Revue Ouverte d'Intelligence Artificielle
Brouillon d'article, version du 10 avril 2024

Etat de l'art sur la co-simulation robotique et réseau des systèmes multi-robots

Théotime Balaguer [a, b, c], Olivier Simonin [a], Isabelle Guerin-Lassous [b], Isabelle Fantoni [c]

[a] INSA Lyon, Inria, CITI Lab., France
E-mail : theotime.balaguer@insa-lyon.fr, olivier.simonin@insa-lyon.fr

[b] Université Lyon 1, ENS Lyon, CNRS, LIP, UMR 5668, France
E-mail : isabelle.guerin-lassous@ens-lyon.fr

[c] Nantes Université, École Centrale Nantes, CNRS, LS2N, UMR 6004, France
E-mail : isabelle.fantoni@ls2n.fr.

RÉSUMÉ. --- La simulation de systèmes multi-robots nécessite l'intégration des composantes robotique et réseau au sein d'un même simulateur. Pour accélérer le développement d'un tel logiciel, il semble efficace de réutiliser les outils existants dans les communautés robotique et réseau, mais la fusion de deux simulateurs présente des défis structurels qu'il faut surmonter afin d'obtenir une simulation "réaliste" d'un système multi-robots. Dans cet état de l'art, nous étudions les "co-simulateurs" qui abordent cette problématique et détaillons les défis auxquels il faut répondre pour créer un co-simulateur utile et performant. Nous présentons aussi nos travaux en cours pour la création d'un co-simulateur dédié aux systèmes multi-robots aériens.

MOTS-CLÉS. --- Systèmes multi-robots, Co-simulation, Réseaux de communication.

1. INTRODUCTION

Ces dernières décennies ont montré que les systèmes multi-robots peuvent surpasser un robot seul pour certaines missions, voir répondre à des problématiques inenvisageables pour un seul robot [5]. Grâce aux coûts décroissants des composants électroniques, il est de plus en plus facile de déployer une flotte de robots collaboratifs capable d'achever des missions plus vite, à plus grande échelle et avec plus de robustesse qu'un robot solitaire. Les développements de l'intelligence artificielle et de l'algorithmique distribuée participent par ailleurs grandement à la démocratisation de ces systèmes multi-robots.

Cependant, le déploiement de systèmes de robots connectés cristallisent des problématiques à la frontière entre le domaine de la robotique et celui des réseaux [19]. Pour les étudier, il est donc essentiel de disposer d'outils de simulation capables de modéliser ces deux aspects de front, pour pouvoir tester beaucoup de configurations différentes, passer à l'échelle en terme de nombre de drones sans investissement important,

permettre l'apprentissage par renforcement, éviter la casse de matériel et les dangers pour les expérimentateurs ou encore offrir des possibilité de reproduction des simulations aux membres de la communauté. Les flottes de drones collaboratifs reflètent parfaitement ce besoin : d'un côté, les drones ont des contraintes physiques qu'il faut respecter (rayon de braquage, accélération, etc.) et de l'autre, les échanges d'information entre les drones sont potentiellement vitaux pour la flotte elle-même (évitement de collisions, vol en formation, etc.) ou pour la mission (partage d'informations, relais réseau, etc.).

Les communautés de chacun de ces deux domaines – robotique et réseau – disposent déjà d'outils stables et performants [9][21], mais ces logiciels ne permettent pas de simuler conjointement les deux aspects.
Ce constat fait, la *co-simulation*, c'est-à-dire l'exécution simultanée de deux simulateurs, semble être la solution idéale : elle permet de gagner du temps en bénéficiant du travail déjà effectué dans chaque domaine. Il nous semble aussi important d'introduire ROS (*Robot Operating System*)[(1)], une suite de logiciels facilitant le développement et l'intégration des systèmes robotiques et qui s'est imposée comme un outil incontournable pour la conception et l'opération de robots. Prendre en compte l'intégration de ROS dans le développement d'un co-simulateur permet une prise en main plus directe par les roboticiens.

Dans cet article, nous présentons et comparons les co-simulateurs existants et discutons des principales problématiques de co-simulation. A notre connaissance, ce travail de synthèse n'a jamais été mené pour la co-simulation robotique et réseau. Cet état de l'art peut servir de guide pour le choix d'un co-simulateur adapté à une problématique, ou fournir des pistes pour le développement d'un nouvel outil de co-simulation.

ORGANISATION DE L'ARTICLE. --- En Section 2, nous présentons les principaux simulateurs multi-robots et les simulateurs de réseau les plus connus. La Section 3 expose les co-simulateurs robotique et réseau existants. En Section 4, nous discutons l'architecture type d'un co-simulateur ainsi que les fonctionnalités recherchées. Les Sections 5 et 6 s'intéressent à deux problématiques récurrentes de la co-simulation, à savoir la synchronisation et l'échange d'information entre les simulateurs. La Section 7 présente nos travaux en cours sur la création d'un co-simulateur réaliste pour les flottes de robots aériens. Enfin, nous concluons en Section 8.

2. SIMULATEURS SPÉCIFIQUES

Si l'évaluation rigoureuse et exhaustive de simulateurs de robotique et de réseau dépasse largement les limites de cet article, il est nécessaire pour la compréhension de notre discours de présenter ici les simulateurs les plus notables pour chacun de ces domaines. Pour plus de détails, le lecteur peut se référer à [9] pour les simulateurs de robotique (aussi appelés simulateurs de physique) et [21] pour les simulateurs de réseau.

[(1)] ROS : https://www.ros.org/

2.1. Simulateurs multi-robots

Le rôle du simulateur de robotique est de simuler les interactions du robot avec son environnement. Dans [9], les auteurs présentent un état de l'art des simulateurs de robotique complet mais sans se focaliser sur les systèmes multi-robots. Selon les auteurs, un simulateur de robotique doit contenir *a minima* :

- Un moteur simulant les phénomènes physiques ;
- Des modèles de friction et de collision ;
- Une interface graphique ;
- Une fonctionnalité d'import de scènes et de *meshes* ;
- Une interface de programmation applicative (API) permettant l'accès à ses propriétés ;
- Une bibliothèque d'effecteurs et de capteurs prêts à l'emploi.

De nombreux simulateurs de physique ont été développés ces dernières années. Certains sont spécialisés dans un sous-domaine de la robotique (robotique aérienne [33], sous-marine [25] [8], corps mous [13], robotique médicale [16], etc.) alors que d'autres permettent de simuler une grande variété de robots [9].
Le système multi-robots est un cas particulier au sein de la robotique, pour lequel la flexibilité et l'efficacité du simulateur sont particulièrement importants. La flexibilité d'un simulateur est son aptitude à s'adapter facilement à de nouvelles situations (différents modèles de robots ou moteurs de physique, par exemple). L'efficacité d'un simulateur est sa capacité à utiliser l'ensemble des capacités de calcul mises à sa disposition.

Devant cette grande diversité de simulateurs, nous nous restreignons ici aux simulateurs multi-robots mobiles, actuellement maintenus et qui sont les plus utilisés par la recherche. Par exemple, Airsim [33] est un simulateur réaliste très utilisé mais ne supportant pas les systèmes multi-robots.

2.1.1. *ARGoS*

ArGoS [30] est un simulateur spécifiquement créé pour la simulation multi-robots. Ce simulateur est basé sur une architecture modulaire offrant à l'utilisateur une grande flexibilité et une bonne efficacité. En effet, différents modules caractérisés par leur précision et leur coût en calcul peuvent être choisis, pour allouer la puissance de calcul là où l'utilisateur considère que c'est nécessaire. ARGoS est capable de simuler de manière efficace des essaims de plusieurs milliers de robots grâce à une caractéristique le démarquant des autres simulateurs : le monde 3D simulé peut être découpé en régions. L'utilisateur peut assigner à chaque région un moteur de physique différent (plus ou moins précis, par exemple), et les calculs associés à chaque région peuvent être parallélisés.

2.1.2. *Webots*

Le développement de Webots a démarré en 1996, faisant de ce simulateur l'un des plus anciens simulateurs multi-robots. Longtemps distribué sous licence propriétaire

par Cyberbotics Ltd., Webots est distribué gratuitement depuis 2018, ce qui a accéléré son adoption par la communauté scientifique. En plus de fonctionnalités avancées et d'un haut niveau de réalisme, Webots propose des services annexes tels que l'accès à une version du simulateur sur le web[(2)] ou un support utilisateur. Concernant notre problématique robotique/réseau, il est intéressant de remarquer que Webot procure des nœuds *émetteur* et *récepteur* pouvant être utilisés pour modéliser des liaisons radio simples, en paramétrant une portée maximale de communication et un angle d'ouverture du cône d'émission pour les émetteurs infrarouges. Cette fonctionnalité reste cependant extrêmement limitée car elle ne permet pas de préciser les technologies radio ou les protocoles.

