Sistema de comunicacions RTPS entre robots en ROS 2

Marc Macanás Manuel

29 de juny de 2025

Resum – Aquest treball es centra en l'estudi i l'anàlisi del sistema de comunicació utilitzat en robots que fan servir el framework ROS 2 (Robot Operating System 2), mitjançant l'estàndard RTPS (Real-Time Publish-Subscribe). L'objectiu principal és avaluar el rendiment i el comportament de la comunicació entre diferents dispositius, en diversos escenaris de xarxa. Per tal de mesurar i comparar paràmetres clau com la latència, l'ús de CPU i memòria per determinar quina tecnologia s'adapta millor als sistemes robòtics distribuïts.

Paraules Clau – ROS2, DDS, RTPS, QoS, middleware.

Abstract – This work focuses on the study and analysis of the communication system used in robots that use the ROS 2 (Robot Operating System 2) framework, using the RTPS (Real-Time Publish-Subscribe) standard. The main objective is to evaluate the performance and behavior of communication between different devices, in various network scenarios. In order to measure and compare key parameters such as latency, CPU and memory usage to determine which technology is best suited to distributed robotic systems.

Keywords – ROS2, DDS, RTPS, QoS, middleware

1 Introducció - Estat de l'art

Amb l'evolució de ROS 2 [1], el framework de robòtica més utilitzat actualment, donat que a partir d'aquest 2025 es deixarà de donar suport a ROS, es va adoptar l'estàndard DDS (Data Distribution Service) [2] com a base per a la comunicació entre els diferents nodes. DDS permet una arquitectura distribuïda, escalable i amb opcions de qualitat de servei (QoS) [3]. On les seves implementacions més comunes en ROS 2 són: Fast DDS [4] i Cyclone DDS [5]. Aquestes solucions poden presentar desafiaments quant a consum de recursos, complexitat de configuració i rendiment en entorns amb restriccions de xarxa o maquinari limitat. Donat que Fast DDS, desenvolupat per eProsima, és l'opció per defecte, està molt completa i personalitzable, ideal per a sistemes en temps real i entorns industrials. Pel que fa a Cyclone DDS, desenvolupat per Eclipse Foundation, destaca pel seu rendiment en xarxes locals amb bona latència. Davant aquestes limitacions, ha sorgit una alternativa que és Zenoh [6], un middleware més flexible i lleuger, dissenyat per a funcionar en xarxes locals com entorns cloud o amb restriccions de connectivitat.Cal recalcar que la versió per defecte ha anat canviant en les diferents versions de ros 2 i encara és un debat obert quina d'elles és millor.

Actualment, existeixen estudis que comparen els diferents middleware, però la majoria no exploren a fons escenaris reals amb configuracions optimitzades ni avaluen el seu comportament en diferents condicions de la xarxa. Per tant, aquest treball busca cobrir aquest buit mitjançant una anàlisi experimental de diversos middleware, avaluen el seu rendiment amb mètriques clau i proposant millores pràctiques en la seva configuració.

A més, totes aquestes implementacions es poden optimitzar mitjançant els seus fitxers d'optimització.

En Fast DDS, la configuració es fa mitjançant un fitxer XML anomenat fastdds.xml, on es poden definir perfils de comunicació. Els paràmetres més importants són: reliability, que assegura l'enviament dels missatges o es prioritza la rapidesa, durability, que manté les dades disponibles si el receptor no està actiu o siguin volàtils, i history.kind i deth, que controlen quants missatges es guarden al subscriber.
Cyclone DDS també utilitza un fitxer XML (cyclonedds.xml) per configurar aspectes generals, com la interfície de xarxa (NetworkInterfaceAddress), el temps entre reintents d'enviament (ResendPeriod) i la mida màxima dels missatges (MaxMessageSize). No obstant això, els paràmetres de qualitat de servei (QoS) s'han de definir directament des del codi en ROS 2.
En el cas de Zenoh, es fa servir un fitxer zenoh.json5, on es configuren els protocols de transport (transport.builtin), el mode de connexió (mode: peer, client o router), i la fiabilitat de l'enviament. També s'inclou un control de congestió (congestion_control), on s'utilitzen diferents algorismes per adaptar la velocitat d'enviament segons les condicions de la xarxa.

