Tentacles: A Middleware with Multi-Network Communication Reliability for Vehicle-Infrastructure Cooperative Autonomous Driving

Tianze Wu*‡, Sa Wang*‡, Yungang Bao*‡, Weisong Shi†
*State Key Lab of Processors, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
†University of Delaware, USA
‡University of Chinese Academy of Sciences, Beijing, China
{wutianze, wangsa, baoyungang}@ict.ac.cn, {weisong}@udel.edu

***Abstract*—Vehicle-Infrastructure Cooperative Autonomous Driving (CAD) is a new paradigm of autonomous driving, which relies on the cooperation between intelligent roads and autonomous vehicles. This paradigm has been shown to be safer and more efficient compared to the on-vehicle-only autonomous driving paradigm. Our real-world deployment data indicate that the effectiveness of Vehicle-Infrastructure CAD is constrained by the reliability and performance of commercial communication networks. This paper targets this exact problem and proposes Tentacles, a middleware to achieve high communication reliability between intelligent roads and autonomous vehicles, in the context of Vehicle-Infrastructure CAD. Specifically, Tentacles dynamically matches Vehicle-Infrastructure CAD applications and the underlying communication technologies based on varying communication performance and quality needs. Evaluation results confirm that Tentacles reduces deadline violations by more than 88%, significantly improving the reliability of Vehicle-Infrastructure CAD systems.**

***Index Terms*—Infrastructure-Vehicle Cooperative Autonomous Driving, Middleware, Vehicle to Everything**

I. INTRODUCTION

In the past decade, on-vehicle-only autonomous driving has made a promise to revolutionize our transportation systems [1]. However, the promise has yet to be delivered and we are still far away from the ubiquitous deployment of fully autonomous vehicles.

If we delve into the problem of autonomous driving progress stagnation, each autonomous vehicle has only limited sensing and computing capability, hence it is challenging for autonomous vehicles to gather comprehensive information from their environment, and they are not intelligent enough to operate on their own with only limited information [2]. Vehicle-Infrastructure Cooperative Autonomous Driving(CAD), on the other hand, utilizes infrastructure-side sensing and computing capabilities to empower all autonomous vehicles to be smarter, and furthermore make the whole transportation system more efficient [3].

A typical Vehicle-Infrastructure CAD system consists of System-on-Vehicles (SoVs), System-on-Roads (SoRs), and intelligent transportation cloud system (ITCS) to coordinate the autonomous vehicles to achieve maximum traffic efficiency [4]. During the operation of a Vehicle-Infrastructure CAD system, the SoRs provide local perception results to the SoVs for blind spot elimination and extended perception to improve safety. The ITCS fuses all incoming data to generate global perception and planning information and plans navigation trajectories for each vehicle to achieve optimal traffic efficiency [5]. Although the exchange of information today only involves small data such as traffic light signals, with the emergence of edge computing and new cooperative mechanisms, the transmission of big messages is on the roadmaps [6].

While Vehicle-Infrastructure CAD has been proven to be an effective autonomous driving paradigm, in our real-world Vehicle-Infrastructure CAD deployments, we have identified network communication as the main technical roadblock for reliable cooperative autonomous driving [7]: 1. Current mobile network bandwidth is constraining the uploading of large data samples. 2. Network latency jitters remain high for a considerable portion of a vehicle's trip. One snapshot of the seriousness of the network communication problem can be found in Table I, which shows the latency variation in 4G and 5G networks. These heavy variations greatly impact reliable deployments of Vehicle-Infrastructure CAD applications.

This paper targets this exact problem and proposes *Tentacles*, a middleware to achieve optimal communication reliability in the context of Vehicle-Infrastructure CAD, under real-world multi-network environments. Evaluation results have verified that *Tentacles* effectively reduces deadline violations.

TABLE I
REAL-WORLD NETWORK LATENCY

	4G Mobile Network	5G Mobile Network
Average (ms)	173	26
Min. (ms)	23	19
Max. (ms)	994	96
Standard Dev. (ms)	258	12

* The measurement session is conducted along our test field route on our commercial deployment site with an average speed of 50 km/h.

In summary, this paper makes the following contributions:

- We propose *Tentacles*, a middleware to achieve optimal communication reliability in the context of Vehicle-Infrastructure CAD. Based on the Vehicle-Infrastructure CAD application need, *Tentacles* dynamically allocates the communication workload to the most suitable underlying network.
- We propose a real-time guarantee mechanism in *Tentacles* for not just Vehicle-Infrastructure CAD but all cooperative robots. The proposed mechanism ensures e2e latency by dynamically adjusting the quality of data processing and transmission.
- We have evaluated the effectiveness of *Tentacles* and demonstrate that *Tentacles* reduces deadline violations by more than 88%, hence significantly improving the reliability of Vehicle-Infrastructure CAD systems.

II. BACKGROUND

In this section, we review the technologies related to *Tentacles* in the context of Vehicle-Infrastructure CAD, including computation offloading, networking challenges, and communication middleware.