2.1.3. *CoppeliaSim*

CoppeliaSim [32], anciennement appelé V-REP, est un simulateur de robotique dont l'architecture permet une gestion efficace des systèmes multi-robots. Une étude récente comparant les performances et précision de quatre simulateurs populaires dans la communauté de la robotique [15] place CoppeliaSim parmi les meilleurs simulateurs multi-robots (Gazebo, Webots, CoppeliaSim et MORSE sont étudiés). Bien que gratuit, CoppeliaSim n'est pas open source, un facteur qui peut être limitant pour l'interfaçage avec d'autres simulateurs.

2.1.4. *Gazebo*

Gazebo (illustré figure 2.1) est le simulateur de robotique le plus utilisé dans la recherche [9], et ses fonctionnalités le rendent parfaitement adapté à la simulation multi-robots.
Gazebo s'est construit autour du projet Player/Stage [18], puis son développement a été confié à l'*"Open Source Robotics Foundation"* (OSRF) en 2012. Pour améliorer sa flexibilité, le développement de Gazebo a pris un tournant majeur en 2019, avec la publication d'une version entièrement réécrite du simulateur. L'architecture monolithique historique ayant été remplacée par une architecture modulaire, Gazebo est capable de simuler tous types de robots, dans n'importe quel environnement, et en acceptant des possibilités d'extension infinies. La plus grande force de Gazebo réside cependant dans sa communauté nombreuse et active, qui partage de nouveaux modules, corrige des bugs et procure de l'aide en ligne. Le support des industriels à la fondation *"Open Source Robotics"* [(3)] fournit un avantage non négligeable pour le maintien et le développement de ce simulateur. L'OSRF étant aussi responsable du développement de ROS, on peut facilement affirmer que ROS et Gazebo sont et resteront entièrement compatibles.

[(2)] RobotBenchmark : https://robotbenchmark.net/
[(3)] Intrinsic Acquires OSRC and OSRC-SG : https://tinyurl.com/5eb7bxfb

FIGURE 2.1. Visualisation de six robots quadricoptères dans Gazebo

2.2. SIMULATEURS RÉSEAUX

Un simulateur de réseau crée un modèle virtuel d'un réseau d'ordinateurs. Il permet d'expérimenter rapidement des algorithmes, d'évaluer et de comparer les performances d'architectures et de technologies de communication diverses. L'unité principale de modélisation est souvent le "paquet", ce qui rend les simulateurs de réseau très réalistes pour simuler les protocoles mais moins pour les couches physiques (propagation notamment) et applicatives, qui sont représentées par des modèles plus abstraits. Dans l'écrasante majorité des cas, les réseaux de robots sont des réseaux sans fil, nous nous intéresserons donc spécifiquement aux simulateurs performants dans la simulation des réseaux sans fil. Nous restreindrons de plus notre étude aux simulateurs open-source, actuellement maintenus et qui sont les plus utilisés pour la recherche. Certains simulateurs ont dû être écartés, comme Netsim [(4)] qui est une alternative payante et OPNET [(5)] qui est de moins en moins utilisé.

2.2.1. *OMNeT++*

OMNeT++ [35] est un simulateur à événements discrets modulaire dont la première publication date de 1997. OMNeT++ a la particularité de se présenter comme une *plateforme de simulation*, sur laquelle les équipes de recherche et les industriels peuvent construire leur propre *simulateur de réseau*. OMNeT++ fournit une interface graphique permettant de prototyper plus rapidement des réseaux grâce aux outils de visualisation.

[(4)] Netsim™ : https://netsim.boson.com/

[(5)] OPNET : https://opnetprojects.com/opnet-network-simulator/

2.2.2. *Mininet*

Mininet [(6)] est un émulateur de réseau léger et facile d'utilisation créé pour la simulation des approches dites *Software Defined Network* (SDN). Son extension Mininet-WiFi [17] permet d'émuler des réseaux sans fil SDN. Mininet-WiFi permet des expérimentations riches et facilement déployables grâce à une distribution par conteneurs Docker. Il se révèle très utile pour l'étude des SDNs et de leur protocole de routage phare, OpenFlow [28].

2.2.3. *ns-3*

ns-3 [31] remplace son prédécesseur NS-2, le simulateur de réseau qui était déjà le plus utilisé dans les années 2010. Bâti sur l'expérience de ce simulateur largement adopté par la communauté, ns-3 capitalise sur des années de contributions d'une communauté nombreuse et active et reste aujourd'hui le simulateur de réseau le plus utilisé dans la recherche. La grande force de ns-3 réside dans le nombre et la diversité de modules disponibles, qui permettent de simuler de nombreuses technologies sous des topologies variées sans avoir à produire beaucoup de code. Notons cependant que ns-3 manque d'une interface graphique intuitive et permettant de simuler rapidement des cas simples.
Enfin, les standards de développement sévères imposés par ns-3 assurent que la documentation est à jour et que chaque module est scrupuleusement testé avant son intégration dans le simulateur. Ces règles permettent aux chercheurs d'accorder une grande confiance aux résultats des études menées dans ns-3.

3. Architecture et propriétés des co-simulateurs

Un co-simulateur assure le fonctionnement de deux (ou plus) simulateurs spécifiques en simultané, mais il permet aussi d'offrir à l'utilisateur des fonctionnalités avancées facilitant la simulation de systèmes multi-robots. Dans cette section, nous présentons l'architecture type d'un co-simulateur robotique et réseau, et listons les fonctionnalités qui nous semblent les plus importantes pour une utilisation fluide du co-simulateur.

3.1. Architecture

Les co-simulateurs présentés ci-après ont tous une architecture différente, mais ils partagent une philosophie de co-simulation qui peut être résumée ainsi : un programme sur-mesure initialise les simulateurs robotique et réseau puis met en place des canaux de communication entre ceux-ci. Si le co-simulateur intègre la gestion du temps, un programme s'assure que les simulateurs fonctionnent de manière synchronisée tout au long de la simulation. Nous proposons une représentation générique de cette architecture-type dans la figure 3.1. Au milieu, le coordinateur est chargé de mettre en

[(6)] Mininet : http://mininet.org/

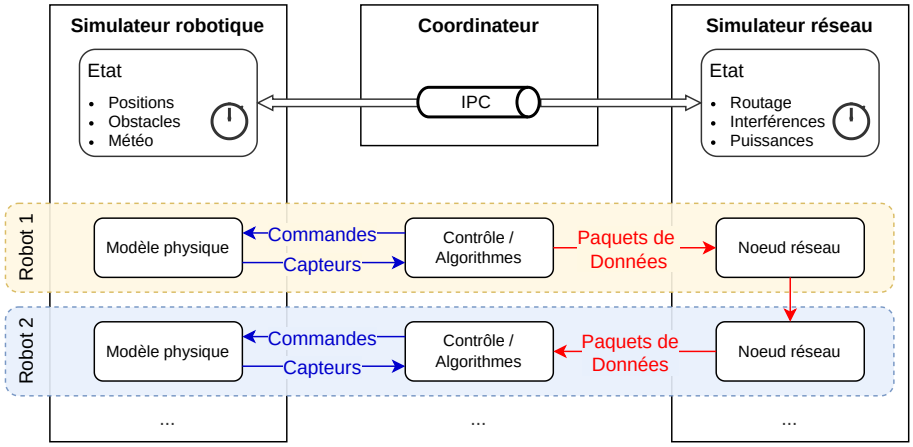

FIGURE 3.1. Architecture représentative d'un co-simulateur robotique et réseau

place un mécanisme de communication entre les deux simulateurs, et déclenche l'exécution ou l'arrêt de chaque simulateur en s'assurant que leurs horloges respectives restent synchronisées. Un robot est représenté par un modèle physique (gauche) et un modèle réseau (droite), ainsi que par un ensemble de programmes s'exécutant à bord du robot (milieu). Le coordinateur s'assure que les états dans l'espace et le temps de chaque paire de modèles restent cohérents tout au long de la simulation.
Les co-simulateurs mentionnés fournissent chacun des fonctionnalités différentes. Nous listons ici les fonctionnalités qui semblent utiles pour la simulation de systèmes multi-robots.

3.2. HARDWARE IN THE LOOP

Étudier un nouveau protocole ou une nouvelle technologie sur une version simulée d'un robot (SITL) peut mener à des erreurs ou des biais. Effectivement, les limitations du robot réel (puissance de calcul, énergie limitée, surchauffe des composants, etc.) doivent être prises en compte dans le développement. Un simulateur permettant le *Hardware In The Loop* (HITL) permet de surpasser ces limitations en exécutant le programme du robot directement sur son ordinateur embarqué. Seules les entrées (capteurs) et sorties (actionneurs) sont alors simulées. Ainsi, le plan de développement d'une nouvelle technique multi-robots est le suivant : une expérimentation rapide en simulation (SITL), suivie d'une vérification du matériel embarqué toujours exécutée dans un environnement simulé (HITL), puis une expérimentation sur le terrain qui valide la résilience de la technique développée au monde réel.

