A Novel ROS2 QoS Policy-enabled Synchronizing Middleware for Co-simulation of Heterogeneous Multi-Robot Systems

Emon Dey†, Mikolaj Walczak†, Mohammad Saeid Anwar†, Nirmalya Roy†,
Jade Freeman§, Timothy Gregory§, Niranjan Suri§, Carl Busart§
†*University of Maryland, Baltimore County, USA*, §*DEVCOM Army Research Lab, USA*
†{edey1, mwalcza1, wy71697, nroy}@umbc.edu,
§{niranjan.suri, timothy.c.gregory6, jade.l.freeman2, carl.e.busart}.civ@army.mil

***Abstract*—Recent Internet-of-Things (IoT) networks span across a multitude of stationary and robotic devices, namely unmanned ground vehicles, surface vessels, and aerial drones, to carry out mission-critical services such as search and rescue operations, wildfire monitoring, and flood/hurricane impact assessment. Achieving communication synchrony, reliability, and minimal communication jitter among these devices is a key challenge both at the simulation and system levels of implementation due to the underpinning differences between a physics-based robot operating system (ROS) simulator that is time-based and a network-based wireless simulator that is event-based, in addition to the complex dynamics of mobile and heterogeneous IoT devices deployed in a real environment. Nevertheless, synchronization between physics (robotics) and network simulators is one of the most difficult issues to address in simulating a heterogeneous multi-robot system before transitioning it into practice. The existing TCP/IP communication protocol-based synchronizing middleware mostly relied on Robot Operating System 1 (ROS1), which expends a significant portion of communication bandwidth and time due to its master-based architecture. To address these issues, we design a novel synchronizing middleware between robotics and traditional wireless network simulators, relying on the newly released real-time ROS2 architecture with a masterless packet discovery mechanism. Additionally, we propose a ground and aerial agents' velocity-aware customized QoS policy for Data Distribution Service (DDS) to minimize the packet loss and transmission latency between a diverse set of robotic agents, and we offer the theoretical guarantee of our proposed QoS policy. We performed extensive network performance evaluations both at the simulation and system levels in terms of packet loss probability and average latency with line-of-sight (LOS) and non-line-of-sight (NLOS) and TCP/UDP communication protocols over our proposed ROS2-based synchronization middleware. Moreover, for a comparative study, we presented a detailed ablation study replacing NS-3 with a real-time wireless network simulator, EMANE, and masterless ROS2 with master-based ROS1. Our proposed middleware attests to the promise of building a large-scale IoT infrastructure with a diverse set of stationary and robotic devices that achieve low-latency communications (12% and 11% reduction in simulation and reality, respectively) while satisfying the reliability (10% and 15% packet loss reduction in simulation and reality, respectively) and high-fidelity requirements of mission-critical applications.**

***Index Terms*—IoT, Heterogeneous multi-robot systems, Gazebo, NS-3, EMANE, TCP, UDP, Synchronization**

I. Introduction

In the field of robotics and wireless sensor networks, multi-agent dynamic systems play a vital role due to their ability in forming large, interconnected networks with coordination among agents that make them an integral part of a variety of robotic and smart city IoT applications [1], [2]. For example, Unmanned Aerial Vehicle (UAV) systems are being increasingly used in a broad range of applications requiring extensive communications, either to collaborate with other UAVs [3], [4] or with Unmanned Ground Vehicles (UGV) [5] and WSNs [6], [7]. Particularly in a smart city environment, the presence of heterogeneous wireless sensor networks, aerial and ground vehicles coordinating with each other is going to be an essential feature in the future. The mutual information transfer between mobile aerial drones, ground robots through IoT networks can help to make intelligent decisions in a disaster-prone area in critical time.

Synchronized communications across UAVs, UGVs, and IoT networks can let each deployed robotic asset and IoT device gain situation awareness in smart environments, smoothly execute civilian commands, and meet QoS requirements for high-fidelity military applications. Due to algorithm parameter fine-tuning, deploying such systems directly in the real world may have negative effects [8]. Thus, to identify and resolve research and implementation issues, a cross-cutting robotic and IoT system must be simulated beforehand using appropriate modeling toolkits. Existing research [8] has mainly simulated such an environment among numerous agents, such as two aerial systems, to pose communication issues between homogeneous robots employing Robot Operating Systems 1 (ROS1). While ROS1 was the defacto standard for robotics middleware, to meet the requirements of real-time distributed embedded systems, ROS2 has been recently introduced with its Data Distribution Service (DDS) capability [9], [10]. DDS, an industry-standard real-time communication system, meets distributed system resilience, fault-tolerance, security, and scalability requirements [11], [12]. DDS with underlying masterless publish-subscribe data transport architecture in ROS2 meets real-time distributed embedded system requirements by being scalable to various operating systems and adaptable to transport layer configurations like deadline, fault-tolerance, and process synchronization [13].

Besides ROS1's master-based publish-subscribe architecture, which contrasts with ROS2's masterless packet discovery

mechanism for real-time distributed embedded systems, the main challenge is synchronizing two simulators with different operating principles [14], [15]. A physics simulator simulates physical robots' interactions with their environs. Network simulators estimate deployed agent wireless network communication performance. Thus, one of our main goals is to discover and assess the communication issues between a Physics-based ROS simulator toolkit that is time-based and a Network-based event-based simulator. We can create a robust perception-action-communication framework that simulates physical surroundings and network traffic by integrating ROS2's real-time characteristics with a masterless packet discovery approach. Representing robotic asset interactions with physical environments and synchronizing packet-level communications amongst numerous agents are the biggest hurdles to creating a seamless networking architecture between robotics platforms and the IoT. These features are needed to accurately imitate reality and fulfill high-fidelity robotic and IoT applications in demanding conditions.

Needless to say, building multi-agent systems is strenuous. Therefore, resilient and coordinated communications across heterogeneous simulated and real systems can make our proposed middleware framework less error-prone and more accurate, reliable, fault tolerant and inter-operable. For example, SynchroSim [16], FlyNetSim [17], ROS-NetSim [18], CORNET [19], CPS-Sim [20], and RoboNetSim [3] are some of such kinds of works in existing literature that have been implemented in ROS1 using AirSim [21], ARGoS [22], Gazebo [23] as physics simulators and OMNeT++ [24], NS-2 [25], NS-3 [26] as network simulators. Nonetheless, some of the major drawbacks that exist in those works are: *(i)* the proposed systems are compatible with either UAVs or UGVs, but not with heterogeneous agents such as a mix of both UAVs and UGVs under the same network, *(ii)* suffer from low co-simulation speed, *(iii)* give rise to floating-point arithmetic error during the time synchronization, *(iv)* difficult to set up proper window size for reliable packet transmissions in presence of diverse agents with different velocities. *(v)* lose synchronization when the relative speed varies between the simulators, and *(vi)* the network does not self-heal or reconfigure to maintain seamless and uninterrupted connections in presence of node loss or adversarial attacks.