A. Computation Offloading

As many autonomous driving workloads are extremely compute-intensive, offloading to edge and cloud has been proven an effective mitigation technique [8]. A key problem facing computation offloading is the exchange of external computational resources for minimal communication overhead. Along this line, Neurosurgeon [9] splits DNNs into mobile devices and data centers with neural network layers granularity to find the best partition points of different models through a large amount of profiling. Kayak [10] combines two methods, ship compute and ship data, making full use of the computational resources of both the application server and storage server. [11] demonstrates that combining different input data and model versions can obtain similar latency and accuracy metrics, and then A2 is proposed to adjust data accuracy and model versions to meet latency and accuracy requirements. Data compression, on the other hand, is an effective way to reduce communication overheads. [12], [13], *etc.* propose different compression techniques. Based on various optimization techniques from prior art, *Tentacles* focuses on integrating these optimization techniques to provide real-time guarantees, which are essential to autonomous driving.

B. Networking Challenges in Vehicle-Infrastructure CAD

Networking has been the main challenge since the inception of Vehicle-Infrastructure CAD [4], [7]. First, there is a contradiction between the low reliability of wireless network communication and the high reliability required by autonomous driving. Even with the latest 5G technology, there are still many network fluctuations reported that prevent 5G commercial networks from being reliably utilized for Vehicle-Infrastructure CAD. Second, the uncertainty of mobile networks [14] leads to the difficulty of establishing real-time assurance mechanisms for Vehicle-Infrastructure CAD. Real-time systems require a highly deterministic system environment, so they can only be implemented on a single machine. It remains a challenge to establish a real-time system in a distributed computing environment [15], where the problem of time synchronization of different nodes alone is very difficult to solve. Without a reliable real-time mechanism, Vehicle-Infrastructure CAD could only be a simple aid to autonomous driving. Third, mobile network technology evolves rapidly [16], leading to many communication technologies with varying standards used in Vehicle-Infrastructure CAD [17]–[20]. We believe that the underlying communication technologies will continue to iterate rapidly for the foreseeable future and that the co-existence of multiple communication technologies will be a distinctive feature of Vehicle-Infrastructure CAD. *Tentacles* targets these exact challenges and provides solutions to the above three problems that Vehicle-Infrastructure CAD currently faces.

C. Communication Middleware

Middleware is the management layer between OS and the upper applications [21], with the target to provide common services and capabilities such as task scheduling, inter-process communication, and data management outside of what is offered by OS [22]. Existing autonomous driving solutions [23]–[25] tend to run on a single machine, so the middleware layer plays a limited role in optimizing communication performance. In contrast, Vehicle-Infrastructure CAD is a highly distributed system suffering from high and varying communication latency [26], and because most current Vehicle-Infrastructure CAD applications only need to transmit simple state information, the existing middleware does not optimize the cross-node communication deeply. Existing middleware such as FastDDS [27] does not provide a QoS guarantee. Recently, [28] proposes ERDOS, which guarantees real-time performance by dynamically adjusting the deadlines of computational tasks according to the current environment. However, ERDOS focuses on single-machine deployments and only considers computation time while ignoring communication time. Inspired by ERDOS, *Tentacles* endows cross-node communication and data processing with the ability to dynamically adjust the run time, ultimately providing end-to-end (e2e) latency guarantees.

III. DESIGN OF TENTACLES

This section first discusses the design requirements for the middleware of Vehicle-Infrastructure CAD systems. Then we describe the architectural and technical details of our proposed middleware system. To better explain our design, we take the typical pipeline in Fig. 1 as an example, where the symbols used in this paper are also defined.

A. Design Requirements

In this subsection, we list the design requirements as follows: **R1:** a unified abstraction and interface for applications transparent to the underlying Vehicle to Everything(V2X)

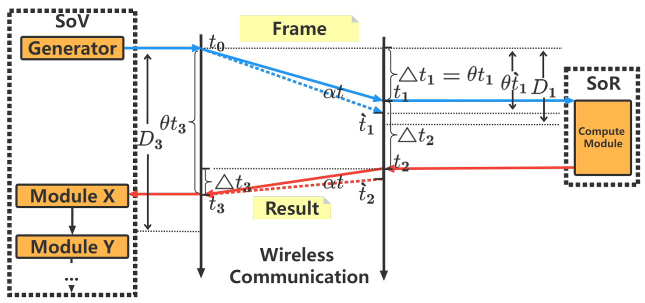

Fig. 1. **Typical Pipeline.** A frame generated at time t_0 marks the beginning of the pipeline. The frame is then sent through a wireless network to SoR. It is received at t_1 and the transmission delay is $\triangle t_1 = t_1 - t_0$. Since the two different devices are not synchronized, the actual timestamps $\hat{t}_1$ & $\hat{t}_2$ recorded by SoR are αt different from the time of SoV. After being handled by the Compute Module in SoR, the result is sent back to SoV at t_2 and is finally received at t_3. We denote the time taken by each computation or transmission task as $\triangle t_n$, the elapsed time from t_0 till current time point t_n as θt_n, and the timeout moment of each θt_n as D_n. In particular, the e2e delay from generating the frame to getting the processing result is D_3.