3.3. CONTRÔLE DU TEMPS

Il est utile, voire nécessaire, de pouvoir exécuter une simulation plus rapidement ou plus lentement que le temps réel. Il est aussi intéressant d'offrir la possibilité aux

utilisateurs d'enregistrer la simulation dans un format qui permet de la rejouer, de la mettre en pause ou de la jouer à l'envers. Cette fonctionnalité facilite l'échange de données de co-simulation et la reproductibilité des expériences de co-simulation.
La possibilité de s'exécuter à un rythme différent du temps réel nécessite généralement l'intégration d'une méthode de synchronisation entre les simulateurs qui prennent part à la co-simulation. Pour l'enregistrement et la capacité à rejouer les simulations, ROS propose par exemple les *rosbags*, des fichiers standards enregistrant l'ensemble des messages échangés et les états des robots, qui permettent de rejouer le scénario de simulation.

3.4. Intégration ROS

Le *Robot Operating System* (ROS) est omniprésent dans la recherche en robotique aujourd'hui. Il met à disposition des roboticiens un système d'échange de messages normalisé qui permet d'une part une communication efficace entre les modules d'un robot sous une forme *publisher / subscriber* et facilite d'autre part la réutilisation de modules sous forme de *packages*.
Il nous semble donc essentiel que le co-simulateur accueille facilement des robots exécutant ROS car celui-ci présente une grande bibliothèque de modules (capteurs, algorithmes, ...) pouvant être intégrés et interfacés avec n'importe quel robot.
Le protocole d'échange d'information a profondemment changé au passage de ROS1 à ROS2. ROS1 utilise un protocole ad-hoc développé spécifiquement pour la robotique, alors que le passage à ROS2 a été l'occasion d'utiliser un protocole standardisé, plus robuste et versatile : *Data Distribution Service*[(7)] (DDS). Initialement développé pour l'industrie et les systèmes cyber-physiques complexes, DDS s'adapte parfaitement aux communications internes d'un robot, mais il existe peu d'études validant son efficacité pour les systèmes multi-robots. Un co-simulateur robotique et réseau adaptable à ROS2 serait une opportunité de valider l'intérêt de DDS en le comparant à d'autres protocoles de réseau ad-hoc.

3.5. Performances

In fine, les co-simulateurs seront surtout évalués sur leur efficacité. Un co-simulateur doit éviter de surcharger les temps de calcul d'un simulateur multi-robots souvent déjà lourds. Bien sûr, la performance d'un co-simulateur dépend directement des performances des simulateurs qu'il connecte et le calcul d'une co-simulation ne sera jamais plus rapide que le plus lent de ses simulateurs, mais les méthodes de synchronisation et d'échange de messages, qui rajoutent inévitablement du temps de calcul, se doivent d'être légères, simples et rapides.

3.6. Exactitude et niveau de détail

La co-simulation ne doit pas changer les résultats de chaque simulateur pris séparément, on parle alors de co-simulation "exacte". Ici, l'exactitude désigne l'adéquation

(7) https://www.dds-foundation.org/

des résultats donnés par le co-simulateur par rapport aux résultats qu'auraient fournis chaque simulateur indépendemment, ce qui ne préjuge en rien du réalisme de chaque simulateur. La question du niveau de détail (réalisme) d'une simulation revêt d'une grande complexité, et a été étudié depuis longtemps notamment dans la théorie de la modélisation et simulation de Zeigler [36]. Dans [20], Heidemann et al. étudient en détails la simulation de réseaux sans fils et montrent que c'est la question scientifique qui doit dicter la finesse de la simulation. Dans certains cas, un haut niveau d'abstraction permet d'accélérer la simulation et d'éviter que les résultats recherchés ne soient noyés dans les artefacts d'une simulation trop détaillée. Dans d'autres cas, une simulation trop abstraite peut empêcher le chercheur de déceler les problèmes que son nouvel algorithme ou nouveau protocole pourrait rencontrer. La co-simulation ne modifie pas les modèles à l'intérieur de chaque simulateur, c'est donc au moment du choix des simulateurs robotique et réseau que le chercheur doit mener une réflexion critique sur le niveau de détail dont il a besoin pour travailler sur une question scientifique précise. Le co-simulateur peut en revanche fournir une garantie d'exactitude.

4. Co-simulateurs existants

Comme illustré dans la Section 2, nous disposons d'outils matures pour simuler d'une part la composante robotique, et d'autre part la composante réseau des systèmes multi-robots, mais aucun des simulateurs mentionnés n'intègre ces deux parties en un seul logiciel. Hors, comme nous l'avons vu, les composantes réseau et robotique ne peuvent pas être simulées séparément, car elles sont intrinsèquement liées. Les chercheurs se sont donc tournés vers la co-simulation (l'exécution simultanée de simulateurs différents) pour bénéficier de l'énorme travail déjà effectué sur les simulateurs existants dans chaque domaine. De plus, la Section 3 illustre l'architecture-type et les fonctionnalités importantes à inclure dans un co-simulateurs robotique et réseau.

Il est intéressant d'examiner les cas d'usages pour lesquels les co-simulateurs présentés ci-après ont été développés. On remarque que l'application des systèmes multi-robots aériens, ou flottes de drones, motive énormément le développement de co-simulateurs robotique et réseau. Nous expliquons cette omniprésence par trois raisons principales. D'abord, la robotique aérienne, comparée aux réseaux de robots terrestres, a des déplacements plus rapides et dans les trois directions de l'espace, entraînant des variations rapides du canal radio. Elle est aussi plus sensible aux pertes de lien radio. En effet, la perte du lien radio d'un robot volant mène le plus souvent à un crash, alors qu'un robot terrestre ne subirait que rarement des dommages dans une situation similaire. Enfin, certaines missions envisagées pour les "*flying ad-hoc networks*" (FANETs) impliquent une utilisation intensive des capacités radio des robots (collecte de données de capteurs, station de base 5G volante, relais d'antennes pour zones à faible couverture, etc.) [29].
Le tableau 4.1 synthétise notre étude en donnant pour chaque co-simulateur : les simulateurs de robotique et de réseau sur lesquels il se base, l'année de publication, le caractère *open-source*, la technique de synchronisation utilisée ainsi que la technologie

TABLE 4.1. Tableau comparatif des co-simulateurs robotique et réseau existants

Nom	Simulateur Multi-Robot	Simulateur Réseau	Synchronisation	Echanges d'information	Année	Open-Source	Actif	Réf.
RoboNet-Sim	ARGoS	NS-2 / NS-3	Time-stepped (Bidirectionnelle)	Socket (TCP et UDP)	2013	✓	✗	[23]
CUSCUS	FL-AIR	NS-3	No (real-time)	Shared Memory	2017	✓	✗	[37]
AVENS	X-Plane	OMNeT++	No	fichier XML partagé	2017	✓	✗	[26]
FlyNet-Sim	Ardupilot	NS-3	Time-stepped (Bidirectionnelle)	Message queue (ZMQ)	2018	✗	✗	[4]
CPS-Sim	Matlab Simulink	QualNet OMNeT++	Variable time-stepped (Bidirectionnelle)	Custom (SNSP)	2018	✗	?	[34]
GzUav	Gazebo	NS-3	Exécution séquentielle	Unix Domain Socket + socket TCP	2019	✓	✗	[14]
CORNET	Gazebo	NS-3	Variable time-stepped (Unidirectionnelle)	Message queue (ZMQ)	2020	✓	✗	[2]
ROS-NetSim	Tous	Tous	Sliding Window (Bidirectionnelle)	Unix Domain Socket	2021	✓	✗	[7]
CORNET 2.0	Gazebo	MiniNet	Variable time-stepped (Unidirectionnelle)	Message queue (ZMQ)	2022	✓	✓	[1] [3]
Synchro-Sim	Gazebo	NS-3	Variable time-stepped (Bidirectionnelle)	Non précisé	2022	✗	?	[10] [12]

de partage d'information entre les simulateurs. Les termes des colonnes Synchronisation et Echange d'Information seront explicités dans les sections 5 et 6.

4.1. CUSCUS

CUSCUS [37] est un co-simulateur qui interconnecte ns-3 avec un simulateur de physique (FL-AIR) développé en interne à l'équipe Heudiasyc de l'Université de Technologie de Compiègne. La force principale de CUSCUS réside dans l'utilisation de conteneurs Linux (LXC) pour isoler les différents acteurs de la co-simulation. Le programme de chaque drone s'exécute dans un conteneur, et les conteneurs sont reliés par des interfaces réseau virtuelles (TAP). Grâce au module *Tap bridge* de ns-3, les paquets qui traversent les interfaces TAP sont interceptés, traversent un réseau simulé, et sont finalement délivrés à l'interface TAP destination. Cette architecture permet d'utiliser les mêmes programmes embarqués sur les drones, seules les couches physique et MAC sont simulées. CUSCUS n'implémente pas de mécanisme de synchronisation entre les deux simulateurs, les deux simulateurs doivent donc s'exécuter au même rythme. Les auteurs montrent en revanche que le "délai architectural" (le délai dû à l'architecture de simulation) des paquets traversant le réseau est négligeable pour une simulation se déroulant en temps réel (de l'ordre de 70 μs). Si son architecture a des caractéristiques intéressantes, CUSCUS reste limité par son simulateur de physique *ad-hoc*, qui ne dispose pas de librairies de capteurs et d'actionneurs variés et régulièrement mises à jour dont peuvent disposer les outils communautaires comme Gazebo.