Notably there are real-time network emulators, for example, EMANE [27] which can help solve some of the aforementioned issues, but we empirically observed that it has a comparatively higher packet delay and is not quite matured enough to be integrated with Robot Operating Systems (ROS). Motivated by this, we propose a novel synchronizing middleware[1] that follows the Data Distribution Service (DDS) architecture [13] and the TCP/IP protocol in order to facilitate the development of an appropriate ROS2-compatible synchronizing middleware. To the best of our knowledge, it is the first endeavor to develop a synchronizing middleware for a real-time masterless ROS2 environment. We postulate a dynamic sliding window-based QoS policy for ROS2 to minimize the average packet loss and latency in presence of heterogeneous ground and aerial agents with disparate velocities. We evaluate the performance of our proposed algorithm after deploying it on a cluster that is comprised of real-world UAVs – Duckiedrone and UGVs – TurtleBot. The specific research contributions are summarized below.

- **Reliable and faster synchronization middleware for co-simulation of multi-agent systems:** We design a robotic and wireless network simulators agnostic co-simulation middleware for heterogeneous multi-agent communications. We integrate a customized QoS policy for Data Distribution Service (DDS) on Transmission Control Protocol (TCP) within the synchronization middleware. We implement our algorithm on real-time robotic operating system 2 (ROS2), namely, *Foxy*. We leverage the low latency virtue of ROS2 along with the TCP/IP protocol to improve the reliability of multi-agent communication paradigm with our proposed customized sliding window-based TCP algorithm. Our proposed QoS policy can vary the window size dynamically while considering the velocity differences between the ground and aerial agents in a heterogeneous setting to minimize the process synchronization and communication delays.
- **System implementation using off-the-shelf real robotic devices:** We extend our simulation based synchronization middleware framework into the real world robots to witness the various network performance metrics such as probability of packet loss and average network latency in case of actual deployment. We assemble commercially available UASs and UGVs robots – Duckiedrone and TurtleBot3 Burger and form a cluster using one UAS and two UGVs to implement our proposed multi-agent synchronizing middleware and evaluate the communication performance. In particular, we record *ROSbag* of uniform size containing information regarding the physical robot states in the wild and send it between the other robotic agents to record the network performance metrics.
- **Empirical evaluation considering different communication scenarios:** We perform extensive performance evaluations, taking into account both line-of-sight (LOS) and non-line-of-sight (NLOS) communication scenarios on the basis of probability of packet loss, and average packet delay. We employ Gazebo as Physics simulator and NS-3 as network simulator at the simulation level to report the network synchronization and performance results. We also perform an ablation study with a real-time network simulator *EMANE* and compare our proposed masterless ROS2 synchronization middleware with master-based ROS1 synchronization middleware. Experimental results both at the simulation and system levels attest that our proposed synchronizing middleware in ROS2 integrated with Transmission Control protocol (TCP) outperforms the traditional ROS1 systems in terms of ensuring fewer packet losses (10% reduction in simulation and 15% in real) and faster

[1]https://github.com/Emon-dey/RobSenCom

data transmissions rates (12% increase in simulation and 11% in real).

II. Challenges

Implementation of a multi-agent heterogeneous system raises a number of challenges due to its inherent nature. The need for prior simulation to get an estimation of the actual performance adds specific software-centric challenges along with deployment issues. The challenges we have addressed in this works are categorized into two sub-modules and illustrated in figure 1. The left side of the figure signifies the simulation level challenges. The two segments inside of it are the two most fundamental research questions in simulating multi-agent systems and working with simulators with different paradigms. When measuring the communication performance among the agents in a simulated environment, the integration of a physics simulator and a network simulator is necessary. However, the issue is that the physics simulator gets updated over time, and the network simulator works based on available events. So, to run those two simulators synchronously (co-simulation), a synchronizing middleware is needed. Also, if heterogeneity exists among the robotic agents, it becomes even more complex to simulate such a scenario. The specification disparity and different capabilities of data packet handling are some of the major reasons for that. Additionally, during

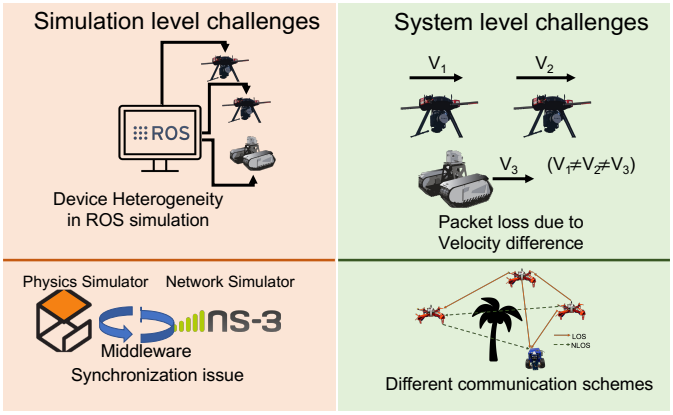

Figure 1. Illustration of the possible hardware and software centric issues while working with multi-agent robotic implementation.

the deployment with actual robotic agents, if the velocity difference between agents exceeds a certain threshold, it can cause severe degradation in the communication performance. Also, the presence of obstacles between the communicating agents (non-line-of-sight scenario) can be considered as another hindrance for the deployment point-of-view.

III. Background and Related Work

In this section, we will provide a brief summary of the strategies and choice of simulators from the previous works that have explored this synchronizing issue.

A. Simulation Tools

Among the robotic simulators, AirSim [21], and ANVEL [15], [28] provides such a toolkit by combining popular graphical representation methods. While both AirSim and ANVEL have significant simulation capabilities, they struggle to create large-scale complex visually rich environments that are more realistic in their representation of the real world, and they have fallen behind various advancements in rendering techniques made by platforms such as Unreal Engine or Unity [29]. Another such stable simulator is Gazebo [14] has a modular design that allows for the implementation of various physics engines, sensor models, and the development of 3D worlds.

