The three middlewares provide many configurable options or QoS settings. We tried different settings, and here are our findings.

Queue length (depth) could be increased to help mitigate extra overhead caused by the frequent replacement of old messages in the queue. However, handling out-of-date messages could make AVs less safe. Fig.4 (a) illustrates that the latency would increase as the depth. Therefore, the optimal length needs to be selected according to actual requirements. At present, there is no adaptive solution to find the most suitable depth, and the depth is usually 1 in real-time systems.

Reliability setting of socket-based methods usually has two options: RELIABLE and BEST_EFFORT; TCP and UDP can also be seen as two mechanisms with different reliability guarantees. For DDS methods, as can be seen in Fig.4 (b), the latency of BEST_EFFORT is slightly less than RELIABLE. The main reason for this gap is that there is an additional confirmation process using RELIABLE, and UDP is called to send and receive messages in both of them. For ROS1 TCP and UDP, as shown in Fig.2 and Fig.3, ROS1 TCP has lower latency and CPU usage than ROS1 UDP. The root cause is the overhead of dividing the message into several datagrams and merging datagrams into original message. We find that increasing the datagram size from 500 to 1500$bytes$ could reduce the medium latency by 64% on average when transferring message sized 4MB. And for ROS1 TCP, enabling TCP_NODELAY could disable the use of the Nagle algorithm, which claims to bring lower publishing latency at the cost of efficiency, but the effect is not obvious in our experiments.

Finding 2: Some QoS settings may incur additional overhead, but they usually have little effect on latency in local communication environment.

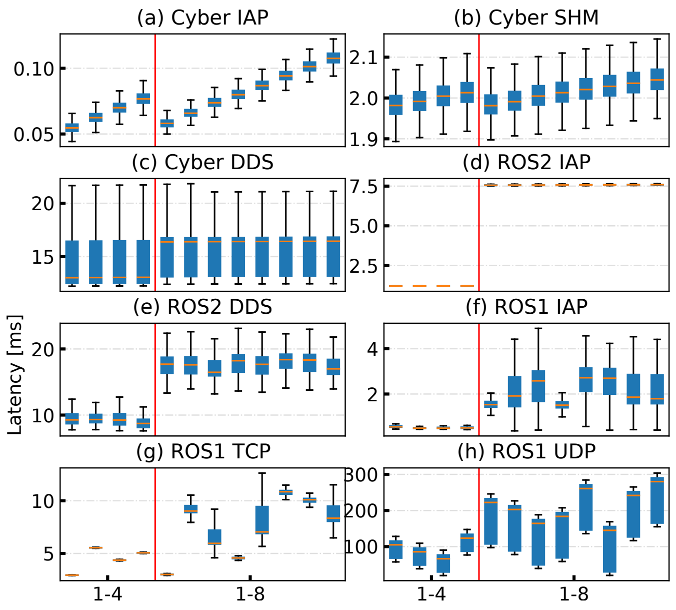

Fig. 5. 4/8 subscribers subscribe to a single topic. Each box in the figure represents one subscriber.

C. Complex communication scenarios

1) Multiple destinations publisher (1-N): This section prepares multiple subscribers subscribing to the same topic and measures the latencies of each linkage.

As shown in Fig.5, we are surprised that in 1-N scenario, the communication **performances** of ROS1 IAP and ROS2 IAP have a great decline. We use perf to check the details of their running processes in 1-N scenario and find the reason. Because ROS1 IAP and ROS2 IAP use the smart pointer provided by Boost or C++ when publishing, if there is only one subscriber, ownership can be transferred directly. Once the number of subscribers is greater than 1, a copy of the data is needed for each additional subscriber, resulting in an increase in latency as the number of subscribers increases. The subscribers in Cyber IAP and Cyber SHM share messages, so the copy step is completely eliminated. Besides, for socket-based methods, their high resource consumption when transferring large messages makes them difficult to support more linkages. Among these socket-based methods, ROS1 TCP consumes the least amount of resources and performs best. We built multiple 1-1 linkages transferring CV data using different methods. Results show that with 4 CPU cores allocated, ROS1 TCP could support more than eight linkages without encountering much packet loss. At the same time, other UDP-based methods could only support four or at most six linkages. The main reason, as discussed earlier, is that the unpacking and merging operations of UDP consume too much CPU cycles.