4.2. AVENS

AVENS [26] est présenté comme un outil de simulation dédié aux systèmes de robots aériens. Il est basé sur un modèle d'architecture nommé LARISSA [27], qui

permet de générer du code automatiquement pour le simulateur de réseau OMNeT++, qui est intégré avec un simulateur de physique dédié aux vehicules volants, X-Plane Flight Simulator. Les deux simulateurs utilisent un fichier XML partagé pour échanger des informations, comme la position des robots. Ce fichier est actualisé par un module X-Plane et régulièrement lu par un module OMNeT++, mais AVENS n'implémente pas de méthode de synchronisation, ce qui oblige les deux simulateurs à s'exécuter au même rythme. Les auteurs ne mentionnent pas la méthode de contrôle choisie, ni l'utilisation de ROS.

4.3. FlyNetSim

FlyNetSim [4] est un environnement de co-simulation dédié aux essaims de drones. Il utilise la pile logicielle Ardupilot [(8)] à laquelle il ajoute une simulation du réseau gérée dans ns-3. Un mécanisme de synchronisation bidirectionnelle y est implémenté : un timestamp est ajouté à chaque message et utilisé pour déterminer quel simulateur est "en avance" sur l'autre. Le simulateur le plus rapide est alors gelé pour attendre le simulateur le plus lent.
Les calculs de physique sont effectués dans un mini-simulateur de physique intégré à la version *Software-In-The-Loop* (SITL) d'Ardupilot, mais celui-ci n'est pas aussi avancé que les moteurs de physique utilisés dans des simulateurs comme Gazebo ou ARGoS. FlyNetSim possède par ailleurs un mode "émulation" dans lequel le programme de l'autopilote n'est plus exécuté par l'ordinateur mais directement sur le drone. Le réseau et les entrées des capteurs sont quant à eux toujours simulés. Cette capacité permet de passer rapidement de la phase de simulation à la phase d'expérimentation.

4.4. CORNET et CORNET 2.0

Introduit en 2020, CORNET [2] est un middleware de co-simulation capable d'interconnecter Gazebo et ns-3 pour le cas des flottes de drones. La deuxième version de ce logiciel, CORNET 2.0 [1], publiée en 2022, capitalise sur les leçons de cette première expérience. On y préfère Mininet-Wifi à ns-3 et des améliorations majeures sont apportées : CORNET 2.0 n'est plus limité aux robots aériens, est indépendant des simulateurs de physique et de réseau utilisés, ne dépend pas de ROS et profite du déploiement conteneurisé de Mininet, permettant de distribuer efficacement les calculs sur plusieurs machines.
La synchronisation temporelle est gérée par un plugin Gazebo. Cette approche ne peut fournir qu'une synchronisation unidirectionnelle du simulateur de réseau vers le simulateur de physique, ce qui engendre une grande contrainte : il faut que le simulateur de réseau fonctionne à un rythme plus élevé que le simulateur de physique.
Une mise à jour récente de CORNET 2.0 [3] propose de nouvelles améliorations ainsi que des exemples de cas d'usages. Un nouveau mécanisme de distribution des paquets permet notamment à cette dernière version d'être très réaliste concernant les délais des paquets, mais la synchronisation reste unidirectionnelle, ce qui ne permet pas au

[(8)] Ardupilot : https://ardupilot.org/

simulateur de réseau de fonctionner plus lentement que le simulateur de robotique. Malheureusement, cette hypothèse ne se vérifie pas dans le cas de réseaux fortement dynamiques, très denses ou avec des calculs de propagation réalistes [23].

4.5. RoboNetSim

RoboNetSim [23] est un environnement de simulation intégrant le simulateur de physique ARGoS et les simulateurs de réseau NS-2 ou ns-3. Créé en 2013, ce co-simulateur dédié à l'étude des grands essaims de robots (plusieurs dizaines) propose une synchronisation bidirectionelle et un échange de messages par socket. Les auteurs ont mené des études de performance et ont montré que : ns-3 est plus rapide que son alter-ego NS-2 ; le simulateur de physique est le facteur limitant si le nombre de robots ne dépasse pas un certain seuil (100 avec ARGoS et ns-3, sans détails sur les communications étudiées) ; le surcoût de temps de calcul de la co-simulation est inévitable, mais pas rédhibitoire ; le co-simulateur est "juste" tel que précédemment défini.

4.6. SynchroSim

SynchroSim [10] est un co-simulateur récent qui se focalise sur la simulation de systèmes multi-robot hétérogènes, avec l'objectif principal de simuler l'opération coordonnée et collaborative de robots terrestres et aériens sur le champ de bataille. Il utilise Gazebo et ns-3, ainsi que ROS 1.
Une extension de SynchroSim [12] exploite le système de communication de ROS2 pour rendre plus efficace la synchronisation entre les différents modules de la simulation.
La principale originalité de cette proposition réside dans sa méthode de synchronisation. SynchroSim introduit une synchronisation basée sur une fenêtre glissante à taille variable, ce qui règle en partie la problématique du choix de la taille de la fenêtre. La taille de la fenêtre est ajustée en fonction de la différence de vélocité entre deux agents, mais les auteurs ne justifient pas ce choix et ils n'apportent pas de preuve de justesse de leur co-simulation. Enfin, ce simulateur n'est pas *open-source*, rendant difficile son évaluation.

4.7. CPS-Sim

CPS-Sim [34] accorde Matlab Simulink (simulation physique) et les simulateurs réseau QualNet ou OMNeT++. Il se démarque des autres co-simulateurs par sa méthode de synchronisation par pas variable, qui permet des calculs plus précis au prix d'un temps de simulation élevé, en particulier quand le nombre d'événements réseau augmente (voir section 5). En revanche, le choix de Matlab/Simulink pour modéliser la physique prive CPS-Sim des avantages d'utilisation de simulateurs robotiques existants, comme une interface visuelle ou un contrôle direct des actionneurs.
Pour certains problèmes, la méthode de synchronisation de CPS-Sim apporte une justesse de résultats nécessaire. Par exemple, dans le cas de la simulation de protocoles

de synchronisation d'horloges dans un réseau de capteurs sans fil, la précision temporelle est à la fois très importante et sensible à la moindre erreur. Cette méthode de synchronisation est donc adaptée.

4.8. GzUav

GzUav [14] est un environnement de co-simulation qui modélise une flotte de drones communiquants en assemblant trois modules complémentaires : Gazebo gère la simulation physique ; Ardupilot gère la simulation de l'autopilote (SITL) ; ns-3 simule les échanges réseau. Il permet une synchronisation bidirectionnelle et sa charge de travail peut être distribuée facilement sur plusieurs ordinateurs en réseau. L'intérêt de ce simulateur est l'intégration *by design* du code de l'autopilote, ainsi que son mécanisme de synchronisation par exécution séquentielle qui assure sa "justesse" au prix d'un temps de simulation élevé (la simulation physique avance d'un pas, attend que l'autopilote et la simulation réseau aient terminé leurs calculs, puis passe au pas suivant). Pour un réseau inférieur à 100 robots, GzUav montre des performances acceptables. S'il est très spécifique aux flottes de robots volants, ce simulateur se place parmi les co-simulateurs les plus avancés.

4.9. ROS-NetSim

L'ambition de ROS-NetSim [7] est de connecter n'importe quel simulateur de physique avec n'importe quel simulateur de réseau, pour des scénarios multi-robots variés, en se chargeant de la synchronisation bidirectionnelle et de l'échange de messages. Ce co-simulateur cherche donc à rester le plus général possible et s'adapte à tous types de robots grâce à l'intégration de ROS. Sa méthode de synchronisation est à fenêtre fixe et il utilise la technologie Protobuf [(9)] pour l'échange d'information entre les simulateurs, une solution flexible et facilement paramétrable.
ROS-NetSim propose aussi une méthode d'abstraction des informations du canal radio : le simulateur de physique fournit au simulateur de réseau les informations nécessaires (chemins d'ondes multiples et pertes associées) pour calculer l'affaiblissement dû à la propagation.
Le grand intérêt de ROS-NetSim est sa transparence du point de vue du robot. Un robot ROS ne peut distinguer s'il évolue dans un environnement simulé ou réel. Quand il est dans le monde réel, son trafic réseau sera géré par sa carte réseau, alors que s'il est simulé, ce trafic sera intercepté, passera dans le simulateur de réseau puis sera réinjecté au nœud destinataire. Cette architecture permet d'utiliser le même programme en simulation et sur le robot, permettant une transition très fluide vers des expérimentations en milieu réel.