2 Objectius

La realització d'aquest treball té com a objectiu principal l'estudi, l'anàlisi i l'optimització dels

- *E-mail de contacte: 1600670@uab.cat*
- *Menció realitzada: Tecnologies de la Informació*
- *Treball tutoritzat per: Bernat Gaston Braso (Àrea de Ciències de la Computació i Intel·ligència artificial)*
- *Curs 2024/25*

diferents middleware de comunicació utilitzats en ROS 2, amb especial èmfasi en DDS. Per això, s'han definit els següents objectius específics:

- Estudiar l'arquitectura i els components principals del sistema ROS 2 durant el primer mes del projecte[A1], mitjançant la revisió de documentació oficial i pràctiques amb entorns simulats, per comprendre el seu funcionament i aplicacions en robòtica.
- Comparar durant les primeres quatre setmanes del projecte almenys tres implementacions de DDS (Fast DDS, Cyclone DDS, Zenoh), identificant i avaluant les seves característiques clau com el rendiment, la fiabilitat i la compatibilitat.
- Establir i mesurar abans de la setmana 8 les mètriques de latència, nombre de paquets i bytes, consum de CPU i memòria, i fiabilitat en escenaris d'error, utilitzant eines de monitoratge per comparar les implementacions DDS en entorns ROS 2.
- Aplicar durant les setmanes 9 i 10 almenys tres configuracions diferents de paràmetres QoS als middleware DDS, amb l'objectiu de millorar l'eficiència i reduir l'ús de recursos, avaluant els resultats amb mètriques clau.
- Crear i executar abans de la setmana 17 un conjunt de proves Publisher-Subscriber en diversos escenaris de xarxa (bona cobertura Wi-Fi, mala cobertura Wi-Fi i VPN), per mesurar i comparar el comportament de les implementacions DDS.

3 Metodologia

Per a la realització d'aquest projecte s'utilitzarà la metodologia DevOps, un enfocament que combina desenvolupament (Dev) i operacions (Ops) amb l'objectiu d'automatitzar, optimitzar i millorar la implementació i anàlisi de sistemes. Tot i que tradicionalment s'aplica en el desenvolupament de programari en entorns empresarials, en aquest projecte es farà servir per millorar l'eficiència en l'execució de proves i l'anàlisi del rendiment de ROS 2 amb DDS.
Els principis clau de DevOps que es tindran en compte són els següents:

- **Integració Contínua (CI):** s'implementarà una estratègia d'integració contínua per automatitzar l'execució de proves en ROS 2. Això permetrà validar de manera sistemàtica diferents configuracions i garantir la fiabilitat dels resultats obtinguts en cada execució.
- **Monitoratge i Observabilitat: e**s desplegaran eines per monitorar i mesurar en temps real el rendiment del sistema, incloent-hi latència, ús de CPU, consum de memòria i tràfic de xarxa.
- **Proves Automatitzades: l**'execució de proves es durà a terme de manera automatitzada per assegurar-ne la repetibilitat i evitar errors humans. Això facilitarà la comparació de resultats entre diferents entorns i configuracions de comunicació.
- **Ajust i Optimització Iterativa:** s'aplicarà un procés d'optimització iterativa, on es realitzaran ajustos en la configuració de les diferents implementacions de DDS i es tornaran a executar les proves per analitzar-ne l'impacte. Aquest enfocament permetrà obtenir conclusions sòlides sobre l'eficiència i el rendiment de cada solució en diferents escenaris.

A més d'utilitzar la metodologia DevOps, també es faran servir altres tecnologies que ajudaran a la realització del d'aquest treball, com podria ser WireShark.

5 Cas Pràctic

5.1. Maquinari

Per la realització de les proves i les simulacions amb ROS 2, s'ha utilitzat un entorn controlat compost per dues màquines virtuals configurades amb el sistema operatiu Ubuntu 22.04 LTS, sobre les quals s'ha instal·lat la versió ROS 2 Jazzy (distribució més recent i oficialment suportada per aquesta versió d'Ubuntu).
Les màquines virtuals estan allotjades en un mateix host, la qual cosa permet controlar d'una manera més eficient les condicions de xarxa i facilitar així l'experimentació amb diferents configuracions de middleware. Cada VM actua com un node independent del sistema, on una s'encarrega del rol de Publisher, i l'altra del rol de Subscriber, comunicant-se a través de la xarxa virtual creada per l'hipervisor.
A nivell de recursos, cada màquina disposa de 2 CPUs virtuals, 4 GB de memòria RAM i 20 GB d'emmagatzematge, prou per a executar ROS 2 Jazzy juntament amb proves de publicació i recepció de grans volums de dades. A més, s'han instal·lat eines complementàries com Wireshark i llibreries com matplotlib, time, psutil, rclpy.qos, cv_bridge i OpenCV, que seran necessàries per capturar el trànsit de la xarxa, visualitzar mètriques i generar gràfics, a més de mesurar la latència, consum de CPU i memòria i configurar la qualitat de servei de ROS 2.