Fairness is also crucial for a real-time system, the time different nodes receive the same message should be close. As demonstrated in Fig.5, the latency gap between different subscribers of ROS1 is the largest because it schedules message publication in order [12]. When it comes to Cyber and ROS2, the situation is much better. Moreover, although the latency of nodes receiving messages has a clear sequence using Cyber, the absolute gap is small (much less than 1ms).

Finding 3: Optimization based on the communication characteristics of different scenarios can bring significant benefits. In this regard, Cyber does better than the other two middlewares.

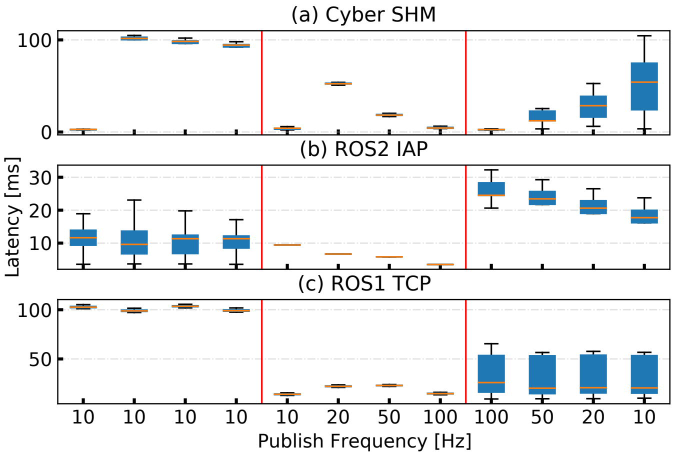

Fig. 6. One subscriber receives data from four publishers with different publish frequencies. Each box in the figure represents one channel associated with one publisher. The message size is 4MB.

2) Data fusion (N-1): This section uses Cyber SHM, ROS2 IAP, and ROS1 TCP as representative methods to conduct our experiments. The message size is 4MB. It should be noted that the message size has an influence on the absolute latency but does not affect our results.

Cyber synchronizes data according to the first channel's frequency, which means that the synchronization is performed whenever the subscriber 0 receives a message. Hence, we could see from Fig.6 (a) that the performance of subscriber 0 does not lose much, while the latencies of other subscribers are much higher than usual and are inversely proportional to the frequency. This strategy's advantage is that it is deterministic, while the disadvantage is that the latency difference between different subscribers is big. When assigning subscriber 0, we should choose the one with the highest frequency to minimize the replacement of other channels' data, and developers can make full use of this mechanism to achieve some deterministic requirements. Both ROS1 and ROS2 use adaptive algorithms to synchronize data. As shown in Fig.6 (b) and (c), we observe that the advantage of this strategy is good fairness among subscribers, and the disadvantage is that the behavior and performance are strongly related to the adaptive algorithm. Moreover, ROS1 and ROS2 can achieve the best overall performance when synchronization happens in data frequency from low to high.

Finding 4: Fusion strategies, the frequency and order of channels to be fused would affect the latency pattern of data fusion.