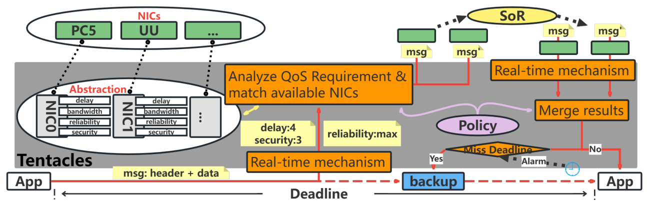

Fig. 2. **Architecture of Tentacles.** *Tentacles* is located between applications and underlying network devices. By abstracting the Network Interface Cards (NICs) on the machine and evaluating their performance (recorded as *Nature*), *Tentacles* is able to match suitable NICs to complete transmission according to the *policy* and QoS requirements. A local backup is also started at the beginning, and its results will be used if the results from SoR are not ready by the deadline.

hardware, and provide applications with the ability to specify communication QoS requirements. **R2:** awareness of changing network environments and capability of adaptive optimization in runtime. **R3:** real-time guarantee under varying network conditions. **R4:** reliable executions tolerant to network performance jitters.

B. *Architecture of Tentacles*

As shown in Fig. 2, *Tentacles* is located between applications and network devices. It has two main responsibilities: one is to provide *QoS guaranteed network communication services* for upper-layer applications using multi-network resources; the other is to dynamically *guarantee the real-time performance* of the entire pipeline according to the current running state. *Policy* in the figure decides how to utilize multi-network resources to meet the QoS requirements when sending messages. Also, *policy* is responsible for merging messages received from different Network Interface Cards (NICs). The task of the real-time mechanism is to analyze the current running state, and then propose QoS requirements for the next communication task (*policy*'s responsibility) or computation task (application's responsibility) based on the analysis results. Last but not least, a local backup task is started at the beginning of the pipeline, it is able to keep the system functioning when the e2e deadline is missed.

C. *Network Abstraction*

To achieve R1 and R2, a software abstraction layer is required to hide the details of the underlying network technologies. *policy* only needs to care about how to utilize the currently available communication resources to meet the QoS requirements, the change of underlying devices should not affect its logic. Tentacles uses *"Nature"* to describe a network connection, which includes four attributes: delay, bandwidth, reliability, and security. Based on prior knowledge, we have currently divided each attribute into five levels. For instance, the *delay* attribute is divided into five levels: >100ms, 50-100ms, 10-50ms, 1-10ms, and <1ms, based on the theoretical average delay from SoV transmitting a data frame (usually 1500B) to SoR. Accordingly, the *delay* value of a 4G connection is 1, and that of 5G is 3. *Nature* is used to help *policy* find which NIC is more suitable for the current task. It's important to note that we do not delve into the selection of *Nature* attributes, the division of attribute levels, or other related aspects in this paper. Rather, we consider this a future work, and we aim to leave room for improvement in the *Nature* mechanism while implementing the Tentacles framework.

D. *Communication* Policy

The network abstraction layer effectively decouples *policy* and network devices. If the *policy* needs to use network functions, it only needs to provide a *Nature* requirement table for *Tentacles* to automatically find the network devices that meet the requirements. The requirement table is calculated by the *policy* based on the QoS requirements of the application (placed in the message header) and real-time mechanism. For example, if an application needs to transmit a piece of sensitive and private information, then its QoS requirement will be extremely security demanding. Then, the *policy* logic may decide to use the NIC with the highest security score. Since *policy* does not depend on underlying devices or upper-layer applications, developers could easily customize *policy* to adapt to other scenarios (R2).

E. *Real-Time Guarantee*

To satisfy R3, we have developed a real-time guarantee mechanism for *Tentacles*. As shown in Fig. 2, the real-time guarantee mechanism is located at the entrance of *Tentacles*. Whenever *Tentacles* receives a message from upper-layer Apps or underlying network, the real-time mechanism will analyze the running state and guide the following operations. Next, we introduce the details of our design, starting with the challenges that the distributed environment poses to real-time guarantee.

Challenges in a distributed environment: The main difference between a distributed environment and a single-node environment is the additional overhead of cross-node communication between modules in a distributed environment.

This poses two fatal problems for real-time guarantee mechanisms: 1. Time is not synchronized between different devices. While there is a lot of work exploring clock synchronization mechanisms such as NTP, PTP, GPST [29]–[31], these mechanisms usually rely on stable network environment or specific devices, making it difficult for them to work in a Vehicle-Infrastructure CAD environment. 2. The uncertainty in communication delay is large. For computation tasks on a single node, the jitters of run time can be released by means such as resource isolation. However, the delay in cross-node communication tasks is difficult to avoid jitters. [15] has made some attempts in the SDN environment, but it is still unable to do anything in the face of an unstable wireless environment like Vehicle-Infrastructure CAD. Time asynchronism combined with uncertain communication delay leads to the system not even being able to determine whether this current moment has timed out.