5.2. Xarxes

Pel que respecta a la xarxa, per avaluar el rendiment dels diferents middleware es faran servir tres escenaris de xarxa, per tal de simular diferents entorns reals que podria operar un sistema robòtic distribuït.

Escenari 1: Xarxa WiFi de bona qualitat
Aquest escenari simula un entorn de xarxa sense fils estable i ràpid, com la què es podria trobar en un laboratori o una instal·lació moderna. Les condicions de la xarxa que simulen un bon senyal són:

- Alta intensitat de senyal: major a -60 dBm

- Baixa latència: < 10 ms
- Taxa de pèrdua de paquets: pràcticament 0%
- Velocitat de descàrrega/pujada: ≥ 50 Mbps
- Jitter baix: < 5 ms

Escenari 2: Xarxa Wifi de baixa qualitat
Aquest escenari simula un entorn de xarxa sense fils més advers, com el que es pot trobar en les zones més allunyades de l'encaminador o xarxes saturades o amb diferents interferències. Les condicions de la xarxa que simulen un mal senyal són:

- Intensitat de senyal baix: entre -70 i -85 dBm
- Latència variable: entre 30 ms i 150 ms
- Pèrdua de paquets ocasional: fins a 5-10%
- Velocitat reduïda: ≤ 10 Mbps
- Jitter elevat: > 20 ms

Aquest escenari serà ideal per a analitzar com els middleware es comporten en situacions de xarxa inestables i si les seves configuracions QoS permeten mantenir la integritat dels missatges.

Escenari 3: Connexió mitjançant VPN
Aquest cas simula la comunicació entre robots distribuïts geogràficament, o en xarxes corporatives segures on la connexió passa per una xarxa privada virtual (VPN). Les característiques típiques d'aquest escenari són:

- Latència mitjana-alta: 40 ms – 100 ms (depenent del servidor VPN)
- Velocitat moderada: 10–30 Mbps
- Lleuger jitter i possible pèrdua de paquets
- Major overhead a causa del xifratge dels paquets

Pel fet que les proves es faran en màquines virtuals allotjades en un mateix host i connectades al mateix WiFi, no es podran reproduir fidelment tots els escenaris desitjats. Per tant, per tal de simular una xarxa WiFi de baixa qualitat tant en l'escenari 2 i 3, s'utilitzarà l'eina tc dins de les màquines virtuals per introduir, de manera controlada, latència addicional, pèrdua de paquets i limitacions d'amplada de banda a la interfície de xarxa. Fent així, que es pugui recrear entorns reals amb condicions de comunicació adverses, sense necessitat de modificar la xarxa física.
Per dur a terme aquest canvi de qualitat s'utilitzarà la següent comanda:

tc qdisc add dev <interfície> root netem delay 300ms 100ms loss 10% rate 512kbit

Aquest comandament tc afegeix una política de cua (qdisc) del tipus netem a la interfície indicada, aplicant-hi un retard mitjà de 300 mil·lisegons amb una variació aleatòria (jitter) de 100 ms a cada paquet, una pèrdua de paquets del 10%, i limitant l'amplada de banda a 512 kbit/s. Aquesta configuració permet emular una xarxa inestable amb latència elevada, pèrdues i restricció de velocitat.

5.3. Mètriques d'estudi

Durant les proves, es mesuraran diverses mètriques per avaluar el rendiment de la comunicació a la xarxa en diferents escenaris. Les mètriques que es consideraran són les següents:

Latència de transmissió
La latència es mesurarà utilitzant la biblioteca Time, en concret les funcions time_sent i time_received. Aquestes funcions permetran calcular el temps que triga un paquet a desplaçar-se des de l'origen fins a la destinació, proporcionant una mesura clau de la velocitat de resposta de la xarxa.

Nombre de paquets i bytes
Aquesta mètrica s'obtindrà fent servir Wireshark, una eina d'anàlisi de xarxa que permet capturar i mesurar el nombre de paquets enviats i rebuts, així com el volum total de dades (en bytes) que circulen per la xarxa. Aquesta informació ajudarà a avaluar l'eficiència de la transmissió de dades.

Consum de CPU i memòria
S'utilitzarà la biblioteca psutil per obtenir dades sobre el consum de CPU i memòria durant les proves. Es faran servir les funcions cpu_percent() per mesurar l'ús de la CPU i virtual_memory() per obtenir informació sobre l'ús de la memòria del sistema. Aquestes mètriques seran útils per analitzar l'impacte de la comunicació en el rendiment de les màquines virtuals.