D. Other ways to improve performance

1) Scheduler policy: Before exploring scheduling policies, we test the effects of multi-threading and different CPU usages on communication latency by running one or more CPU-consuming processes besides communication processes on a single core. The experimental results show that increasing the number of threads without changing CPU usage has no significant impact. Only an increase in the CPU usage of the system will result in an increase in communication latency. There are two types of methods that are most affected by CPU usage: IAP-type methods (up to 67% increase in latency) and UDP-based methods (packet loss rate can be 100%). By breaking down the communication process, we find possible reasons. For IAP-type methods, the elimination of data copying and serialization/deserialization makes the scheduling overhead account for a much larger proportion of the total communication overhead (around 35%) than other methods (less than 5%). So an increase in the system's scheduling time has the greatest impact on the IAP method. However, since the communication latency of IAP-type methods is so small, the increase in latency is imperceptible. For UDP-based methods, their packet loss rate is higher because UDP itself is very vulnerable to interference especially when message size is large. For other methods, in addition to the low proportion of scheduling overhead, another reason that their latencies do not increase much is that they consume little CPU time. Then the task completion time will not be affected much under Linux's time slice scheduling mechanism.

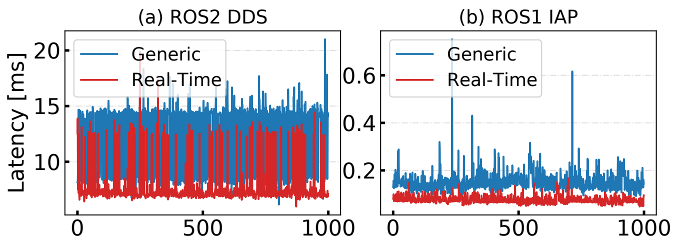

Fig. 7. Latencies of two different kernels, the x-axis represents timeline. The real-time kernel is built by patching the generic Linux kernel PREEMPT RT [42]. Two methods is shown in this figure and the results of other methods are similar.

Finding 5: The overhead associated with task scheduling does not have a significant impact on the communication latency.

The scheduler policy of Cyber provides two ways to ensure the quality of critical links. One is to give them a higher priority so that they could be scheduled first; the other is to allocate exclusive CPU cores to them to avoid interference. Take the Cyber SHM experiment in Fig.5 (b) as an example. The subscriber with the biggest latency is chosen to be protected (the eighth subscriber). Giving it a higher priority moves its latency ranging from eighth to third; assigning it a separate CPU core moves its ranking to second. However, the actual performance gains are little because the scheduling overhead itself is negligible.

2) Real-Time (RT) kernel: Fig.7 shows the results of two methods running in different kernels, and we could see that RT kernel reduces overall communication latency and performance jitter. So it is recommended to use RT kernel even the upper-level applications are not changed for real-time kernel. This improvement is achieved through a series of optimizations made by the RT kernel, the details will not be explored in this article.

V. Results of ComP experiments

In this section we use ComP to evaluate Cyber in different hardware platforms. Data in Tab.III is collected running the group of real AD applications of ComP all together. We make sure that computing resource is sufficient during the experiments.

1) Comparison between different platforms: It can be seen from Tab.III that the ratio between transmission latency and processing latency is larger in server PC than in AGX. Although data processing and transmission cost less time in server PC, the difference of data transmission latency between the two platforms is much smaller than that of data processing latency. Therefore, reducing transmission latency can lead to more performance gains on more powerful platforms. For example, replacing Cyber SHM with Cyber IAP on server PC gives *MOSSE* a 27% performance boost. However, the same action brings only 4% improvement on AGX. It should be noted that the reduction of communication overhead in a less powerful platform is also meaningful since it can save much CPU resource and then improve the overall performance of the system.

Finding 6: The higher platform's performance, the more likely communication is to become a bottleneck.

TABLE III
The ratio of transmission latency to processing latency.

Method and Platform	*Lane*	*Yolo*	*MOSSE*
IAP-PC/AGX	1 / 1 %	4.2 / 3.1 %	3.1 / 8.7 %
SHM-PC/AGX	7.5 / 4.2 %	37.0 / 9.9 %	42.9 / 31.6 %
DDS-PC/AGX	117.4 / 51.5 %	566.7 / 132.6 %	733.3 / 478.9 %

2) Data fusion: Since the fusion action of Cyber is triggered once the first channel receives a message, we could make use of this strategy in some cases. For example, *MOSSE* receives data from *Camera* and *Yolo*, but the detection results from *Yolo* are only used to re-initialize the tracker when it loses targets. Therefore, we only need to ensure that the data from *Camera* could be processed as soon as possible. So we can set *Camera* as the first channel to minimize its latency. Compared with the fusion strategy of ROS1 and ROS2, the strategy of Cyber is more controllable.