From network simulators point-of-view, OMNeT++ [24] is an object-oriented and modular discrete event network simulation framework. OMNeT++ also supports parallel distributed simulation, and inter-participant communications can be achieved via a variety of approaches. However, we chose to utilize event-based simulator, NS- 3 [26], in which the scheduler normally runs the events sequentially without synchronization with an external clock. The NS-3 simulator shows all of the network models that make up a computer network and also the features offered by this simulator greatly overlaps with our requirement. In contrast of the prior works, we target a special communication scenario for dynamic multi-agents systems where the velocity differences of the agents may impact the performance. Simulating such scenario require high-fidelity physics simulator and a network simulator with flexible network topology design capability.

B. Synchronization Middleware

The working principles of existing middleware for the synchronization purpose varies from time-stepped to sliding window based protocol. In some works, a scheduler was also introduced to decrease latency and a improved version of it named variable-stepped was adopted. In table I, we have summarized such existing approaches along with the probable issues may arise if we choose to utilize them in our use case. Another aspect in this topic is the choice of communication protocol for middleware testing. RUDP [30] provides a solution for real-time embedded systems with transmission speed and reliability criteria that TCP and UDP have not been able to achieve. In a real-world demonstration, the claim that the suggested technique can give a faster throughput than TCP without experiencing packet loss has been validated. But the issue with RUDP is that, Robot Operating System hasn't yet finalized the compatibility test with it. In this work, UAV and UGV components are being simulated with 3D visualization using Gazebo, while the network infrastructure is being provided by NS-3 and middleware is being developed for the creation of an inter-simulation data-path with time and position synchronization at both ends using our co-Simulation of robotic networks. The whole operation will be conducted under a ROS2 environment.

Table I
SUMMARY OF PRIOR WORKS ON DEVELOPING SYNCHRONIZING MIDDLEWARE BETWEEN PHYSICS AND NETWORK SIMULATORS.

Synchronization Method	Working Procedure	Issues
Time-stepped [3], [31]	Used advanced network simulation tools together with robotic simulators. At predefined thresholds, the simulators halt their individual simulations to share data with each other.	The network simulator isn't capable to take into account the presence of obstacles. Data exchange issues.
Time-stepped with scheduler [17]	Common sampling period to be used by both simulators.	If the network simulator runs faster than the physics simulator, the network events must be buffered in a cache and wait to be processed until the next sampling time.
Variable-stepped [19]	A global event scheduler keeps track of all the events from both simulators and schedules based on their timestamps, allowing only one event process to operate at a time.	Float-point arithmetic error.
Global event driven [20]	The server is forced to reproduce the exact same time-steps.	Higher latency during co-simulation.
Sliding window [8], [16]	Capture and track network events over the window period and allow the network simulator to step up to the end of the window.	Difficulty in maintaining performance for multi-agent system with disparate velocity.

IV. METHODOLOGY

The design basics of our proposed synchronizing middleware, modification of conventional TCP/IP architecture, integration procedure with masterless DDS approach are described in the section.

A. Synchronization Middleware Design through Modifying the TCP/IP Architecture

The sliding window-based synchronization strategy [16] is the foundation of our algorithm design for the masterless multi-agent scenario. When dealing with a multi-agent system, the choice of window size is the most important design parameter that you can choose. In such a case, a fixed window size at all points will not be relevant since different agents may perform better with different radio frequencies, and there may also be differences in terms of the hardware specifications. Moreover, there may be differences in the amount of available memory. Our system begins with a value for the window size w that has been manually initiated and then takes into account the total number of agents T that are present for a particular scenario. It chooses the appropriate publisher (P) and subscriber (S) for the data transfer event based on the information provided by both parties. After that, it performs a stringent check on the difference in velocity between the participating agents in order to accurately calculate an appropriate window size for a particular communication event. The process of velocity difference calculation can be found in 1. When it comes to the transmission scenario between a ground vehicle and an aerial vehicle, this technique becomes more apparent. We decided to use the packet loss probability (L_p), and average delay (PD_a), which are two of the most important parameters to monitor while sending valuable information, in the operating moment of the middleware in order to keep the level of information that is transmitted at an acceptable point. This was done in order to keep the level of information that is transmitted at a satisfactory point. In order to inherently determine the likelihood of packet loss as well as the average delay and report it within a synchronization window, the following equations are used:
Packet loss probability,

$$ {}^{i}_{n}Lp = 1 - \frac{{}^{i}_{n}D_s}{{}^{i}_{n}D_p} \tag{1} $$

Here, ${}^{i}_{n}D_s$ is the number of data packets delivered to the total number of subscribers and ${}^{i}_{n}D_p$ is the number of data packets transmitted from the publishers. The average value is reported after the data transmission is complete between each publisher-subscriber pair.

Average packet delay,

$$ PD_a = D_{pr} + D_t + D_{pg} + D_q \tag{2} $$

Here, D_{pr} is processing delay D_t is transmission Delay, and D_{pg}, D_q represent propagation, and queuing delay respectively.

Algorithm 1 Pseudo-code for Integrating Proposed QoS Policy into Synchronizing Middleware

Input: Data packets D, Total number of agents N, Window size w, Velocity of agents V, Packet loss threshold Q_T, Packet loss value for each event Q
Output: Packet loss probability Lp, and average delay PD_a
Simulation Initialization: Publisher P and subscriber S agents ip are determined where $P, S \in N$, initialize physics simulator time = 0, window size w
Data transmission and Update co-simulation: Select topic of interest and transmit data packets from publisher
Update the timestamp t = t
if there is a network event **then**
    Calculate Q for the event using equation 1
    if $Q > Q_T$ **then**
        Adapt the window size for that event until $Q \leq Q_T$
        Sliding Window adaptation: Acquire the velocity of Publisher V_p and Subscriber V_s in meters/sec
Calculate the required adaptation of sliding window,

$$ w_a = w \pm \lceil |(V_p - V_s)|/1000 \rceil \tag{3} $$

    end if
    Report PD_a **and** L_p **upon synchronization:** Calculate average delay and packet loss probability for the synchronized event using equation 2 and 1 respectively
    Timestamp update: *Update* $t = t + w_a$ and request for next window to iterate the process
end if

Updating the co-simulation steps. Another essential part of our process is the development of a mutual information update method for both the physics simulator and the network simulator. At the outset of the simulation, the Physics simulator establishes two essential characteristics of the agents that will be employed for a particular communication round. These characteristics are the velocity of the agents and the distance that separates them as determined by positional coordinates. The information that the network simulator stores includes the communication scheme that will be used (for reasons related to reliability, we have decided to use a TCP/IP-based approach rather than an UDP-based approach), the number of packets, the length of each packet, and the IP addresses of both the publisher and subscriber agents.