Our real-time mechanism addresses these challenges in the following ways:

The elapsed time of this round is compared with a reference time to avoid the problem of time asynchronism: Take the scenario in Fig. 1 as an example, because of the time difference αt, SoR considers the current elapsed time to be $\theta\grave{t}_1 = \grave{t}_1 - t_0$, while the actual elapsed time is $\theta t_1 = t_1 - t_0 = \grave{t}_1 - \alpha t - t_0$. Since it is hard to get αt in the Vehicle-Infrastructure CAD environment, we choose to introduce a reference elapsed time $\theta\grave{t}_0^{base}$ to avoid αt. We assume that $\theta\grave{t}_0^{base}$ is the value recorded by SoR in some previous perfect run of the pipeline. Since that run was in the most perfect state, we could evaluate the state of the current run by comparing $\theta\grave{t}_1$ and $\theta\grave{t}_1^{base}$ as follows: $\theta\grave{t}_1 - \theta\grave{t}_1^{base} = \theta t_1 + \alpha t - \theta t_1^{base} - \alpha t = \theta t_1 - \theta t_1^{base}$. In this way, we managed to avoid the influence of αt.

Obtain the reference time by collecting recent performance statistics online: Then, the problem is how we could provide a "perfect" reference time $\theta\grave{t}_1^{base}$? On a single machine, the reference time for a computation module can be obtained by offline profiling. However, as mentioned earlier, the communication latency is very unstable in the wireless network environment of Vehicle-Infrastructure CAD, so online profiling is almost the only option. Through testing, we find that the communication latency jitter of a certain network in the Vehicle-Infrastructure CAD environment is not completely disordered. Take 4G, 5G, and other mobile networks for example, most of the time, their performance is relatively stable. The problem occurs mainly in the following scenarios: 1. When the vehicle is driving away from the communication base station; 2. At the junction of multiple base station coverage areas; 3. A large number of nearby devices cause signal interference or network congestion. Delays in recent periods could reflect the current network status accurately. Therefore, we choose to use recent θt_n data to calculate θt_n^{base} (here and later, we use θt_n instead of $\theta\grave{t}_n$ for convenience). The data being used must meet the requirement that the pipeline on which it is running does not end up with an e2e timeout. It should be noted that the reference time mechanism can be used

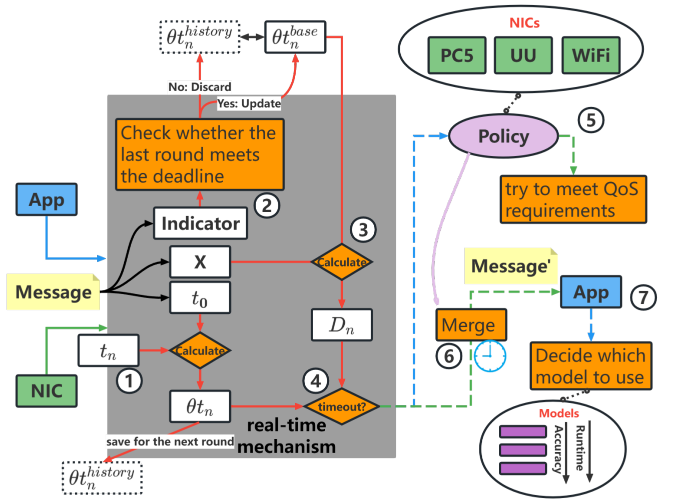

Fig. 3. **Real-time mechanism.** Upon receiving a message, a timestamp t_n is recorded, then *Tentacles* calculates the time already elapsed $\theta t_n = t_n - t_0$ (t_0 is carried by the message header) and saves it for next round (1). Whether the previous round meets the e2e deadline is stored in **Indicator** of the message header, which decides whether to discard the previous $\theta t_n^{history}$ or use it to update the θt_n^{base} (2). The upper limit of θt_n (D_n) is calculated using **X** (carried by the message header) and θt_n^{base} (3). By comparing θt_n and D_n (4), *Tentacles* could tell whether a timeout happens. This information affects the execution of subsequent tasks. If the message comes from the application layer, *Tentacles* would utilize multi-network resources to meet QoS requirements according to a *policy* (5). If the message comes from underlying NICs, *Tentacles* would first merge messages from different NICs into one piece (Message') according to a *policy* (6). The header of Message' contains the info on whether the application needs to take actions to reduce its run time. Then, the application could decide which model to use according to the header of Message' (7).

not only for communication tasks but also for computation tasks.

Implementation details are shown in Fig. 3. θt_n^{base} is the reference time we need, and $\theta t_n^{history}$ is the θt_n of the previous round which is saved in step (1). **Indicator** in the message header tells the real-time mechanism of whether the last round meets the e2e deadline. If not, $\theta t_n^{history}$ does not meet the requirement and will be discarded. Otherwise, θt_n^{base} would be updated as follows: $\theta t_n^{base} = \theta t_n^{base} * k + \theta t_n^{history} * (1-k), k \in [0,1]$. **k** is used to smooth the change curve of the reference time. It should be noted that we do not need to record a θt_n^{base} for each NIC individually; we only care about how much time the transmission module costs.