Fiabilitat en escenaris d'errors
Per avaluar la fiabilitat, se simularan desconnexions en el Subscriber i es configuraran polítiques de QoS Reliability. La biblioteca rclpy.qos permetrà configurar el perfil de QoS mitjançant les classes QoSProfile, QoSReliabilityPolicy i QoSHistoryPolicy. Aquestes configuracions ajudaran a analitzar com es comporta el sistema quan es produeixen errors o desconnexions, i com les polítiques de fiabilitat influeixen en la comunicació.

Per tant, les mètriques seleccionades són clau per avaluar el rendiment de les diferents implementacions de ROS2 en diversos escenaris. La latència de transmissió és fonamental per mesurar la velocitat de resposta en temps real, mentre que el nombre de paquets i bytes ajuda a analitzar l'eficiència de la xarxa. El consum de CPU i memòria permet identificar l'impacte en el rendiment del sistema, i la fiabilitat en escenaris d'errors és essencial per comprendre com el sistema suporta fallades i desconnexions, aspectes crítics en entorns de robòtica. Aquestes mètriques proporcionen una visió integral de la capacitat de ROS2 per gestionar la comunicació en situacions reals.

5.4. Proves a realizar

Tal com s'ha anat detallant al llarg del projecte, es duran a terme diverses proves pràctiques, cadascuna enfocada a analitzar un aspecte concret de la

comunicació i el rendiment. Per tant, les proves que es realitzaran seran les següents:

Prova 1 : Publisher/Subscriber
Aquesta prova servirà com a base per mesurar el comportament general de ROS 2 en un escenari simple i estable. La qual consistirà a implementar un sistema de comunicació bàsic entre un Publisher i un Subscriber enviant missatges curts de tipus string.
L'objectiu és mesurar la latència de transmissió, el nombre de paquets i bytes, així com el consum de recursos del sistema.

Prova 2: Publisher/Subscriber amb gran quantitat de dades
Aquesta prova anirà un pas més que la prova anterior, però s'enviaran missatges amb dades de gran volum (strings d'1 MB).
L'objectiu és observar com afecta la mida del missatge a la latència, l'ús de recursos i l'estabilitat de la comunicació.

Prova 3: Captura de la càmera
Aquesta prova consistirà a capturar imatges de la càmera del portàtil en temps real i enviar-les a través del Publisher cap al Subscriber mitjançant la llibreria OpenCV. Aquesta prova s'utilitza per simular un escenari comú en robòtica, on una càmera envia fluxos d'imatges per a processament o visualització remota.

Prova 4: Utilització d'una VPN (OpenVPN)
En aquesta prova, s'establirà una connexió VPN entre les dues màquines virtuals mitjançant OpenVPN per verificar com afecta la xarxa virtual privada a la latència, fiabilitat i ús de recursos. Aquesta prova és clau per simular escenaris en què robots han de comunicar-se a través d'Internet o xarxes externes protegides. Per aquesta prova, un cop configurada la VPN, s'executarà la prova 1 per fer la comprovació.

5.5. Optimització
Amb l'objectiu de millorar el rendiment i l'estabilitat de la comunicació en entorns amb latència elevada, pèrdua de paquets o gran volum de dades, s'han realitzat configuracions específiques per a cadascuna de les implementacions utilitzades: Fast DDS, CycloneDDS i Zenoh. Aquestes optimitzacions es fan mitjançant fitxers de configuració propis de cada tecnologia, els quals permeten ajustar paràmetres com la fiabilitat, la retenció d'historial, la interfície de xarxa o la gestió de la congestió.

Fast DDS
Fast DDS permet definir perfils detallats mitjançant un fitxer XML fastdds.xml. Els paràmetres més rellevants són:

- reliability: on aplicarem el paràmetre RELIABLE donat que volem que els missatges es lliurin de manera garantida.
- durability: el qual aplicarem el paràmetre TRANSIENT_LOCAL donat que permet que les dades es mantinguin disponibles si el subscriber no està actiu.
- history.kind i depth: controlen quan historial de missatges es conserva per al subscriber el qual aplicarem el paràmetre KEEP_LAST i 10.