After that, the Physics simulator and the network simulator are iterated upon while using a single initially fixed window size. When the simulation is advanced, the Physics simulator sends the information about distance and velocity to the network simulator, and the network simulator uses that information to compute the chance of packet loss. When the value of the loss is compared to a predetermined threshold, the result is sent to the display. In the event that this is not the case, the information is sent back to the synchronizing module, where it is processed, and the initial window size is modified in accordance with the method 1. This process will continue until a result is obtained for this particular round that is deemed to be satisfactory, after which the initialization procedure will begin once again with a fresh group of agents.

B. Theoretical Analysis of QoS Policy Design

We design a mathematical analysis to demonstrate how the varying window size is going to impact latency, packet loss when there is a velocity difference among the agents. The three major assumptions for our analysis are given below:

- Steady-state approximation [32]
- Packet transmission follows Little's Law
- Data transmission rate can be varied with the velocity difference between the robotic agents

We consider a well-defined communication channel with known characteristics, such as bandwidth, packet size to be transmitted. Let $L(t)$ denote the latency of data transmission at time t. Assuming that the network is in a steady-state and the number of nodes is large, we can use the steady-state approximation to derive $L(t)$. Using probability theory, the latency of data transmission can be derived as follows:

$$L(t) = \frac{E(W_i)}{\mu_i} \tag{4}$$

where E is the expected value, W_i is the transmission window of node i, and μ_i is the transmission rate of node i. The transmission rate is given by:

$$\mu_i = W_i \Delta V_i \tag{5}$$

where ΔV_i is the velocity difference between the sender and receiver nodes for node i. The expected transmission window size can be written as:

$$E(W_i) = \lambda_i E(T_i) \tag{6}$$

where λ_i is the data generation rate of node i, and $E(T_i)$ is the expected time it takes for node i to send a packet. Using Little's Law, we can write:

$$E(T_i) = L(t) + S_i \tag{7}$$

where S_i is the expected time it takes for node i to process a packet. The expected latency of data transmission can then be written as:

$$L(t) = \frac{E(W_i)}{\mu_i} = \frac{\lambda_i (L(t) + S_i)}{W_i \Delta V_i} \tag{8}$$

Rearranging the terms, we obtain:

$$L(t) = \frac{\lambda_i S_i}{W_i \Delta V_i - \lambda_i} \tag{9}$$

To minimize the latency of data transmission, we need to find the optimal transmission window size for each node. This can be done by minimizing the expected latency of data transmission subject to the constraint that the probability of packet loss is below a certain threshold. Let Q_T denote the threshold probability of packet loss. We need to find the transmission window size W_i that minimizes the expected latency of data transmission subject to the constraint:

$$P(t) \leq Q_T$$

where $P(t)$ is the packet loss probability of time t Using Lagrange multipliers (λ_L), we can write the objective function as:

$$L = \frac{\lambda_i S_i}{W_i \Delta V_i - \lambda_i} + \lambda_L (P(t) - Q_T) \tag{10}$$

Solving this inequality constraint based Lagrange equation using partial differentiation we find a valid solution at $W_i = \frac{S_i}{\Delta V_i \lambda_i}$ for the condition $S_i > \frac{\lambda_i^2}{\lambda_i^2 - 1} \Delta V_i$. We have performed a partial second derivative test to confirm the solution corresponds to the local minima not a saddle point.

V. Simulation Setup

As stated in earlier sections, we have chosen Gazebo and NS-3 as physics and network simulators respectively for our experimentation. Here, we provide the brief description of the simulators, detailed simulation setup and analysis on the achieved simulation results.

A. Deploying Synchronization Middleware on DDS using ROSbridge

We use the ROSbridge idea to combine the functionalities of Data Distribution Service (DDS) with our suggested middleware solution. To ensure speedier data transmission, the communication sockets are developed in C, which, unlike ROS1, is a well-supported platform in the ROS2 environment. Some of the operational robots in our configuration do not have ROS2 functionality by default. To prevent the time-consuming re-installation, we built a bridge between two ROS versions to exchange rostopics. The subject flow of ROSbridge is depicted in figure 3. We used DDS to further process the rostopics from a communication standpoint. DDS uses a publish-subscribe framework to send and receive data, events, and commands between nodes. Nodes that generate information (publishers) generate "topics" and "samples". DDS delivers

samples to subscribers who express an interest. DDS is in charge of message addressing, delivery, and flow control. Any node has the ability to publish, subscribe, or both. We were able to simplify distributed application network programming thanks to the DDS publish-subscribe idea. DDS requires less design time for interactions between nodes. It can also add another essential security feature because applications do not require knowledge about the existence or location of other participants. DDS allows users to configure discovery and behavior approaches based on QoS factors. By permitting anonymous message interchange, DDS promotes modular, well-structured programs. Figure 2 depicts the functionality of DDS within the ROS2 architecture and its differences from the ROS1 design. The specific ROS2 settings we have used for this work are highlighted in red.

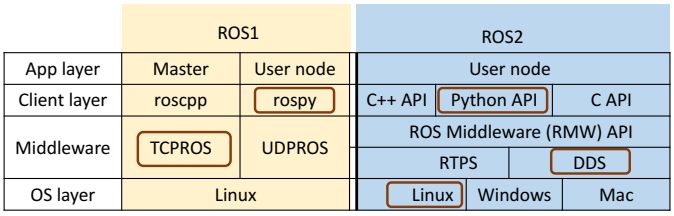

Figure 2. Comparison between the two ROS versions.

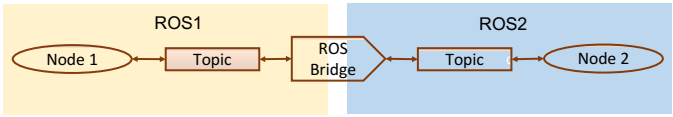

Figure 3. ROSbridge implementation to import the ROS1 topics to ROS2 environment.

B. Selected Simulation Tools

Keeping our specific simulation requirements in mind, we have chosen Gazebo and NS-3 as physics and network simulators respectively.

Gazebo. The fully open-source physics simulator Gazebo is often nicknamed a robotics simulator due to its vast libraries of robot models, sensors, and compatibility with ROS. Being integrated with the ODE physics engine and its ability to be integrated with several other physics engines, allows users to create virtual environments with accurate real-world properties such as: gravity, friction and drag. To aid in generating such an environment, Gazebo provides a GUI alongside hundreds of object models (trees, rocks, buildings, etc) for creating complex indoor/outdoor terrain on which virtual robots can be placed. Once a robot model is placed and integrated with ROS, the robot can move around the virtual terrain and receive sensor data.