Determine the current running state: With a "perfect" reference time, the next step is how to judge the current running state based on it. Specifically, if $\theta t_n < \theta t_n^{base}$ which means previous tasks are completed ahead of schedule, it is obvious that the current state is good. However, if $\theta t_n > \theta t_n^{base}$, it is hard to decide whether the current state is poor. Consider the following scenario, a recent e2e run time of the pipeline has been much less than the specified timeout. Thus, even if θt_n is a little larger than θt_n^{base}, it may not cause serious consequences. So we define a threshold ratio **X** as shown in Fig. 3, only when $(\theta t_n - \theta t_n^{base})/\theta t_n^{base}$ exceeds X that we say the current state is bad. We define $D_n = X * \theta t_n^{base}$, D_n

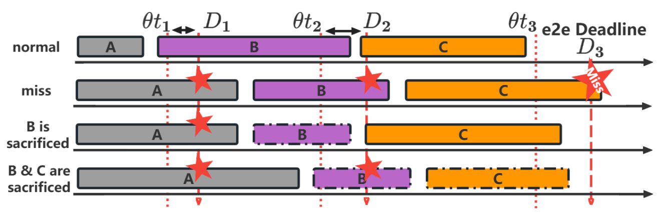

Fig. 4. **Real-time guarantee in pipeline.** A, B, and C are three modules in a pipeline. Line 1 is a normal round, each module finishes without missing its deadline. In Line 2, the delay of A increases which causes the miss of D_1. If no action is taken, the e2e deadline will be missed. Line 3 demonstrates how we deal with the problem. When *Tentacles* detects that D_1 is missed, it will sacrifice B to avoid missing D_2. Then, since D_2 is not missed, C could run normally. If D_2 is still missed as in Line 4, C will be sacrificed too.

as the upper limit of θt_n that the real-time mechanism can tolerate. To reflect the above run-time surplus scenario, we let $X = 1 - \theta t_3^{base}/D_3 + \beta$, where θt_3^{base} is the e2e reference run time of the previous round and D_3 is the preset deadline of the pipeline demonstrated in Fig. 1. The regulation amount β is used for special adjustments.

Guide the running of subsequent modules: If the current running state is poor ($\theta t_n > D_n$), the run time of subsequent modules should be sacrificed to avoid missing the e2e deadline. Fig. 4 visually illustrates this idea of *trade-off*. The line *1* shows a successful pipeline under normal circumstances, with each module completing its task before its respective deadline despite some performance jitter. Module A in the last three lines both encounter performance issues and triggers the alarm. As can be seen from the figure, by sacrificing the next module after the timeout occurs, missing of e2e deadline can be avoided as much as possible.

There are two main approaches to sacrificing the run time depending on the task type of the next module: 1. Computation task. The controlling of computation tasks is mainly done by reducing the quality of the results in exchange for less time consumption. Taking algorithms of machine learning as an example, many models [32]–[34] provide users with different configurations. The better the performance, the slower the model tends to run. 2. The controlling of communication tasks becomes more complicated because of the introduction of the multi-network environment. Here, we also take the down-sampling *policy* before as an example. The original *policy* uses two different networks to send the raw data and the sampled data respectively and will wait for the raw data until D_n on the receiving side. Now, to complete the mission of reducing the transmission delay ordered by the real-time mechanism, the *policy* will have both networks send the sampled data, and it will immediately send the message received from either network to the upper-layer application without waiting.

F. Local Backup and Timeout Handling

Finally, to achieve R4, *Tentacles* provides a local running backup and timeout handling mechanism. As shown in Fig. 2, a local backup loading will take place at the same time as computation offloading. The local backup is generally executed at the minimum processing quality level to save resources. If an e2e timeout occurs, the local result is returned to the application to keep the system running properly.

Besides the backup, timeout handling would ask *Tentacles* to take active measures to accelerate the next round of operation. First, *policy* will be forced to transmit only the sampled data in the next round. Second, *Tentacles* will drastically reduce the value of X: $X = max(X-1, -1)$, so that the next round would impose more stringent delay requirements and the modules would be more likely to sacrifice quality in exchange for speed. Thus, the e2e deadline would not be missed.

IV. Evaluation

The typical pipeline shown in Fig.4 is selected to demonstrate the effectiveness and efficiency of *Tentacles*. Without loss of generality, we choose YoloV5 as the data processing application on SoR.

Experiments are performed in a simulated V2X environment: a machine with Intel 12400F CPU, 64GB DDR4 RAM, RTX 3070 GPU, Intel AX210 NIC is used as the SoR, while a Nvidia Jetson Nano(4-core Arm A57 CPU, 4GB DDR4 RAM, Maxwell GPU) is used as the SoV. The multi-network environment is simulated by two wireless NICs connected to Jetson Nano(Intel 8265NGW together with a USB NIC: NT-AX1800M-X). Transmission delay jitter is created by suspending threads and bandwidth difference is simulated by limiting connection bandwidth through the WiFi router(a wireless access point built on Raspberry Pi cm4). It's worth noting that there are currently no standard hardware requirements for V2X , so our considerations in choosing devices were low cost and easy deployment. Furthermore, variations in hardware devices hardly have any impact on the experimental conclusions.