CycloneDDS
En el cas de CycloneDDS, la configuració es fa mitjançant un fitxer XML (cyclonedds.xml) que permet establir preferències generals com la interfície de xarxa, la mida màxima dels missatges i els períodes de reenviament. Tot i això, els paràmetres QoS específics (com RELIABLE o KEEP_LAST) s'han de definir des del codi mitjançant la classe QoSProfile de ROS2. Paràmetres destacats del fitxer:

- NetworkInterfaceAddress: especifica la interfície de xarxa utilitzada en el nostre cas aplicarem el paràmetre auto o la interfície que vulguem.
- ResendPeriod: estableix la freqüència amb què es reintenten enviaments, en el meu cas serà de 100 ms, ja que és un bon equilibri per a entorns on vols fiabilitat però sense saturar la xarxa amb massa reintents.
- MaxMessageSize: defineix la mida màxima dels missatges, útil en transmissions pesades, que en el meu cas aplicaré el valor 65536 que és el valor límit teòric màxim per UDP.

Zenoh
Zenoh utilitza un fitxer en format JSON5 (zenoh.json5) per configurar el transport, el mode de funcionament i diversos paràmetres de rendiment. Paràmetres rellevants:

- transport.builtin: defineix els protocols de transport, com TCP o UDP.
- mode: en aquest cas aplicarem el paràmetre peer per permetre una comunicació simètrica entre els nodes.
- reliability: activarem aquesta funció la qual permet assegurar que no es perdin dades.
- congestion_control: permet definir l'estratègia de gestió de congestió, en el meu cas s'utilitzarà l'algorisme Cubic, ja que s'adapta de manera intel·ligent la velocitat d'enviament segons la latència i pèrdues.

6. Comparació de resultats
6.1. Prova 1 : Publisher/Subscriber
En la realització de la primera prova la qual consisteix en la comunicació entre el Publisher i el Subscriber amb l'enviament de missatges curts de tipus string he obtingut els següents resultats en les diferents mètriques d'estudi:

Latència de transmissió
Pel que fa a l'anàlisi de la latència en les diferents implementacions[Fig.3], s'ha pogut comprovar que Fast DDS manté una latència bastant estable fins i tot

quan es produeixen variacions en la qualitat de la connexió WiFi. Aquesta estabilitat indica que Fast DDS és més resistent a les fluctuacions de la xarxa i ofereix un rendiment consistent en entorns amb condicions variables de connectivitat.
D'altra banda, la implementació basada en Zenoh ha estat la que ha experimentat una variació més significativa en la latència quan la qualitat del WiFi es veu afectada. En concret, s'ha observat un empitjorament del 151,7%, cosa que significa que la latència gairebé es va doblar en condicions de connexió menys òptimes. Aquest resultat suggereix que Zenoh és més sensible als canvis en la qualitat de la xarxa i que pot patir una degradació notable del seu rendiment en entorns amb connectivitat inestable.
Finalment, en el cas de la implementació amb Cyclone DDS, també s'ha registrat un augment de la latència quan la qualitat del WiFi empitjora, però aquest increment ha estat molt menys preocupant, situant-se en un 29,37%. Això indica que, encara que Cyclone DDS pateix un cert impacte per les fluctuacions de la xarxa, la seva capacitat per mantenir un rendiment raonablement estable és millor que la de Zenoh, tot i que no arriba al nivell de resistència de Fast DDS.

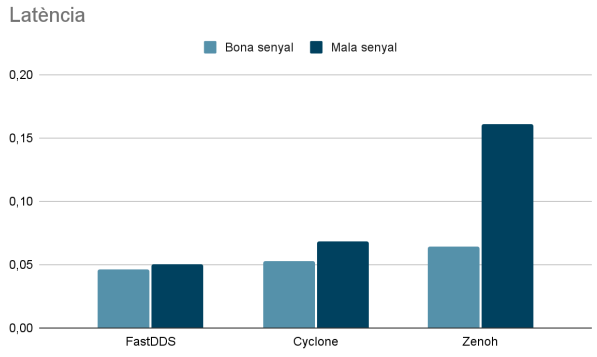


Fig. 3: Diagrama de latència

Nombre de paquets
Durant l'anàlisi de la comunicació, s'observen diferents paquets amb una mida de 226 bytes que corresponen a la transmissió d'informació pròpia del missatge. Paral·lelament, també es detecten paquets de 602 i 610 bytes, els quals estan relacionats amb la gestió de la connexió entre nodes.
En la fase inicial de la prova, no es detecta cap pèrdua de paquets. Tanmateix, quan s'inicia una fase de degradació de la connexió, es manté la integritat de la transmissió gràcies a l'activació de les polítiques de Qualitat de Servei (QoS), concretament la política de fiabilitat (Reliability). Aquesta configuració fa que el sistema resisteixi la inestabilitat de la xarxa i garanteixi que els missatges s'entreguin correctament sense necessitat de retransmissions, ja que el middleware s'encarrega de conservar els paquets fins que són confirmats pel receptor.
En canvi, en les proves realitzades amb Cyclone DDS i Zenoh, sí que s'observa la retransmissió de paquets. Aquest comportament es pot explicar per diversos motius tècnics:

- En el cas de Cyclone DDS, quan s'utilitza QoS Reliability = Reliable, el sistema implementa la fiabilitat a través del protocol RTPS sobre UDP. Si un paquet no és reconegut (ACK) pel receptor, es considera perdut i se'n programa la retransmissió, ja que Cyclone DDS utilitza temporitzadors, mecanismes de heartbeat i missatges NACK per controlar aquesta fiabilitat.
- En el cas de Zenoh, tot i no ser una implementació DDS pròpiament dita, ofereix funcionalitats de comunicació eficients i lleugeres. Quan es configura per garantir la fiabilitat, pot retransmetre automàticament els missatges que no han estat confirmats.

Consum de CPU i memòria
Pel que fa al consum de CPU de les diferents implementacions, s'han mesurat els valors en dos escenaris: quan el senyal WiFi és bona i quan és de baixa qualitat, així com la normalització dels valors restablint la connexió al seu estat original, per poder comparar-los més fàcilment[Fig.4].
En l'escenari amb bon senyal, Fast DDS presenta un consum molt baix de CPU, situant-se en 1,25%, fet que indica una eficiència alta en condicions òptimes. Cyclone, en canvi, ja consumeix més recursos, amb un valor de 4,27%, mentre que Zenoh es troba en un punt intermedi amb un consum de 2%.
Quan la qualitat del senyal empitjora, es produeix un increment significatiu en el consum de CPU per a totes les implementacions. Fast DDS incrementa el consum fins a 2,37%, gairebé el doble respecte al bon senyal, tot i que encara es manté dins de límits baixos. Cyclone mostra un augment molt més acusat, arribant al 21,4%, gairebé cinc vegades més que en bones condicions, fet que indica una gran dependència de la qualitat de la connexió i un possible sobreesforç per mantenir la comunicació. Zenoh també incrementa el seu consum, passant al 4,96%, un augment que encara que és més moderat que el de Cyclone, reflecteix una certa vulnerabilitat a la degradació del senyal.
Finalment, si mirem els valors normalitzats —que serveixen per comparar l'eficiència relativa en condicions variables— Fast DDS manté un valor molt baix, 1,21%, confirmant la seva eficiència en l'ús de recursos, fins i tot amb mal senyal. Cyclone, amb 4,2%, mostra que el seu consum és molt elevat en comparació amb les altres dues opcions, cosa que podria afectar la seva escalabilitat i viabilitat en entorns amb connectivitat inestable. Zenoh, amb un valor de 2,5%, es situa en una posició intermèdia, millor que Cyclone però sense arribar a l'eficiència de Fast DDS.

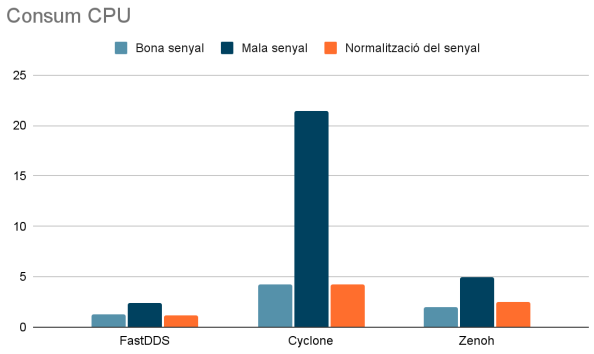


Fig. 4: Diagrama de CPU

En l'escenari de bona connexió[Fig.5], Fast DDS mostra un ús de memòria del 42,6%, el valor més baix entre les tres implementacions, la qual cosa indica una gestió eficient dels recursos de memòria. Cyclone, per la seva banda, presenta un consum més elevat, del 51,51%, i Zenoh se situa en un nivell intermedi, amb un 45,4%. Quan la connexió empitjora, l'ús de memòria de totes tres implementacions es manté gairebé estable, amb variacions molt mínimes. Fast DDS augmenta lleugerament fins al 42,8%, Cyclone també incrementa lleugerament fins al 52,21%, i Zenoh puja fins al 45,7%. Aquestes petites diferències indiquen que, en general, el consum de memòria no es veu gaire afectat per la qualitat de la connexió WiFi en cap de les tres implementacions.