[(9)] Protocol Buffer : https://protobuf.dev/

4.10. Discussion

Notre étude des projets existants, synthétisée dans le tableau 4.1 et qui a été menée dans le cadre d'un projet de développement d'une flotte de robots aériens communicants, a montré qu'aucun co-simulateur n'a été suffisamment maintenu et documenté pour faire consensus dans le domaine et rassembler la communauté. En particulier, ils peinent à maintenir leur compatibilité avec les outils modernes (ROS2, Ubuntu, etc.) et sont souvent abandonnés une fois le projet terminé. Les changements de popularité des simulateurs spécifiques (surtout dans le domaine robotique) participent à la multiplication des co-simulateurs et justifient le développement de plateformes de co-simulation versatiles qui peuvent facilement être adaptées à d'autres simulateurs (ROS-NetSim). Les propriétés fournies par le co-simulateur (capacité à simuler en HITL, performances de calcul, etc.) sont le principal facteur de choix, en fonction des objectifs scientifiques de la simulation (préparation à l'expérimentation, passage à l'échelle d'un grand nombre de robots, etc.). Parmi ces propriétés, la synchronisation temporelle et l'échange d'information jouent un rôle crucial, notamment sur les performances globales de l'outil. Les Sections 5 et 6 donnent des détails techniques sur ces deux caractéristiques d'une architecture de co-simulation.

5. Synchronisation temporelle

La synchronisation des horloges entre plusieurs simulateurs est une problématique récurrente en co-simulation. Les simulateurs de robotique décrivent la dynamique et cinématique des modèles grâce à des équations différentielles, qui doivent être discrétisées pour être résolues de manière logicielle. Cette discrétisation est faite avec un temps d'échantillonnage, qui constitue la granularité de la simulation. Ces simulateurs sont dits *temps discret à pas constant* (Gazebo, ARGoS, Webots, etc.). Les simulateurs de réseau n'ont pas la contrainte d'un temps qui s'écoule régulièrement. Un nœud peut être silencieux pendant un temps long, puis déclencher une suite d'événements en décidant d'émettre un paquet. Dans ces simulateurs, le temps simulé avance d'événement en événement, le temps entre les événements ne provoque donc pas d'attente. Ce sont des simulateurs à *événements discrets* (ns-3, OMNeT++, QualNet, etc.).
Cette différence de nature rend nécessaire l'implémentation de mécanismes de synchronisation qui peuvent être unidirectionnels (un simulateur utilise l'horloge simulée de l'autre simulateur) ou bidirectionnels.
Cette section présente les techniques de synchronisation utilisées dans les co-simulateurs existants. Une représentation graphique de ces techniques est donnée en figure 5.1.

5.1. Time-stepped

C'est l'approche la plus simple : la synchronisation a lieu régulièrement avec une période multiple de la durée du pas du simulateur de physique (sur la figure 5.1, cette période est de 5 pas). A chaque synchronisation, les simulateurs échangent leurs statuts et variables puis avancent indépendamment jusqu'à la prochaine synchronisation. Cette méthode est simple à implémenter mais présente deux problèmes affectant la

justesse des calculs : un délai injustifié est introduit pour le simulateur de robotique et les informations de mobilité fournies au simulateur de réseau sont imprécises.
Le premier effet s'explique facilement : le simulateur de robotique est notifié de la réception d'un message seulement lors de la prochaine synchronisation. Il reçoit toujours les paquets avec un délai qui dépend de la taille de la fenêtre de synchronisation. Si les paquets arrivent de manière uniforme et aléatoire, le délai moyen introduit est donc de la moitié de la fenêtre de synchronisation (vérifié en pratique dans [23]).
Le deuxième problème est dû à la fréquence des mises à jour des positions dans le simulateur de réseau. En effet, les robots sont considérés immobiles durant toute la fenêtre de simulation, ce qui peut entraîner des imprécisions dans les calculs du simulateur de réseau. Cependant, cet effet ne représente en pratique que de faibles imprécisions [23].

5.2. Variable time-stepped

Pour améliorer la précision des calculs, il est possible d'adapter la méthode précédente en brisant la linéarité du simulateur de physique. La procédure est alors la suivante : si le simulateur de réseau possède un événement pendant la prochaine fenêtre du simulateur de physique, ce dernier se met sur pause. Le simulateur de réseau calcule l'événement puis envoie au simulateur de physique les informations de synchronisation et le *timestamp* de l'événement. Le simulateur de physique doit alors avancer jusqu'à ce *timestamp*. Avec cette méthode, le simulateur de physique obtient les informations des activités du réseau sans délai, ce qui représente un avantage dans les applications où le réseau et la robotique sont très intriqués. En revanche l'implémentation de pas variables dans le simulateur de physique se révèle souvent techniquement difficile. Les performances de cette méthode dépendent beaucoup de l'application étudiée : s'il y a peu d'événements réseau, les performances sont bonnes car il y a peu de synchronisation et donc peu d'échanges de messages. En revanche avec un grand nombre d'événements réseau, la surcharge de temps de calcul augmente dramatiquement.

5.3. Global Event-driven

Une approche radicalement différente consiste à gérer les deux simulateurs depuis un ordonnanceur unique. Chaque pas du simulateur de physique est alors traité comme un événement et mis dans la même file d'attente que les événements du simulateur de réseau. Les événements sont exécutés de manière séquentielle. L'avantage majeur de cette méthode est qu'elle n'engendre pas de délai supplémentaire ni d'approximations de calcul. En retour, les événements des deux simulateurs n'étant pas calculés simultanément, le coût en temps de calcul est décuplé.

6. Flux d'information entre les simulateurs

6.1. Technologie d'échange

Les simulateurs s'échangent des informations tout au long de la simulation, au minimum pour se synchroniser mais aussi pour collaborer. Ces échanges de messages

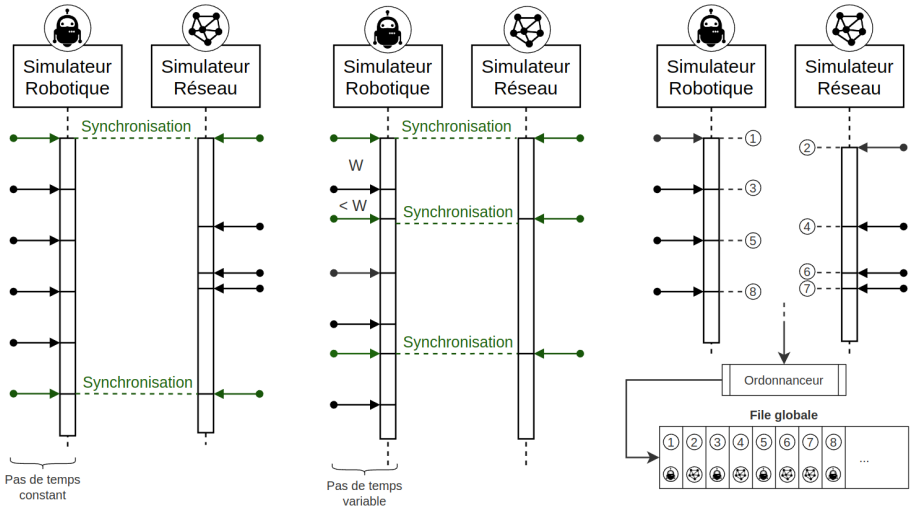

FIGURE 5.1. Méthodes de synchronisation de simulateurs de physique et de réseau. de gauche à droite : *time-stepped*, *variable time-stepped*, *global event-driven*

représentent l'essentiel de la surcharge de calcul générée par la co-simulation, il est donc important de choisir une technologie de communication judicieuse. Le tableau 4.1 précise les techniques de communication utilisées dans les co-simulateurs étudiés.

Si détailler les avantages et inconvénients de chacune de ces techniques sort du cadre de ce papier, nous discutons ci-après les techniques utilisées dans les co-simulateurs que nous avons étudiés.
Le choix de la technique de communication entre simulateurs sera avant tout déterminé par l'exécution distribuée du co-simulateur : s'il doit pouvoir s'exécuter sur plusieurs ordinateurs en réseau (pour partager la charge de calcul), l'utilisation de *sockets* Internet (UDP ou TCP) sera presque automatique. Dans le cas contraire, les autres techniques de communication inter-processus (*Inter-Process-Communication*, IPC) peuvent s'avérer plus efficaces.
Les *sockets* sont probablement la technique la plus versatile. Ils peuvent être utilisés localement ou en réseau (permettant des calculs distribués), sont indépendants de la plateforme (permettant l'exécution sur différents systèmes d'exploitation) et proposent des communications bloquantes ou non-bloquantes (ce qui rend possible la synchronisation des horloges).
Les files de messages présentent moins de charge de calcul que les *sockets* mais dépendent du système d'exploitation et peuvent être problématiques quand le nombre de messages est grand.
La mémoire partagée ne peut pas être utilisée si les calculs doivent être distribués sur plusieurs machines, mais est la technique la plus rapide en temps de calcul. Cependant,

cette technique requiert une organisation rigoureuse des accès en mémoire, complexifiant le développement du co-simulateur.

En plus de ces technologies de transport, le format des messages et le protocole de communication peuvent influer sur la performance globale du système. Des protocoles de communication inter-processus évolués, tels que DDS, pourraient être utilisés mais semblent disproportionnés par rapport à la tâche demandée. La complexité de ces protocoles rendrait la configuration et la maintenance difficiles et participerait à l'opacité de la solution.