NS-3. Using the discrete event network simulator NS-3, it is possible to create an accurate prediction of how a network will perform given a specific network topology. The simulator describes each of the devices in the simulated network as nodes which each have their own network properties. It can then analyzing the distance, interference and individual properties between each of the nodes to predict network performance using metrics such as: packet loss probability and packet delay.

Simulation environment design. We generated simulated scenarios for both LOS and NLOS channels to demonstrate that our technique is capable of executing its intended function. The environments are rendered on Gazebo (*Baylands* environment), which in our case, hosts the physics simulator. The designed environmental setting has dimensions of one hundred by one hundred meters, as seen in the figure 4. And each grid symbolizes a square with a side length of 20 meters. The environment was designed to accommodate both LOS and NLOS scenarios, and the majority of the environment is populated with trees, house model, wooden case, and elevated land to support the NLOS abstraction. A range of robotic agents, including UGVs and UAVs, have been chosen for use in this task. The UAV was chosen to be an Iris drone, while the UGV was chosen to be two Husky robots. We have used the MAVROS package for the connection between the Iris drone and the ROS environment. Also, the data

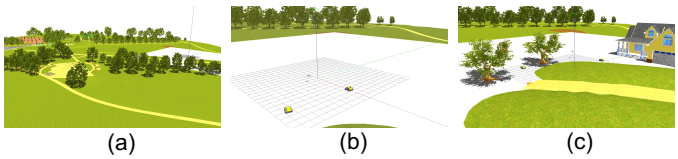

Figure 4. The Gazebo simulation interface of our system. We have used (a) the Baylands terrain as our base environment. The next two images are (b) LOS, (c) NLOS communication scenario respectively. We have introduced trees, dense forest, wooden box, and house model for NLOS abstraction.

packets captured from the drone are published as MAVLINK messages. Considering the masterless scenario, each robotic agent is equipped with its own network. The IEEE 802.11 (Wi-Fi) interface has been used as our wireless communication medium. All the deployed agents are aware of the IPs of all the agents present in the simulation, and the agents themselves are configured to communicate data points across TCP/IP lines. We have captured ROSbag containing the accelerometric data of the robotic agents for a fixed 10 seconds every time we create a running event. In order to analyze the performance of the communication, we have opted to utilize one of the most advanced network simulators available, NS-3, which is also compatible with the wireless stack that we have chosen. In order to integrate Gazebo with NS-3, we deployed our technique as a synchronizing middleware. When employing our technique, the velocity difference between the agents were calculated from the data of the $/gazebo\_states/twist$ topic to adjust the window size. The velocity of the agents is modified in line with the circumstances in order to carry out this scenario. The algorithm changes the initial window size, which is set to 1 millisecond (mS).

The agents in the line of sight scenario are put in an environment devoid of any barriers, as shown in the figure 4. The ability to test communication performance while maintaining LOS abstraction is enabled by simulating the UGVs and the UAV traveling at different speeds. The performance

matrices, according to the information in this section, include the average delay and the proportion of missed packets. The results are shown together with the resulting shift in the distance between the agents. For NLOS implementation, we have introduced dense forest and also elevated surfaces between agents when transmitting data.

C. Simulation Results

We investigate two unique communication scenarios: communication between UGVs and communication between UAVs and UGVs. The agents in the case of NLOS abstraction are programmed to operate in an object-populated environment. The signal's strength is likely to be reduced under these conditions. In this scenario, the reporting paradigm and communication conditions are identical to those used in the preceding case. When communicating between two UGVs, it has been shown that when the difference in relative velocity between the two vehicles is lower, the agents retain the data quantity during transmission for around 40 meters, as reported at table II. In this case, the average delay (the sum of all delays experienced during the transmission of 10 seconds of accelerometric data) is also less than 0.1 second. Up to this point, the proposed alternative, which uses an adjustable window under masterless environment, perform well for line-of-sight communication. The master-based technique without window adjustment V, on the other hand, begins to lose essential information significantly after this point, resulting in a delay. When the agents in this simulation are the furthest apart from one another, a relatively small adjustment to the sliding window value (0.1 milliseconds in this case) is observed to contribute to a nearly 20% improvement in LOS retrieval and at least a 10% improvement in NLOS retrieval (100m).

Table II
SIMULATION RESULTS FOR OUR MASTERLESS SYNCHRONIZING MIDDLEWARE ON BOTH LOS, AND NLOS COMMUNICATION SCHEME USING AVERAGE DELAY (PD_a) (S) AND PACKET LOSS PROBABILITY (L_p) (%) MATRICES CONSIDERING A UGV TO UGV COMMUNICATION SCENARIO.

Distance (m)	**LOS**		**NLOS**	
	PD_a (s)	L_p (%)	PD_a (s)	L_p (%)
20	0.01	0	0.2	0
40	0.1	0	0.5	10
60	0.4	0	0.8	10
80	0.4	10	1	20
100	0.6	10	1	30

Non-line-of-sight (NLOS) communication performance was significantly reduced during testing with UGVs to UAVs, as seen in the table III. To be more specific, when employing the masterless strategy with a distance of 100 meters or more separates each pair of agents, nearly all of the data is lost. This circumstance has an effect on the proposed adjustable window technique, which results in a 0.3 millisecond increase and leaves space for future advancement. Aside from this one issue, the redesigned window improves both line-of-sight (LOS) and non-line-of-sight (NLOS) communication significantly. In summary, we achieved an improvement of 40% and 20% in terms of average delay and packet loss probability respectively for NLOS scenario in comparison with master-based approach. For LOS, the margins are 20% and 10% respectively.

Table III
SIMULATION RESULTS FOR OUR MASTERLESS SYNCHRONIZING MIDDLEWARE ON BOTH LOS, AND NLOS COMMUNICATION SCHEME USING AVERAGE DELAY (PD_a) (S) AND PACKET LOSS PROBABILITY (L_p) (%) MATRICES CONSIDERING A UGV TO UAV COMMUNICATION SCENARIO.

Distance (m)	**LOS**		**NLOS**	
	PD_a (s)	L_p (%)	PD_a (s)	L_p (%)
20	0.1	0	0.5	0
40	0.2	10	0.7	20
60	0.2	10	0.8	20
80	0.5	20	1.3	60
100	0.6	30	1.6	90

D. Baseline Comparison Results for the Proposed QoS Policy

When developing real-time robotic systems, Quality of Service (QoS) policies play a crucial role in determining how data is transmitted and processed. The Robot Operating System 2 (ROS2) provides a set of QoS policies that can be used to customize the behavior of data transmission and processing. In this work, we will compare three of the most commonly used ROS2 QoS policies varying their 'Reliability' parameter: 'Best_effort', 'Reliable' and our proposed one.