The latency gap of receiving data from different channels in *Fusion* could reach 100ms in our experiments. The main reason for such a large gap is the low data generation frequency due to the long processing time. Unlike *MOSSE* we talk about before, the best thing for a component such as *Fusion* to reduce the latency is to increase data generation frequency (increase the Camera's data generation frequency or speed up data processing). For ROS1 and ROS2, merely increasing the frequency cannot solve the problem, and it is necessary to choose the appropriate fusion strategy and order according to the frequency of different channels. Of course, many AD solutions now implement their own fusion mechanism. We have learned that many of them separate the message update and data processing. Message reception of each channel takes place independently. When data processing is triggered, message queue of each channel is locked and the up-to-date message is fetched from queues. This is actually similar to Cyber's fusion strategy.

3) Publishing frequency and processing time: We observe that the communication latency is high during the experiments if the message is published faster than it can be processed. One reason is that the arrived messages may need to wait for the previous data processing operation to complete. Moreover, if the length of the subscriber's waiting queue is not long enough, newly arrived messages will frequently replace older messages waiting in the queue. They both introduce additional latency and waste CPU cycles. It should be noted that reducing the publishing frequency may introduce new problems. Publishing messages too slowly would cause the algorithm process to fall into sleep frequently, resulting in large notifying and waking up latency.

Finding 7: Additional overhead can be introduced if the publishing frequency and corresponding processing time are not well matched.

VI. Vehicle-to-everything (V2X)

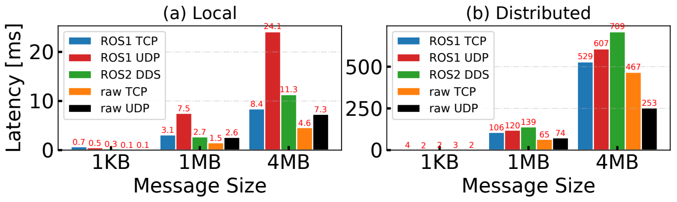

Fig. 8. Communication performance of socket-based methods in local and distributed environments. The latency in the image is half the round-trip latency.

V2X is communication between a vehicle and any entity that may affect, or may be affected by, the vehicle. In this section we show the performances of different communication methods in a V2X scenario to see whether it is possible to conduct complex distributed data processing in a V2X scenario nowadays. We use AGX as the vehicle and server PC as a computing node. They are connected to a wireless network under a same router. We test the round-trip time between AGX and server PC, as a comparison we also conducted the experiments on AGX locally.

As shown in Fig.8, the communication performance in the distributed environment has a great loss compared to the local environment especially when message size is large. In addition, UDP based methods have serious packet loss problems when transmitting large messages. Obviously, most of the computing load still needs to be carried out locally today.

We also measured the performance of original TCP and UDP, and we can see from Fig.8 that the performance loss brought by the middleware (within 50%) is acceptable. Considering that middleware can provide more reliable and easy to use communication functions, we suggest that users directly choose mature middleware solutions to realize communication functions. It should be noted that the reason why ROS1 UDP latency is so high in the local environment is mainly because ROS1 UDP only supports datagram size of up to $1500 bytes$ which we mentioned in Section.IV-B.

VII. Related Work

[12] showed us the difference between ROS1 and ROS2, and it tested different DDS implementations. [13] not only evaluated the performances of different DDS implementations, but also profiled the stack of ROS2 and pointed out latency bottlenecks. [14] evaluated ROS2 for AVs, it focused on real-time capability and found that the jitter of time difference could be reduced by using a real-time kernel. Also from a real-time perspective, [43] proposes a scheduling model and a response-time analysis for ROS2, it enables ROS users to determine temporal safety and latency properties of their applications. Besides these works, several studies in robotic systems tried to use new methods in communication middlewares. For example, [9], [44] used SHM methods to minimise memory copy operations, [7] designed a combined data transmission mechanism to provide both high performance and high reliability.