6.2. Contenu des échanges

Le contenu des échanges entre les simulateurs relève aussi une importance déterminante. Il est crucial de correctement définir les informations échangées : partager trop d'information générera une surcharge du temps de calcul, mais en partager trop peu limite l'intrication des simulateurs, rendant le co-simulateur inutile. Par exemple dans notre cas, il est primordial que les positions des robots soient cohérentes entre la simulation de physique et la simulation de réseau.

Le concepteur du co-simulateur doit donc répondre à deux questions essentielles : quel est le contenu des messages échangés entre les deux simulateurs, et à quelle fréquence ces échanges ont-ils lieu ? Le problème de l'abstraction du canal radio est un exemple intéressant dans le cas des simulations de systèmes multi-robots. Pour déterminer la réussite de réception d'un paquet entre deux robots, le simulateur de réseau doit calculer un bilan de liaison et comparer la puissance reçue au seuil de réception du robot destinataire. Pour cela, il doit détenir une représentation du canal radio (distance, obstacles, interférences, etc.), qui soit cohérente avec la représentation du monde dans le simulateur de physique. Pour faire passer ces informations du simulateur de physique au simulateur de réseau, [7] propose une abstraction du canal qui cherche à compresser les informations essentielles nécessaires au calcul du bilan de liaison. Une autre approche consiste à dupliquer les obstacles du simulateur de physique dans le simulateur de réseau, qui peut donc calculer les bilans de liaison en prenant en compte les obstacles. Dans le cas d'obstacles statiques, cette méthode limite la quantité d'information transmise entre les deux simulateurs, et donc d'améliorer la vitesse de calcul globale.

7. Vers un co-simulateur dédié aux flottes de drones

Notre étude des projets existants, synthétisée dans le tableau 4.1 et qui a été menée dans le cadre d'un projet de développement d'une flotte de robots aériens communicants, a montré qu'aucun co-simulateur n'a été suffisamment maintenu et documenté pour faire consensus dans le domaine et rassembler la communauté. En particulier, ils peinent à maintenir leur compatibilité avec les outils modernes (ROS2, Ubuntu, etc.) et sont souvent abandonnés une fois leur but principal rempli. Dans le cadre du projet ANR/AID CONCERTO, nous avons donc débuté le développement d'un

environnement de co-simulation robotique et réseau adapté aux flottes de robots aériens. Nous avons choisi de nous baser sur les principes de co-simulation proposés par ROS-NetSim [7], qui nous semblent les plus concluants. Cependant, ROS-NetSim n'est qu'une preuve de concept et son intégration réelle avec un simulateur de robotique et de réseau nécessite le développement de connecteurs adaptés. Nous avons choisi les simulateurs Gazebo et ns-3, qui sont à ce jour les références de leur domaine respectif, et bénéficient tous les deux d'une communauté active et aidante, qui nous permet d'accélérer le développement de notre outil.

7.1. Synchronisation

Notre co-simulateur implémente, comme ROS-NetSim, une synchronisation bidirectionnelle gérée par un coordinateur. Nous avons simplifié l'architecture de ROS-NetSim en fusionnant les deux coordinateurs proposés en un seul coordinateur multithreadé (voir figure 7.1).
ROS-NetSim souffre d'un défaut important : il ne peut pas simuler des délais de messages plus petits que la taille de sa fenêtre de simulation, qui est commune entre le simulateur de robotique et le simulateur de réseau. Or, les échelles de temps de la simulation physique et réseau sont très différentes : il est généralement utile de connaître l'heure d'arrivée d'un paquet réseau avec une précision de l'ordre de la milliseconde ou moins, mais forcer le simulateur de physique à discrétiser ses équations différentielles avec un tel pas augmente énormément la charge de calcul. Pour palier ce problème, nous proposons de découpler la taille des pas de simulation entre les deux simulateurs. Cette technique permet d'améliorer la précision des délais des paquets échangés, tout en gardant des temps de calcul raisonnables pour le simulateur de robotique. Par exemple, le simulateur de réseau effectue dix itérations de 0.5 ms pendant que le simulateur de physique effectue une seule itération de 5 ms. La synchronisation, elle, reste assurée.
C'est le coordinateur qui est chargé de faire avancer l'horloge de la simulation, à chaque fois que le simulateur de réseau avance d'un pas. Cette horloge est partagée avec l'ensemble des nœuds ROS2 à travers le topic standard $/\mathrm{clock}$, qui devient l'horloge de référence pour tous les nœuds ROS2. Ce mécanisme de synchronisation, présenté en figure 7.1 permet d'exécuter la simulation plus ou moins vite que le temps réel, quel que soit le simulateur le plus lent.

7.2. Echange d'information

Comme dans ROS-NetSim, le flux d'information entre les deux simulateurs prend la forme de messages Protobuf configurables échangés à travers des *sockets* gérés par le coordinateur. Les *sockets* peuvent être de type UDS (*Unix Domain Sockets*) dans le cas d'une exécution locale, ou TCP dans le cas d'une exécution des modules de simulation distribuée sur plusieurs machines. A la différence de ROS-Net-Sim, c'est ns-3 qui calcule le bilan de propagation des ondes dans l'environnement, et les obstacles, fixes, sont initialisés dans Gazebo et ns-3 à l'initialisation de la simulation. Ainsi, il ne reste que la position des robots à envoyer de Gazebo vers ns-3.

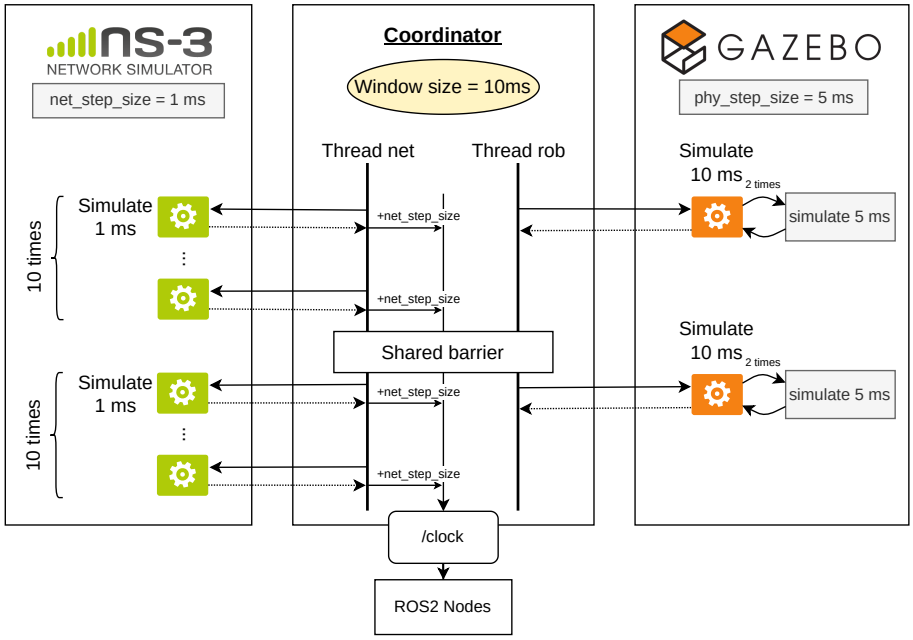

FIGURE 7.1. Principe de synchronisation par fenêtre permettant des pas de simulation découplés

7.3. SIMULATION DU ROBOT

Les robots aériens modernes embarquent généralement deux ordinateurs complémentaires : un "contrôleur de vol" (Autopilot) qui contient les fonctions de bas niveau permettant au drone de voler, et un "ordinateur de bord" qui exécute les algorithmes de déplacement de haut niveau et gère les fonctions liées à la mission. Pour obtenir une simulation réaliste, il faut que le code exécuté en simulation soit le plus proche possible du code qui sera exécuté sur chaque ordinateur lors des phases d'expérimentation. Pour cela, nous avons choisi d'utiliser la version SITL du contrôleur de vol open-source PX4 [(10)]. Ce programme s'occupe, en simulation comme dans la réalité, de la fusion des capteurs, de résoudre les équations de contrôle, ainsi que d'autres fonctions essentielles. La version SITL de l'autopilot est conforme à la version réelle à l'exception des valeurs d'entrée et de sortie : des capteurs simulés remplacent les véritables capteurs du drone et les commandes moteur sont envoyées au modèle simulé de Gazebo à la place de véritables moteurs.
L'ordinateur de bord est lui aussi simulé de manière fiable, puisqu'il embarque un système d'exploitation Linux et utilise l'environnement ROS2, qui sont facilement réplicables sur la machine de simulation.
Au niveau logiciel, il y a donc très peu de différences entre la simulation et le robot