The 'Best_effort' policy is the default policy used by ROS2 when creating a publisher or subscriber. This policy provides best-effort reliability, meaning that there is no guarantee that all messages will be received by the subscriber. This policy also sets the deadline period to 100ms, which means that if a subscriber does not receive a message within 100ms of the publisher sending it, the message will be considered lost. On the other hand, the 'Reliable' policy provides a higher level of reliability. This means that all messages sent by the publisher will be received by the subscriber, provided that they are subscribed at the time the message is sent. This policy also sets the deadline period to infinite, meaning that there is no time limit for a message to be received by the subscriber.

Our proposed QoS policy is suitable for applications where message loss is undesirable and higher reliability is required at the cost of a considerable trade-off with latency. Our proposed QoS policy is suitable for applications where message loss is unacceptable and higher reliability is required at the cost of a considerable trade-off with latency. Analyzing the detailed results in IV, we can observe that even keeping the agents 100 meters apart, we can preserve 90% of the data. There is a $0.3s$ of increase in the latency compared to the 'Reliable' policy due to the iteration needed in our case to select the proper window size. However, we record an improvement of 10% and 30% respectively, compared to the default 'Best_effort' and 'Reliable' baselines. If we vary the 'Depth' parameter, we can see an increase in both PD_a and L_p due to the higher volume of data being transferred.

Table IV
COMPARISON OF OUR PROPOSED QOS POLICY WITH TWO OF THE BASELINE ROS2 QOS POLICIES IN TERMS OF 'RELIABILITY' PARAMETER

Depth	Distance (m)	Best_Effort		Reliable		Proposed	
		PDa (s)	Lp (%)	PDa (s)	Lp (%)	PDa (s)	Lp (%)
5	20	0.01	0	0.01	0	**0.03**	**0**
	40	0.08	0	0.1	0	**0.24**	**0**
	60	0.42	10	0.48	10	**0.82**	**0**
	80	0.46	30	0.6	10	**1.08**	**10**
	100	0.5	40	0.8	20	**1.27**	**10**
10	20	0.18	10	0.2	10	**0.32**	**0**
	40	0.4	30	0.5	10	**0.56**	**10**
	60	0.54	50	0.8	30	**1.1**	**20**
	80	0.96	60	1	40	**1.6**	**20**
	100	1	60	1.4	50	**2.1**	**30**

VI. SYSTEM IMPLEMENTATION

To compare the results from the simulated environment, we conducted experiments to emulate master (ROS1) and master-less (ROS2) architectures in the real world. However, the robotic agents are controlled manually and distances are far closer to avoid damaging equipment.

A. Hardware

The hardware used for this experiment are: two Turtlebot3 Burgers and a Duckiedrone.

Duckiedrone. This UAV is developed by Duckietown, a company known for its open source and differential robots. The drone is equipped with a Raspberry Pi 3b that we will use to produce the wifi signal, which can reach a distance of about 30 meters at 59/97.67 Mbps. ROS Kinetic and several other packages are installed on the drone to enable the use of positional control. This is where the drone automatically shifts its position to maintain a static image from the *Arducam*, which is faced directly toward the ground.

Turtlebot3 Burger. This UGV is manufactured by Robotis, which focuses on its compatibility with ROS and its available versions. This is exemplified by many ROS developers (such as the developers of [8]) using the robot to demonstrate newly released packages. The robot is equipped with a LIDAR for object detection and two wheels controlled by an OpenCR board for movement.

B. Experimentation Flow

The experimentation is categorized into two main sections: master-based and master-less approaches. Each approach is then tested using TCP and UDP in line-of-sight (LOS) and non-line-of-sight (NLOS) situations. The overall process can be summarized as follows:

1) On the sender (Turtlebot) generate a ROSbag.
2) Open filters to detect the amount of packets being sent and received. Then send the ROSbag.
3) Use ping to determine the average delay.
4) Repeat for an increased distance.

This experimentation is repeated 20 times to generate an average packet loss and delay.

To Acheive LOS, Turtlebot(s) are placed in direct line-of-sight of the Duckiedrone. During NLOS we placed several chairs and a foam box obstructing line-of-sight. A sample

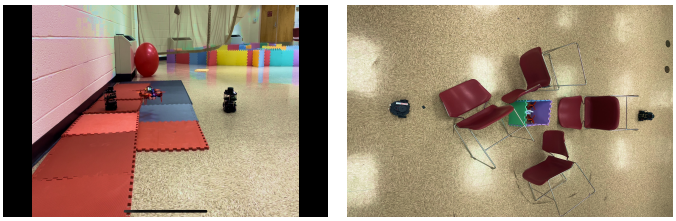

Figure 5. Sample implementation of LOS (image on the left), and NLOS (image on the right) communication scenario with a Duckiedrone and two turtlebots.

illustration of the implementation scenario is provided in figure 5. However, this scenario did not create any visible changes in the results. Therefore NLOS results are generated with the Turtlebot sender placed on the other side of a wall.

ROSbag Formation The file to be sent across agents was generated by using ROSbag through subscribing to the $/odom$ topic of a Turtlebot for 15 seconds. In this experiment, the generated file reached a size of 293797 bytes.

Communication Scheme For evaluating network performance, we used *Netcat* traditional as it allows to send files using both UDP and TCP protocols and *TCPdump* to evaluate the performance metrics. Similar to the receiver, a filter is set on the sender for *TCPdump* to capture any packets coming from the $sender\_ip$. *Netcat* then sends file to the receiver. While conducting the experiment using the rosbag file. *TCPdump* detected an average of $207 - 502$ byte packets sent using TCP. While for UDP an average of $36 - 8192$ byte packets were detected. The window size of each protocol will be used with ping, to approximate delay and time to send the file.