VIII. Conclusion and future outlook

We perform an in-depth study on three open-source middlewares and presents ComP for evaluating middleware's communication performance. Our study reveals many interesting findings and provides useful guidelines for AD researchers and insightful optimization advice for designing middlewares.

IX. Acknowledgement

We would like to thank anonymous reviewers for their valuable feedbacks and suggestions. We thank our group members for their help on this work.

References

[1] T. Kanade, C. Thorpe, and W. Whittaker, "Autonomous land vehicle project at CMU," in *Proceedings of the 1986 ACM fourteenth annual conference on Computer science*, 1986, pp. 71–80.

[2] J. Schmidhuber, "Robot car history," [Online], http://people.idsia.ch/juergen/robotcars.html.

[3] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," CARNEGIE-MELLON UNIV PITTSBURGH PA ARTIFICIAL INTELLIGENCE AND PSYCHOLOGY . . . , Tech. Rep., 1989.

[4] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke *et al.*, "Junior: The Stanford entry in the urban challenge," *Journal of field Robotics*, vol. 25, no. 9, pp. 569–597, 2008.

[5] N. O'Mahony, S. Campbell, A. Carvalho, S. Harapanahalli, G. V. Hernandez, L. Krpalkova, D. Riordan, and J. Walsh, "Deep learning vs. traditional computer vision," in *Science and Information Conference*. Springer, 2019, pp. 128–144.

[6] S.-C. Lin, Y. Zhang, C.-H. Hsu, M. Skach, M. E. Haque, L. Tang, and J. Mars, "The architectural implications of autonomous driving: Constraints and acceleration," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, 2018, pp. 751–766.

[7] W. Liu, H. Wu, Z. Jiang, Y. Gong, and J. Jin, "A robotic communication middleware combining high performance and high reliability," in *2020 IEEE 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. IEEE, 2020, pp. 217–224.

[8] J. Tang, S. Liu, B. Yu, and W. Shi, "Pi-edge: A low-power edge computing system for real-time autonomous driving services," *arXiv preprint arXiv:1901.04978*, 2018.

[9] Baidu, "Apollo Cyber," [Online], https://github.com/ApolloAuto/apollo/tree/master/cyber.

[10] NVIDIA, "DRIVE OS," [Online], https://docs.nvidia.com/drive/index.html.

[11] "AUTOSAR," [Online], https://www.autosar.org/.

[12] Y. Maruyama, S. Kato, and T. Azumi, "Exploring the performance of ROS2," in *Proceedings of the 13th International Conference on Embedded Software*, 2016, pp. 1–10.

[13] T. Kronauer, J. Pohlmann, M. Matthe, T. Smejkal, and G. Fettweis, "Latency overhead of ros2 for modular time-critical systems," *arXiv preprint arXiv:2101.02074*, 2021.

[14] M. Reke, D. Peter, J. Schulte-Tigges, S. Schiffer, A. Ferrein, T. Walter, and D. Matheis, "A self-driving car architecture in ros2," in *2020 International SAUPEC/RobMech/PRASA Conference*. IEEE, 2020, pp. 1–6.

[15] Open Robotics, "ROS," [Online], https://www.ros.org/.

[16] ——, "ROS2," [Online], https://index.ros.org/doc/ros2/.

[17] B. Gregg, "Perf," [Online], 2029, http://www.brendangregg.com/perf.html.

[18] Tianze Wu, Baofu Wu, "Comp," [Online], https://github.com/wutianze/AD_Middle_Test.