A. Down-sampling Policy Case

An effective method to alleviate network pressure is decreasing message size. Various strategies can achieve this, and our research employs down-sampling in our experiments. If the outbound message is an image, down-sampling can transform the image from RGB to grayscale or sample it at pixel intervals. Similarly, for point cloud data messages, down-sampling can reduce their density. Consequently, down-sampled data is considerably smaller than the raw data. For applications with QoS requirements concerning transmission delay, this *policy* will choose two networks from currently available networks — those adequately meeting the basic bandwidth requirement for image transmission — with the highest delay scores. Under standard conditions, the network with lesser bandwidth is tasked with transmitting the down-sampled data, while the network with higher bandwidth takes on transmitting the raw data. Both networks resort to transmitting down-sampled data exclusively when the real-time *policy* explicitly mandates it. On the receiving side, if down-sampled data arrives first, the *policy* provisionally stores this data while awaiting the real-time mechanism's directive. Should the real-time mechanism decide not to wait for the arrival of raw data,

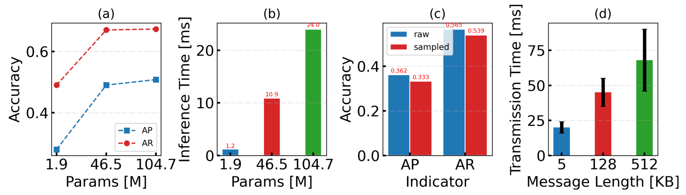

Fig. 5. (a) shows the quality of inference result using model weights with a different number of parameters(YoloV5n has 1.9M, YoloV5l has 46.5M, YoloV5x6 has 104.7M). (b) shows the inference times of these weights. (c) shows the quality of inference result using images with different resolutions(raw image size: 512KB, sampled image size: 128KB) in YoloV5. The performance indicators are Average Precision(AP) and Average Recall(AR), they are measured using YoloV5s with operating parameters: IoU=0.5:0.95, area=all, maxDets=100. (d) shows the transmission time of messages with different lengths, 5KB message is for transferring inference results.

TABLE II
WITH & WITHOUT TENTACLES

	e2e Timeout	* Average e2e run time	Average AP	Average AR
With Tentacles	18 rounds	113msec	0.475	0.631
Without Tentacles	152 rounds	109msec	0.504	0.66

* The average e2e run time is calculated without counting the rounds where timeout occurs.

the *policy* would directly deliver the down-sampled data to the application.

It should be noted that this paper aims to provide a framework for users to utilize multi-network resources, down-sampling *policy* is just an example to show how the framework works. Users are able to define their own *policies* to meet their needs.

B. *Tradeoff Methods*

First, adjusting computing time could be done by using different model weights provided by YoloV5. As shown in Fig. 5 (a), as the number of parameters increases, detection accuracy increases accordingly. At the same time, the cost of achieving higher precision is an increase in inference time, as shown in Fig. 5 (b). Based on this observation, a key knob for *Tentacles* to adjust is the model size, such that we can tradeoff accuracy for a shorter inference time. Second, adjusting the transmission delay could be done by scaling the message length. As shown in Fig. 5 (c) & (d), although using down-sampled images has a 4% to 8% loss in accuracy, the pressure of transferring down-sampled images is much lower. However, since Yolo would eventually scale different lengths of images uniformly into a fixed size before inference, the execution time is not affected by the length of the message.

C. *Evaluation Results*

We evaluated whether *Tentacles* could reduce occurrences of timeouts. We executed the pipeline for 300 rounds with and without *Tentacles*. We randomly injected delays into computation(θt_2) or communication(θt_1) tasks, and the distribution of delay data conforms to the standard normal distribution, with a mean value of 10. As shown in Table II, the experimental results confirm that the number of timeouts with *Tentacles* is 88% fewer, thus verifying *Tentacles*'s capability to greatly improve communication reliability in Vehicle-

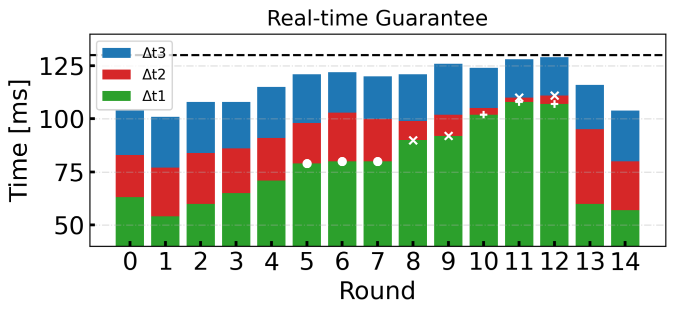

Fig. 6. We vary the transmission time of images($\triangle t_1$) and show how *Tentacles* respond to the changes and avoid a timeout. The e2e timeout time is set to 130ms (marked by the black dashed line), and the parameter k for updating θt_n^{base} is set to 0.3. The white dot in the figure marks the positions where $\theta t_1 = D_1$, 'x' marks the positions where $\theta t_1 > D_1$ or $\theta t_2 > D_2$, and '+' marks the positions where θt_1 is much greater than D_1.