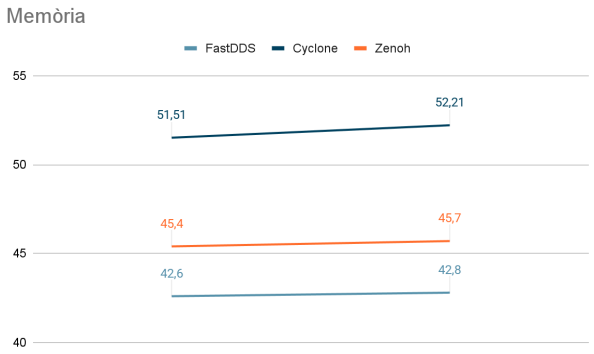


Fig. 5: Diagrama de memòria

6.2. Prova 2: Publisher/Subscriber amb gran quantitat de dades

En la realització de la segona prova la qual consisteix en la comunicació entre el Publisher i el Subscriber amb l'enviament de missatges amb dades de gran volum (strings d'1 MB) he obtingut els següents resultats en les diferents mètriques d'estudi:

Latència de transmissió

A partir de les proves realitzades[Fig.6], s'observa que Fast DDS és l'opció que ofereix una latència mitjana més baixa i estable en ambdós escenaris, amb només 0,117 ms en condicions de bon senyal i 5,82 ms en condicions adverses, demostrant així una elevada eficiència i robustesa davant situacions de degradació de la connexió. Zenoh presenta també un bon rendiment, amb una latència de 0,163 ms en escenari favorable i 8,003 ms en escenari desfavorable, posicionant-se com una alternativa equilibrada entre eficiència i capacitat d'adaptació, en aquestes dues implementacions veiem que tenim un empitjorament similar en termes relatius (prop de 50 vegades més lent), però Fast DDS es manté sempre amb una latència més baixa. En canvi, Cyclone DDS és la implementació que experimenta un impacte més gran davant la mala qualitat de la connexió, ja que la seva latència passa de 0,239 ms a 18,75 ms que provoca que tingui un empitjorament d'unes 78,8 vegades més lent, fet que indica una major sensibilitat a les condicions de xarxa adverses, probablement a causa d'una gestió menys eficient dels mecanismes de retransmissió i fiabilitat.

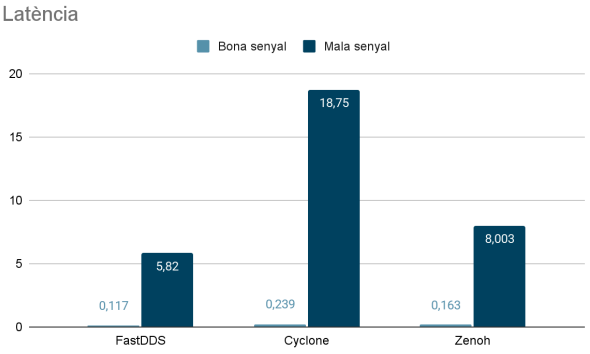


Fig. 6: Diagrama de latència

Nombre de paquets

Respecte al tràfic generat durant la prova que ha sigut capturat mitjançant l'eina Wireshark s'ha observat que els missatges es fragmenten en múltiples paquets UDP amb mides de 422 i 678 bytes a causa de la limitació de l'MTU. En aquests paquets apareixen diferents submissatges RTPS com DATA_FRAG, que transporta fragments del missatge original, permetent garantir la fiabilitat sobre UDP. També s'observen els submissatges INFO_DST, que identifica el destinatari del missatge mitjançant un GUID, INFO_TS, que aporta marques de temps per a la sincronització, i HEARTBEAT, que el publisher emet per indicar quins missatges ha publicat, ajudant el subscriber a detectar possibles pèrdues. Durant la degradació intencionada de la xarxa, s'han registrat paquets ICMP amb el missatge "Time-to-live exceeded", senyal que alguns paquets han estat descartats per excedir el nombre de salts permesos, fet que ha provocat l'aparició de múltiples paquets amb els missatges NACK_FRAG i ACKNACK a causa de la pèrdua de fragments, generats pel subscriber per tal de sol·licitar la retransmissió dels fragments perduts. Tot i això, alguns missatges han aconseguit arribar correctament i, un cop restablerta la connectivitat, la comunicació s'ha normalitzat, amb la retransmissió dels fragments perduts i la continuació del flux de dades sense errors. Aquest comportament demostra l'eficàcia dels mecanismes de control de fiabilitat de RTPS per garantir el lliurament ordenat de missatges.