[19] B. Yu, W. Hu, L. Xu, J. Tang, S. Liu, and Y. Zhu, "Building the computing system for autonomous micromobility vehicles: Design constraints and architectural optimizations," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 1067–1081.

[20] S. Kato, E. Takeuchi, Y. Ishiguro, Y. Ninomiya, K. Takeda, and T. Hamada, "An open approach to autonomous vehicles," *IEEE Micro*, vol. 35, no. 6, pp. 60–68, 2015.

[21] R. Hussain and S. Zeadally, "Autonomous cars: Research results, issues, and future challenges," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1275–1313, 2018.

[22] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.

[23] A. Newell and S. K. Card, "The prospects for psychological science in human-computer interaction," *Human-computer interaction*, vol. 1, no. 3, pp. 209–242, 1985.

[24] R. E. Schantz and D. C. Schmidt, "Middleware," *Encyclopedia of Software Engineering*, 2002.

[25] S. Liu, L. Liu, J. Tang, B. Yu, Y. Wang, and W. Shi, "Edge computing for autonomous driving: Opportunities and challenges," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1697–1716, 2019.

[26] D. Bakken, "Middleware," *Encyclopedia of Distributed Computing*, vol. 11, 2001.

[27] ePROSIMA, "eProsima Fast RTPS," [Online], https://www.eprosima.com/index.php/products-all/eprosima-fast-rtps.

[28] Red Hat, "What is middleware?" [Online], https://www.redhat.com/en/topics/middleware/what-is-middleware.

[29] G. Pardo-Castellote, "Omg data-distribution service: Architectural overview," in *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings.* IEEE, 2003, pp. 200–206.

[30] "Apex.AI," [Online], https://www.apex.ai/.

[31] Baidu, "Apollo," [Online], http://apollo.auto/.

[32] Y.-P. Wang, W. Tan, X.-Q. Hu, D. Manocha, and S.-M. Hu, "TZC: Efficient inter-process communication for robotics middleware with partial serialization," *arXiv preprint arXiv:1810.00556*, 2018.

[33] "Velodyne lidar products," [Online], 2019, https:velodynelidar.com/products.html.

[34] "Continental ars4-a 77ghz radar," [Online], 2017, https:www.systemplus.fr/reverse-costing-reports/continental-ars4-a-77ghz.radar/.

[35] L. Liu, S. Lu, R. Zhong, B. Wu, Y. Yao, Q. Zhang, and W. Shi, "Computing systems for autonomous driving: State-of-the-art and challenges," *IEEE Internet of Things Journal*, 2020.

[36] D. J. Yeong, J. Barry, and J. Walsh, "A review of multi-sensor fusion system for large heavy vehicles off road in industrial environments," in *2020 31st Irish Signals and Systems Conference (ISSC)*, 2020, pp. 1–6.

[37] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui, "Visual object tracking using adaptive correlation filters," in *2010 IEEE computer society conference on computer vision and pattern recognition*. IEEE, 2010, pp. 2544–2550.

[38] "OpenCV," [Online], https://opencv.org/.

[39] S. Agarwal, A. Vora, G. Pandey, W. Williams, H. Kourous, and J. McBride, "Ford multi-av seasonal dataset," *The International Journal of Robotics Research*, vol. 39, no. 12, pp. 1367–1376, 2020.

[40] S. Hykes *et al.*, "What is Docker?" [Online], https://www.docker.com/whatisdocker/.

[41] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and linux containers," in *2015 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE, 2015, pp. 171–172.

[42] Real-Time Linux, "Preempt rt," [Online], https://rt.wiki.kernel.org/index.php/Main_Page.

[43] D. Casini, T. Blaß, I. Lütkebohle, and B. B. Brandenburg, "Response-time analysis of ros 2 processing chains under reservation-based scheduling," in *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[44] D. Otstott, L. Ionkov, M. Lang, and M. Zhao, "Tcasm: An asynchronous shared memory interface for high-performance application composition," *Parallel Computing*, vol. 63, pp. 61–78, 2017.