Infrastructure CAD. In the presented case, the entire pipeline process is relatively short, thus providing less room for flexibility. In a realistic autonomous driving system, the end-to-end execution process might comprise multiple computing and communication tasks [35], thereby enhancing Tentacles's ability to compensate for latency. It is critical to note that developers need to adjust algorithms to harness the capacity of the Tentacles mechanism to modulate runtime. In fact, many algorithms have the potential for such transformation [28], [36], [37].

A typical process is presented in Fig. 6. In the first three rounds, the system is in a normal state, where all the modules are outputting the highest quality results. Then the image transmission delay increases. In rounds 3 & 4, θt_1 is still within D_1, so all tasks continue to run normally. However, with the reference e2e run time (θt_3^{base}) increases (X decreases), *Tentacles* becomes more sensitive to delay jitters. Next, in rounds 5-7, raw image transmission time exceeds D_1. According to the down-sampling *policy* introduced in Section III, before using the sampled data, *Tentacles* would wait for the raw image until D_1. Therefore, at the white dots in the figure, although the system ultimately uses the sampled data, the image transmission time is exactly equal to D_1.

Then, we increase the latency of the other network to make the delay of transmitting sampled data exceed D_1 in rounds 8-12. As a result, D_1 is missed, and *Tentacles* has to sacrifice the computation module to reduce time consumption. The computation module will select the appropriate weights according to the timeout degree of the transmission delay. In rounds 10-12, the computation module selects the weight with fewer parameters than in rounds 8-9 because of the serious timeout of the transmission task. And in rounds 11 & 12, D_2 is missed, so the third module is sacrificed too.

D. *X jitter problem*

Experimental results show that the continuous timeouts can be effectively eliminated by adjusting X to dynamically limit the execution quality in the next round. However, this brings the X jitter problem: After round i misses the deadline, the timeout handler adjusts X to a negative value, and round i+1 finishes on time. But because round i+1 takes so little time (caused by drastically reduced quality), X will become too large again. And then round i+2 will run at a high quality, which is easy to cause timeout again since the environment is likely to be unstable. Our solution is: when X is negative, completing the pipeline on time in each round will add 0.2 to X until X is positive. Experimental results verify that this method can effectively eliminate the occurrence of X jitter.

V. Conclusion

This paper proposes Tentacles, a middleware to achieve high communication reliability in Vehicle-Infrastructure CAD. With the aim of achieving optimal Vehicle-Infrastructure CAD reliability, Tentacles exploits the characteristics of the existing multi-network communication environments to dynamically match Vehicle-Infrastructure CAD applications to the underlying communication networks. Tentacles is designed to be lightweight, high-performance, reliable, and easy to deploy. Evaluation results have confirmed the effectiveness of Tentacles, which reduces deadline violations by more than 88%. As a next step, we plan to extend Tentacles to more usage scenarios beyond Vehicle-Infrastructure CAD, including vehicle-to-vehicle communication, and cooperative robots.

References

[1] S. Liu, L. Li, J. Tang, S. Wu, and J.-L. Gaudiot, "Creating autonomous vehicle systems," *Synthesis Lectures on Computer Science*, vol. 8, no. 2, pp. i–216, 2020.

[2] 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.

[3] S. Liu and J.-L. Gaudiot, "self-driving cars work better with smart roads: intelligent infrastructure makes autonomous driving safer and less expensive," *IEEE Spectrum*, 2022.

[4] S. Liu, B. Yu, J. Tang, and Q. Zhu, "Towards fully intelligent transportation through infrastructure-vehicle cooperative autonomous driving: challenges and opportunities," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 1323–1326.

[5] S. Liu, J. Wang, Z. Wang, B. Yu, W. Hu, Y. Liu, J. Tang, S. L. Song, C. Liu, and Y. Hu, "Brief industry paper: The necessity of adaptive data fusion in infrastructure-augmented autonomous driving system," in *2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2022, pp. 293–296.

[6] J. Peeck, M. Möstl, T. Ishigooka, and R. Ernst, "A middleware protocol for time-critical wireless communication of large data samples," in *2021 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2021, pp. 1–13.

[7] S. Liu, B. Yu, J. Tang, Y. Zhu, and X. Liu, "Communication challenges in infrastructure-vehicle cooperative autonomous driving: A field deployment perspective," *IEEE Wireless Communications*, vol. 29, 2022.

[8] S. Liu, J. Tang, Z. Zhang, and J.-L. Gaudiot, "Computer architectures for autonomous driving," *Computer*, vol. 50, no. 8, pp. 18–25, 2017.