[(10)] https://github.com/PX4/PX4-Autopilot

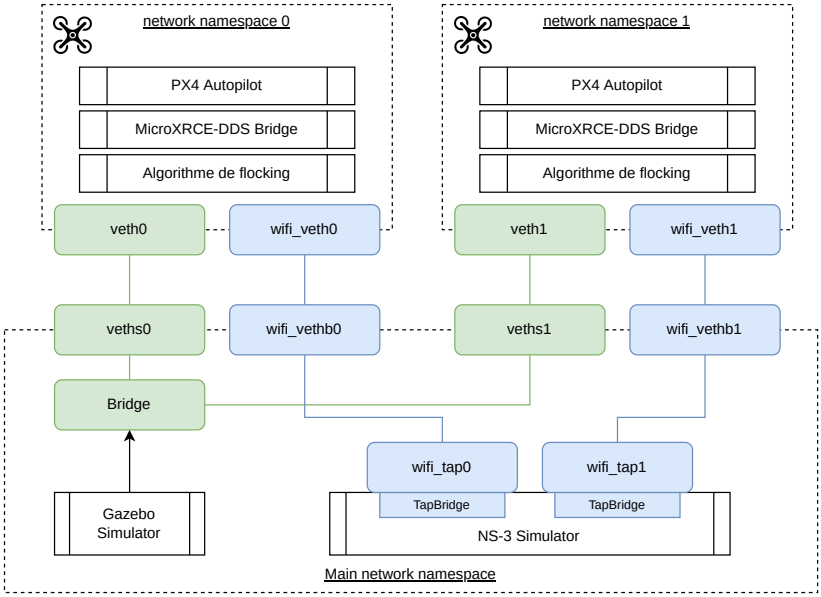

FIGURE 7.2. Architecture réseau de notre co-simulateur (2 drones)

réel, à l'exception de la puissance de calcul disponible à la fois pour le contrôleur de vol et l'ordinateur de bord.

7.4. ASPECT RÉSEAU

Il y a deux points importants à mentionner pour la simulation du réseau : l'architecture du système et la simulation réseau au sein de ns-3.

ARCHITECTURE SYSTÈME. --- Les protocoles de communication sont fortement liés à l'environnement dans lequel ils s'exécutent. Par exemple, le protocole DDS fonctionne différemment si les deux entités qui communiquent sont sur la même machine ou sur deux machines en réseau. Pour représenter fidèlement la présence de plusieurs ordinateurs, nous utilisons la virtualisation, qui permet d'émuler plusieurs machines virtuelles au sein de la machine physique qui exécute la simulation. Pour cela, nous utilisons les *network namespaces* (appelés ci-après "espaces réseau"), une fonction qui permet de dupliquer les fonctions réseau d'une machine Linux. On obtient ainsi des "machines virtuelles" qui ont chacune leurs fonctions réseau séparées, alors qu'elles sont hébergés sur une même machine. Les *network namespaces* sont préférés à des techniques de conteneurisation complètes (telles de Docker) pour leur facilité d'implémentation et leur performances. En revanche, ces espaces réseau ne permettent pas de contrainte la puissance de calcul disponible pour chaque robot. Cette technique de conteneurisation a fait ses preuves par le passé, dans le co-simulateur CUSCUS [37].

La figure 7.2 permet de visualiser les différents espaces réseau et les interfaces virtuelles nécessaires pour les connecter. Les deux espaces réseau du haut représentent les drones, ils hébergent l'ensemble des programmes qui s'exécutent à bord du drone (dont le contrôleur de vol). Le rectangle du bas représente l'espace réseau principal de la machine de simulation dans lequel Gazebo et ns-3 sont exécutés.
Pour connecter entre eux ces machines virtuelles, deux ponts de communication sont créés : un pont "direct" et un pont "Wi-Fi". Le pont "direct", en vert sur la figure 7.2, permet à Gazebo de partager avec les robots leur état (position, vitesse, collisions, etc.) ainsi que l'horloge de simulation. Le pont "Wi-Fi", en bleu, représente le lien radio entre les drones. Tous les messages qui transitent sur ces interfaces bleues sont interceptés et simulés dans ns-3.

SIMULATION DU RÉSEAU. --- Nous utilisons le module ns-3 "TapBridge", qui permet de faire le lien entre le réseau simulé et les machines virtuelles à travers une interface virtuelle de type "TAP". Ainsi, chaque message qui arrive sur une interface de ce type est traité par ns-3, qui simule sa transmission dans le réseau. Si le paquet arrive à destination, il est libéré à une autre interface "TAP", qui le transmet à la machine virtuelle concernée.
ns-3 est chargé de simuler les couches physique et transport. Les calculs de propagation, les différentes étapes protocolaires et les collisions sont donc des modèles ns-3.
La position des robots dans ns-3 est mise à jour grâce aux informations partagées par Gazebo à chaque fenêtre de simulation.

7.5. SIMULATION ILLUSTRATIVE

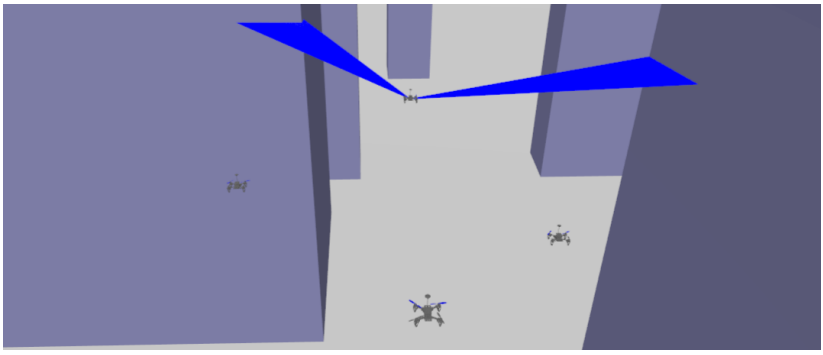

FIGURE 7.3. Co-simulation robotique et réseau d'une flotte de 4 quadricoptères exécutant un algorithme de vol en essaim (flocking) en présence d'obstacles.

La figure 7.3 illustre l'avancée de nos travaux. Nous avons mis en place une co-simulation de quatre drones quadricoptères, qui exécutent un algorithme distribué de vol en essaim (*flocking*) avec évitement d'obstacle basé sur un lidar embarqué. Chaque drone exécute l'algorithme VAT précédemment développé par notre équipe dans [6]. Les obstacles de l'environnement sont définis dans un fichier de configuration et créés

à l'identique dans Gazebo et ns-3. Leur forme est pour le moment limitée à des parallélépipèdes rectangles. Le contrôleur de vol *PX4 Autopilot* est utilisé dans sa version SITL pour la simulation des capteurs, des moteurs, et des équations de contrôle du drone.
Pour exécuter l'algorithme, chaque robot doit recevoir les informations de position et de vitesse de chacun de ses voisins. Chaque drone récupère la position de son modèle dans Gazebo (ce qui s'apparente à un "GPS parfait") et sa vitesse grâce à un filtre de Kalman étendu, implémenté par le *PX4 Autopilot*, qui fusionne les valeurs des capteurs simulés (IMU, magnétomètre, baromètre, etc.) pour calculer la pose du robot. Cette position et cette vitesse sont envoyées en *broadcast* UDP par l'ordinateur de bord, à travers un réseau Wi-Fi 802.11g simulé dans ns-3. La couche physique est simulée grâce au module ns-3 *Spectrum* qui permet de prendre en compte les obstacles et les interférences. Les conditions de chaque canal radio entre les drones sont recalculées toutes les 100 ms.
Une vidéo illustrant le fonctionnment du co-simulateur est disponible au lien suivant : https://youtu.be/f8b8ZZgmxMA. On peut y voir une flotte de 4 drones quadricoptères exécuter un vol en essaim dans les conditions expérimentales décrites ci-dessus. Les rayons du lidar d'un drone sont affichés en bleu.

8. Conclusion

Pour étudier les systèmes multi-robots communicants, nous avons montré la nécessité de simuler simultanément la physique et le contrôle des robots d'un côté, et les flux d'information traversant le réseau de l'autre.

L'étude des simulateurs de robotique et de réseau existants a montré que ces outils sont complexes et résultent d'années de développement. Il est donc essentiel de capitaliser sur ce travail en réutilisant les simulateurs qui font consensus dans leur communauté respective. Les équipes de recherche se sont donc tournées vers la co-simulation et nous avons présenté les neuf co-simulateurs qui nous semblent les plus aboutis. Ces projets répondent tous aux deux problématiques centrales de la co-simulation – synchronisation et échange de messages – et ont chacun leurs avantages et inconvénients. A notre connaissance, aucun n'a pour l'instant fait consensus dans la communauté multi-robots. En particulier, la maintenance de ces plateformes et leur compatibilité avec les outils modernes (ROS2, Ubuntu, etc.) est à notre sens une barrière difficile à surmonter. Nos travaux en cours pour le développement d'un co-simulateur basé sur Gazebo et ns-3 et réutilisant les principes de co-simulation de ROS-NetSim ont aussi été présentés.

Lors de notre étude, nous avons noté que la problématique des flottes de robots aériens connectés motive une grande partie des projets de co-simulateurs, mais les avancées dans ce domaine pourraient se révéler très utiles pour d'autres systèmes cyber-physiques complexes. Avec un effort de généralisation, il ne nous semble pas impossible de créer un co-simulateur versatile, complet et facilement utilisable qui permettrait d'adresser des cas d'usages variés dans les multiples domaines tels que la ville

connectée, les véhicules autonomes ou les réseaux de capteurs mobiles. Pour être réellement utilisé par la communauté, un soin important devra être apporté à l'expérience utilisateur, notamment en termes de qualité d'implémentation, de présence d'une documentation, de facilité d'installation et d'utilisation, et de versatilité.