C. System Results

In a Master-less system all robotic agents are connected to each other and communication doesn't flow through a single point. This testing represents a single branch in a master-less system using two robots as a full system. To emulate that branch, a Turtlebot connects to the Duckiedrone's network, and the procedure begins between the Turtlebot and Duckiedrone. UDP shows an initial packet loss of 35%, which

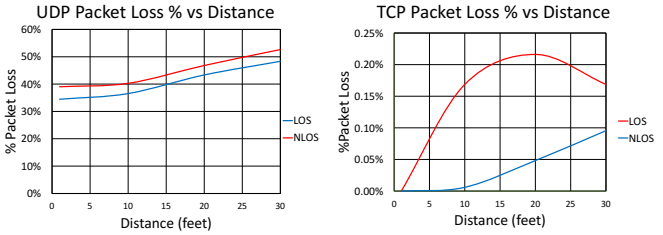

Figure 6. A comparison of TCP and UDP is based on packet loss probability trends that vary with agent distance in a multi-agent environment. This system uses a master-based (ROS1) architecture.

steadily increases as the robots are moved further away. When the sending Turtlebot is placed behind a wall, this shifts the packet loss by 5%. At first glance of the TCP plot, it seems that NLOS is more efficient than LOS. However, the scale

of the plots shows overall packet loss is less than 0.25% meaning it is almost negligible. The detailed results can be found in figure 6. In terms of latency, when completing the experiment up to a maximum distance of 55 feet between the Turtlebot and Duckiedrone, the delay of UDP averaged at about 13ms (file transferred in 468ms) and the delay of TCP averaged at 4.5ms (file transferred in 931.5ms). Notably, for master-based (ROS1) architecture, the delay would reach approximately 16ms for UDP (file transferred in 576ms) and 5ms for TCP (file transferred in 1035ms).

VII. Ablation Study

In this section we present some baseline studies with ROS1 implementation of our proposed synchronizing middleware and experimentation using both TCP and UDP transmission protocols. Also, we present the analysis on real-time network simulator, EMANE along with some possible future research directions.

A. Varying Transmission Protocol within a Master-based System

We have implemented our middleware on a master-based environment for both TCP and UDP to present a comparative analysis of our proposed masterless system with the ROS1 architecture. For both cases, the same Gazebo environment was chosen. The steps required for to transform our proposed algorithm into UDP are described in 2. As soon we implement those steps, we can run the example simulation and communicate between agents in UDP mode. After running

Algorithm 2 Conversion of TCP to UDP pseudocode

Require: $socketType = DGRAM$
Ensure: $socketDomain = INET$
  remove listen and accept
  if Send **then**
    $sendAll \leftarrow$ size of message
    $sendAll \leftarrow$ message
  else if Recv **then**
    $recvAll \leftarrow$ size of message
    $recvAll \leftarrow$ message
    $connect \leftarrow address$ ▷ address returned from recvall
  end if

the simulation, the results are shown in table V. From LOS situations, we can observe that UDP communication has an overall higher packet loss probability but lower packet delay. However, at low distances, the performance of UDP is competitive to our proposed TCP-based architecture. On the other hand, in NLOS situations, the results were somewhat self-explanatory. Similarly to LOS situations, there was little difference in packet delay, but, for the most part, delay was less in comparison to TCP. Packet loss probability, however, showed a more accurate comparison, showing that the packet loss probability for UDP was much higher than TCP. Another simulation to consider is testing a more congested NLOS environment, introducing more obstacles and observing their change to the results. Introducing obstacles between the agents would cause the aspects of UDP to increase. As TCP delay increases, UDP delay increases as well, albeit by a smaller amount. On the other hand, as packet loss in TCP increases, UDP packet loss also increases greatly. This means that when an NLOS environment is simulated for UDP, both the benefits of lower latency and the drawback of less reliability are made worse. To state the comparative margins between our technique and master-based: we improved average delay and packet loss probability by 40% and 20% for NLOS against the master-based technique. LOS margins are 20% and 10%, respectively.

Table V
Result comparison between TCP and UDP transmission protocol for master-based (ROS1) system considering both LOS and NLOS under a UGV to UGV communication scenario.

Distance (m)	LOS				NLOS			
	Average Delay (s)		Packet Loss Probability (%)		Average Delay (s)		Packet Loss Probability (%)	
	UDP	TCP	UDP	TCP	UDP	TCP	UDP	TCP
20	0.01	0.01	0	0	0.18	0.2	10	10
40	0.08	0.1	0	0	0.4	0.5	30	10
60	0.42	0.48	10	10	0.54	0.8	50	30
80	0.46	0.6	30	10	0.96	1	60	40
100	0.5	0.8	40	20	1	1.4	60	50

B. Experimentation with a Real-time Network Simulator

We have chosen the EMANE emulator to focus on real-time modeling of network configurations. This experimentation is conducted to counter the need for synchronizing middleware and check whether a real-time network emulator is sufficient to replace our proposed middleware along with the advantages we are offering. The integration of EMANE with ROS is still not officially included. As a result, it is not possible to replicate the same experimental setting as we did for our middleware. Instead, we have designed a similar data transmission environment inside of EMANE, utilizing its emulator mode to measure the same performance metrics we have used for our synchronizing middleware.

We have designed a 10 node mesh network working on the TCP/IP transfer protocol. The default conditions create a 0% packet loss with an average delay of 5.377ms using 100 pings of 64 bytes. With the increase in the longitude, latitude, and altitude of node 10 by 100, the packet delay increases to 54.309ms. This is with pinging from node 1 to node 10 without any hops. The takeaway from this experiment is that although EMANE ensures the receipt of all the packets between nodes (TCP/IP mode), the packet delay is almost 10 times greater than our method. Moreover, the scenario was relatively less congested and NLOS was not present in EMANE. Creating the right ROS packages for integrating EMANE with a physics simulator could be an interesting area of research for analyzing masterless systems in depth.

VIII. Conclusion

In this work, we have questioned the communication overhead issue of the master-based ROS1 system while developing a synchronizing middleware for co-simulation of physics and network simulator. Our middleware uses a customized QoS policy which works on a masterless (ROS2) system and is capable of handling the heterogeneity in a multi-robot environment. Additionally, to increase reliability in terms of minimizing packet loss probability, the QoS policy is integrated with a dynamic adjustment strategy within the TCP/IP sliding window. This algorithm can change the size of the window based on how fast the agents are moving so that they can work better together and communicate. We have designed extensive empirical analysis for both simulation and real robot deployment to showcase the efficacy of our proposed system. Even in difficult non-line-of-sight (NLOS) environments with heterogeneous agents, experimental data shows that our solution outperforms the standard fixed window-based strategy in terms of ensuring fewer packet losses (on average 10% improvement) and faster transmission (about 12% improvement). Experimental results also show that our synchronizing middleware can outperform the real-time network simulator, i.e., EMANE, in terms of efficient communication, which validates the necessity of such middleware.

Acknowledgements

This work has been supported by U.S. Army Grant #W911NF2120076.

References

[1] C. Gomes, C. Thule, D. Broman, P. G. Larsen, and H. Vangheluwe, "Co-simulation: a survey," *ACM Computing Surveys (CSUR)*, vol. 51, no. 3, pp. 1–33, 2018.