[9] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1, pp. 615–629, 2017.

[10] J. You, J. Wu, X. Jin, and M. Chowdhury, "Ship compute or ship data? why not both?" in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, 2021, pp. 633–651.

[11] J. Jiang, Z. Luo, C. Hu, Z. He, Z. Wang, S. Xia, and C. Wu, "Joint model and data adaptation for cloud inference serving," in *2021 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2021, pp. 279–289.

[12] K. Du, A. Pervaiz, X. Yuan, A. Chowdhery, Q. Zhang, H. Hoffmann, and J. Jiang, "Server-driven video streaming for deep learning inference," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 557–570.

[13] S. Yao, J. Li, D. Liu, T. Wang, S. Liu, H. Shao, and T. Abdelzaher, "Deep compressive offloading: Speeding up neural network inference by trading edge computation for network latency," in *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, 2020, pp. 476–488.

[14] B. Lucia, V. Balaji, A. Colin, K. Maeng, and E. Ruppel, "Intermittent Computing: Challenges and Opportunities," in *2nd Summit on Advances in Programming Languages (SNAPL 2017)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), B. S. Lerner, R. Bodík, and S. Krishnamurthi, Eds., vol. 71. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, pp. 8:1–8:14. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2017/7131

[15] H.-Y. Choi, A. L. King, and I. Lee, "Making dds really real-time with openflow," in *2016 international conference on embedded software (EMSOFT)*. IEEE, 2016, pp. 1–10.

[16] M. N. Ahangar, Q. Z. Ahmed, F. A. Khan, and M. Hafeez, "A survey of autonomous vehicles: Enabling communication technologies and challenges," *Sensors*, vol. 21, no. 3, p. 706, 2021.

[17] J. San Martín, A. Cortés, L. Zamora-Cadenas, and B. J. Svensson, "Precise positioning of autonomous vehicles combining uwb ranging estimations with on-board sensors," *Electronics*, vol. 9, no. 8, p. 1238, 2020.

[18] D. Parthasarathy, R. Whiton, J. Hagerskans, and T. Gustafsson, "An in-vehicle wireless sensor network for heavy vehicles," in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2016, pp. 1–8.

[19] A. Fitah, A. Badri, M. Moughit, and A. Sahel, "Performance of dsrc and wifi for intelligent transport systems in vanet," *Procedia Computer Science*, vol. 127, pp. 360–368, 2018.

[20] R. Molina-Masegosa and J. Gozalvez, "Lte-v for sidelink 5g v2x vehicular communications: A new 5g technology for short-range vehicle-to-everything communications," *IEEE Vehicular Technology Magazine*, vol. 12, no. 4, pp. 30–39, 2017.

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

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

[23] 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.

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

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

[26] T. Wu, B. Wu, S. Wang, L. Liu, S. Liu, Y. Bao, and W. Shi, "Oops! it's too late. your autonomous driving system needs a faster middleware," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 7301–7308, 2021.

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

[28] I. Gog, S. Kalra, P. Schafhalter, J. E. Gonzalez, and I. Stoica, "D3: a dynamic deadline-driven approach for building autonomous vehicles," in *Proceedings of the Seventeenth European Conference on Computer Systems*, 2022, pp. 453–471.

[29] Y. Geng, S. Liu, Z. Yin, A. Naik, B. Prabhakar, M. Rosenblum, and A. Vahdat, "Exploiting a natural network effect for scalable, fine-grained clock synchronization," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018, pp. 81–94.

[30] Y. Li, G. Kumar, H. Hariharan, H. Wassel, P. Hochschild, D. Platt, S. Sabato, M. Yu, N. Dukkipati, P. Chandra *et al.*, "Sundial: Fault-tolerant clock synchronization for datacenters," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 1171–1186.

[31] S. Liu, B. Yu, Y. Liu, K. Zhang, Y. Qiao, T. Y. Li, J. Tang, and Y. Zhu, "Brief industry paper: The matter of time—a general and efficient system for precise sensor synchronization in robotic computing," in *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2021, pp. 413–416.

[32] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015.

[33] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, “Ssd: Single shot multibox detector,” in *European conference on computer vision*. Springer, 2016, pp. 21–37.

[34] M. Tan, R. Pang, and Q. V. Le, “Efficientdet: Scalable and efficient object detection,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 10 781–10 790.

[35] S. Liu, B. Yu, N. Guan, Z. Dong, and B. Akesson, “Real-time scheduling and analysis of an autonomous driving system,” *Proc. RTSS Ind. Challenge Problem*, pp. 1–47, 2021.

[36] R. Han, Q. Zhang, C. H. Liu, G. Wang, J. Tang, and L. Y. Chen, “Legodnn: block-grained scaling of deep neural networks for mobile vision,” in *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, 2021, pp. 406–419.

[37] L. Liu, Z. Dong, Y. Wang, and W. Shi, “Prophet: Realizing a predictable real-time perception pipeline for autonomous vehicles,” in *2022 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2022, pp. 305–317.