BIBLIOGRAPHIE

[1] S. Acharya, B. Amrutur, M. Bharatheesha & Y. Simmhan, « CORNET 2.0 : A Co-Simulation Middleware for Robot Networks », in *COMSNETS*, 2022.

[2] S. Acharya, A. Bharadwaj, Y. Simmhan, A. Gopalan, P. Parag & H. Tyagi, « CORNET : A Co-Simulation Middleware for Robot Networks », in *COMSNETS*, 2020.

[3] S. Acharya, M. Bharatheesha, Y. Simmhan & B. Amrutur, « A Co-simulation Framework for Communication and Control in Autonomous Multi-Robot Systems », in *IROS*, 2023.

[4] S. Baidya, Z. Shaikh & M. Levorato, « FlyNetSim : An Open Source Synchronized UAV Network Simulator based on ns-3 and Ardupilot », in *Proceedings of the ACM MSWIM*, 2018.

[5] I. Bekmezci, O. K. Sahingoz & S. Temel, « Flying Ad-Hoc Networks (FANETs) : A survey », *Ad Hoc Networks* (2013), p. 1254-1270.

[6] A. Bonnefond, O. Simonin & I. Guérin-Lassous, « Extension of Flocking Models to Environments with Obstacles and Degraded Communications », in *IROS*, 2021.

[7] M. Calvo-Fullana, D. Mox, A. Pyattaev, J. Fink, V. Kumar & A. Ribeiro, « ROS-NetSim : A Framework for the Integration of Robotic and Network Simulators », *IEEE Robotics and Automation Letters* (2021).

[8] P. Cieślak, « Stonefish : An Advanced Open-Source Simulation Tool Designed for Marine Robotics, With a ROS Interface », in *OCEANS MTS/IEEE*, 2019.

[9] J. Collins, S. Chand, A. Vanderkop & D. Howard, « A Review of Physics Simulators for Robotic Applications », *IEEE Access* (2021), p. 51416-51431.

[10] E. Dey, J. Hossain, N. Roy & C. Busart, « SynchroSim : An Integrated Co-simulation Middleware for Heterogeneous Multi-robot System », in *DCOSS*, 2022.

[11] E. Dey, M. Walczak, M. S. Anwar & N. Roy, « A Reliable and Low Latency Synchronizing Middleware for Co-simulation of a Heterogeneous Multi-Robot Systems », 2022.

[12] E. Dey, M. Walczak, M. S. Anwar, N. Roy, J. Freeman, T. Gregory, N. Suri & C. Busart, « A Novel ROS2 QoS Policy-Enabled Synchronizing Middleware for Co-Simulation of Heterogeneous Multi-Robot Systems », in *ICCCN*, 2023, p. 1-10.

[13] E. J. F. Dickinson, H. Ekström & E. Fontes, « COMSOL Multiphysics® : Finite element software for electrochemical analysis. A mini-review », *Electrochemistry Communications* (2014).

[14] F. D'Urso, C. Santoro & F. F. Santoro, « An integrated framework for the realistic simulation of multi-UAV applications », *Computers & Electrical Engineering* (2019).

[15] A. Farley, J. Wang & J. A. Marshall, « How to pick a mobile robot simulator : A quantitative comparison of CoppeliaSim, Gazebo, MORSE and Webots with a focus on accuracy of motion », *Simulation Modelling Practice and Theory* (2022).

[16] F. Faure, C. Duriez, H. Delingette, J. Allard, B. Gilles, S. Marchesseau, H. Talbot, H. Courtecuisse, G. Bousquet, I. Peterlik & S. Cotin, « SOFA : A Multi-Model Framework for Interactive Physical Simulation », in *Soft Tissue Biomechanical Modeling for Computer Assisted Surgery*, 2012, p. 283-321.

[17] R. R. Fontes, S. Afzal, S. H. B. Brito, M. A. S. Santos & C. E. Rothenberg, « Mininet-WiFi : Emulating software-defined wireless networks », in *International CNSM*, 2015.

[18] B. P. Gerkey, R. T. Vaughan & A. Howard, « The Player/Stage Project : Tools for Multi-Robot and Distributed Sensor Systems », in *Proceedings of the ICAR*, 2003.

[19] S. Hayat, E. Yanmaz & R. Muzaffar, « Survey on Unmanned Aerial Vehicle Networks for Civil Applications : A Communications Viewpoint », *IEEE Communications Surveys & Tutorials* (2016).

[20] J. Heidemann, N. Bulusu, J. Elson, C. Intanagonwiwat, K.-C. Lan, Y. Xu, W. Ye, D. Estrin & R. Govindan, « Effects of Detail in Wireless Network Simulation », in *USC/ISI*, 2001.

[21] M. H. Kabir, S. Islam, J. Hossain & S. Hossain, « Detail Comparison of Network Simulators », *IJSER* (2014).

[22] N. Koenig & A. Howard, « Design and use paradigms for Gazebo, an open-source multi-robot simulator », in *IEEE/RSJ IROS*, 2004.

[23] M. Kudelski, L. M. Gambardella & G. A. Di Caro, « RoboNetSim : An integrated framework for multi-robot and network simulation », *Robotics and Autonomous Systems* (2013).

[24] J. Lächele, A. Franchi, H. H. Bülthoff & P. Robuffo Giordano, « SwarmSimX : Real-Time Simulation Environment for Multi-robot Systems », in *Simulation, Modeling, and Programming for Autonomous Robots*, Springer, 2012.

[25] M. M. M. Manhães, S. A. Scherer, M. Voss, L. R. Douat & T. Rauschenbach, « UUV Simulator : A Gazebo-based package for underwater intervention and multi-robot simulation », in *OCEANS MTS/IEEE*, 2016.

[26] E. A. Marconato, M. Rodrigues, R. d. M. Pires, D. F. Pigatto, L. C. Q. Filho, A. R. Pinto & K. R. L. J. C. Branco, « AVENS - A Novel Flying Ad Hoc Network Simulator with Automatic Code Generation for Unmanned Aircraft System », in *Hawaii International Conference on System Sciences*, 2017.

[27] E. A. Marconato, D. F. Pigatto, K. R. L. J. C. Branco & L. H. C. Branco, « LARISSA : Layered architecture model for interconnection of systems in UAS », in *ICUAS*, 2014.

[28] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker & J. Turner, « OpenFlow : enabling innovation in campus networks », *ACM SIGCOMM Computer Communication Review* (2008).

[29] M. Mozaffari, W. Saad, M. Bennis, Y.-H. Nam & M. Debbah, « A Tutorial on UAVs for Wireless Networks : Applications, Challenges, and Open Problems », *IEEE Communications Surveys & Tutorials* (2019), p. 2334-2360.

[30] C. Pinciroli, V. Trianni, R. O'Grady, G. Pini, A. Brutschy, M. Brambilla, N. Mathews, E. Ferrante, G. Di Caro, F. Ducatelle, M. Birattari, L. M. Gambardella & M. Dorigo, « ARGoS : a modular, parallel, multi-engine simulator for multi-robot systems », *Swarm Intelligence* (2012).

[31] G. F. Riley & T. R. Henderson, « The ns-3 Network Simulator », in *Modeling and Tools for Network Simulation*, Springer, 2010.

[32] E. Rohmer, S. P. N. Singh & M. Freese, « V-REP : A versatile and scalable robot simulation framework », in *IEEE/RSJ IROS*, 2013.

[33] S. Shah, D. Dey, C. Lovett & A. Kapoor, « AirSim : High-Fidelity Visual and Physical Simulation for Autonomous Vehicles », in *Springer Proceedings in Advanced Robotics*, 2018.

[34] A. Suzuki, K. Masutomi, I. Ono, H. Ishii & T. Onoda, « CPS-Sim : Co-Simulation for Cyber-Physical Systems with Accurate Time Synchronization », in *IFAC NECSYS*, 2018.

[35] A. Varga & R. Hornig, « An overview of the OMNeT++ Simulation Environment », in *ICST*, 2010.

[36] B. P. Zeigler, H. Praehofer & T. G. Kim, *Theory of Modeling and Simulation*, 2000.

[37] N. R. Zema, A. Trotta, G. Sanahuja, E. Natalizio, M. Di Felice & L. Bononi, « CUSCUS : An integrated simulation architecture for distributed networked control systems », in *IEEE CCNC*, 2017.

Abstract. --- Simulation of multi-robots systems require the integration of both the robotic and network components. Leveraging the existing tools from each community seems an evidence, but merging two completely different simulators proves challenging. This paper is a state of the art study on co-simulators that tackle this problematic, and we provide details on the challenges arising when one wants to create an efficient and useful co-simulator. We also present our ongoing work toward the creation of such a simulator.

Keywords. --- Multi-robots systems, Co-simulation, Communication networks.