[2] T. Ahn, J. Seok, I. Lee, and J. Han, "Reliable flying iot networks for uav disaster rescue operations," *Mobile Information Systems*, vol. 2018, 2018.

[3] M. Kudelski, L. M. Gambardella, and G. A. Di Caro, "Robonetsim: An integrated framework for multi-robot and network simulation," *Robotics and Autonomous Systems*, vol. 61, no. 5, pp. 483–496, 2013.

[4] D. Brunori, S. Colonnese, F. Cuomo, and L. Iocchi, "A reinforcement learning environment for multi-service uav-enabled wireless systems," in *2021 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, pp. 251–256, IEEE, 2021.

[5] P. J. Durst, C. Goodin, C. Cummins, B. Gates, B. Mckinley, T. George, M. M. Rohde, M. A. Toschlog, and J. Crawford, "A real-time, interactive simulation environment for unmanned ground vehicles: The autonomous navigation virtual environment laboratory (anvel)," *2012 Fifth international conference on information and computing science*, pp. 7–10, 2012.

[6] W. K. Seah, Z. A. Eu, and H.-P. Tan, "Wireless sensor networks powered by ambient energy harvesting (wsn-heap)-survey and challenges," in *2009 1st International Conference on Wireless Communication, Vehicular Technology, Information Theory and Aerospace & Electronic Systems Technology*, pp. 1–5, Ieee, 2009.

[7] M. Chen, W. Liang, and S. K. Das, "Data collection utility maximization in wireless sensor networks via efficient determination of uav hovering locations," in *2021 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pp. 1–10, IEEE, 2021.

[8] M. Calvo-Fullana, D. Mox, A. Pyattaev, J. Fink, V. Kumar, and A. Ribeiro, "Ros-netsim: A framework for the integration of robotic and network simulators," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1120–1127, 2021.

[9] J. M. Schlesselman, G. Pardo-Castellote, and B. Farabaugh, "Omg data-distribution service (dds): architectural update," *IEEE MILCOM 2004. Military Communications Conference, 2004.*, vol. 2, pp. 961–967 Vol. 2, 2004.

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

[11] M. Xiong, J. Parsons, J. R. Edmondson, H. Nguyen, and D. C. Schmidt, "Evaluating the performance of publish / subscribe platforms for information management in distributed real-time and embedded systems," 2006.

[12] S. Sierla, J. Peltola, and K. Koskinen, "Evaluation of a real-time distribution service," *Kari.O.Koskinen@hut.fi*, 01 2003.

[13] Y. Maruyama, S. Kato, and T. Azumi, "Exploring the performance of ros2," in *Proceedings of the 13th International Conference on Embedded Software*, pp. 1–10, 2016.

[14] S. Moon, J. J. Bird, S. Borenstein, and E. W. Frew, "A gazebo/ros-based communication-realistic simulator for networked suas," in *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 1819–1827, IEEE, 2020.

[15] M. Fields, R. Brewer, H. L. Edge, J. L. Pusey, E. Weller, D. G. Patel, and C. A. DiBerardino, "Simulation tools for robotics research and assessment," in *Unmanned Systems Technology XVIII*, vol. 9837, p. 98370J, International Society for Optics and Photonics, 2016.

[16] E. Dey, J. Hossain, N. Roy, and C. Busart, "Synchrosim: An integrated co-simulation middleware for heterogeneous multi-robot system," in *2022 18th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pp. 334–341, IEEE, 2022.

[17] S. Baidya, Z. Shaikh, and M. Levorato, "Flynetsim: An open source synchronized uav network simulator based on ns-3 and ardupilot," in *Proceedings of the 21st ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pp. 37–45, 2018.

[18] M. Calvo-Fullana, D. Mox, A. Pyattaev, J. Fink, V. Kumar, and A. Ribeiro, "Ros-netsim: A framework for the integration of robotic and network simulators," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1120–1127, 2021.

[19] S. Acharya, A. Bharadwaj, Y. Simmhan, A. Gopalan, P. Parag, and H. Tyagi, "Cornet: A co-simulation middleware for robot networks," in *2020 International Conference on COMmunication Systems & NETworkS (COMSNETS)*, pp. 245–251, IEEE, 2020.

[20] A. Suzuki, K. Masutomi, I. Ono, H. Ishii, and T. Onoda, "Cps-sim: Co-simulation for cyber-physical systems with accurate time synchronization," *IFAC-PapersOnLine*, vol. 51, no. 23, pp. 70–75, 2018.

[21] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and service robotics*, pp. 621–635, Springer, 2018.

[22] C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. D. Caro, and F. Ducatelle, "Argos: a modular, parallel, multi-engine simulator for multi-robot systems," *Swarm intelligence*, vol. 6, no. 4, pp. 271–295, 2012.

[23] "Gazebo." https://gazebosim.org/. Accessed: 2022-08-06.

[24] A. Varga, "Omnet++," in *Modeling and tools for network simulation*, pp. 35–59, Springer, 2010.

[25] "Ns-2." https://www.isi.edu/nsnam/ns/. Accessed: 2021-12-17.

[26] G. F. Riley and T. R. Henderson, "The ns-3 network simulator," *Modeling and tools for network simulation*, pp. 15–34, 2010.

[27] J. Ahrenholz, T. Goff, and B. Adamson, "Integration of the core and emane network emulators," in *2011-MILCOM 2011 Military Communications Conference*, pp. 1870–1875, IEEE, 2011.

[28] P. J. Durst, C. Goodin, C. Cummins, B. Gates, B. Mckinley, T. George, M. M. Rohde, M. A. Toschlog, and J. Crawford, "A real-time, interactive simulation environment for unmanned ground vehicles: The autonomous navigation virtual environment laboratory (anvel)," in *2012 Fifth international conference on information and computing science*, pp. 7–10, IEEE, 2012.

[29] "Unity simulator." https://unity.com/products/unity-simulation-pro. Accessed: 2021-12-17.

[30] Y. Chen, S. HU, X. LI, C. Wang, and Z. Chen, "An improved rudp for data transmission in embedded real-time system," in *2019 IEEE International Conference on Signal, Information and Data Processing (ICSIDP)*, pp. 1–5, 2019.

[31] M. Björkbom, S. Nethi, L. Eriksson, and R. Jantti, "Wireless control system design and co-simulation," *Control Engineering Practice*, vol. 19, pp. 1075–1086, 09 2011.

[32] C. Lu, J. A. Stankovic, S. H. Son, and G. Tao, "Feedback control real-time scheduling: Framework, modeling, and algorithms," *Real-Time Systems*, vol. 23, pp. 85–126, 2002.