Consum de CPU i memòria

L'anàlisi del consum mitjà de CPU en els diferents escenaris de prova mostra diferències notables entre les implementacions avaluades[Fig.7]. Fast DDS és la solució amb el consum més baix en tots els casos, amb un 12,29 % en condicions de bon senyal, que augmenta fins al 40,37 % amb mala connexió —és a dir, un increment de 28,08 punts percentuals—, i es redueix posteriorment fins a un 11,94 % quan la connexió es restableix, demostrant una gran eficiència i estabilitat. Zenoh es comporta de manera intermèdia, amb un consum de 19,66 % en escenari favorable que s'incrementa en 29,9 punts fins al 49,56 % amb mal senyal, mantenint posteriorment un valor acceptable del 24,73 % un cop recuperada la qualitat de la connexió. En canvi, Cyclone DDS presenta el consum més elevat: parteix d'un 42,03 % amb bon senyal i augmenta dràsticament fins al 81,21 % quan la connexió empitjora, el que representa un empitjorament de 39,18 punts percentuals; fins i tot després de la normalització del senyal, es manté en un valor alt de 41,43 %.

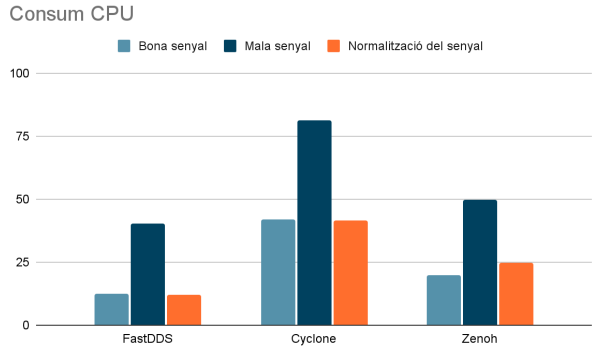


Fig. 7: Diagrama de CPU

A partir de les dades obtingudes sobre l'ús de la memòria en els diferents escenaris de connexió[Fig.8], s'observa que totes tres implementacions —Fast DDS, Cyclone DDS i Zenoh— mostren un comportament força estable, amb increments molt lleus quan la connexió empitjora. Fast DDS passa d'un ús de 44,6 % amb bona connexió a 45,9 % amb mala connexió, cosa que representa un augment molt moderat de només 1,3 punts percentuals. Zenoh segueix una tendència similar, amb un increment de només 1,37 punts (de 47,55 % a 48,92 %), mentre que Cyclone DDS és qui presenta el consum més alt tant en bona com en mala connexió, passant de 53,91 % a 56,06 %, amb un augment lleuger de 2,15 punts percentuals. Aquestes diferències mostren que, a diferència del consum de CPU i de la latència, l'ús de la memòria no es veu significativament afectat per la qualitat de la connexió. En aquest sentit, totes tres implementacions gestionen la memòria de manera estable, tot i que Fast DDS destaca novament com la implementació més lleugera pel que fa a l'ús de recursos, seguida de Zenoh i, finalment, Cyclone DDS, que manté un consum sistemàticament superior.

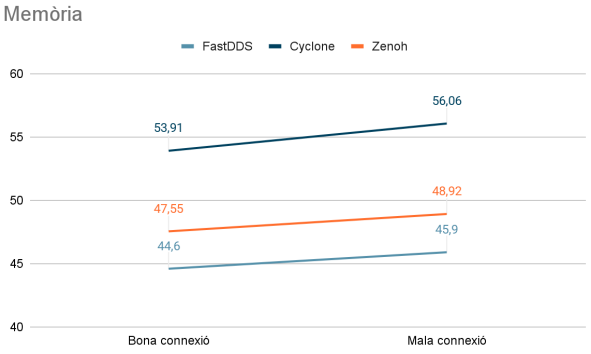


Fig. 8: Diagrama de memòria

6.3. Prova 3: Captura de la càmera

Aquesta prova consisteix a capturar imatges de la càmera del portàtil en temps real i enviar-les a través del Publisher cap al Subscriber.

Latència de transmissió

A partir de les dades recollides sobre la latència en la transmissió d'imatge mostra diferències notables[Fig.9]. En condicions de bona connexió, Fast presenta una latència de 459,09 ms, Cyclone de 493,8 ms, i Zenoh de 534,7 ms. Amb mala connexió, totes les implementacions experimenten un augment significatiu, però Zenoh és el més afectat (1473,5 ms), seguit de Fast DDS (1267,56 ms), mentre que Cyclone DDS es manté més estable amb 1108,2 ms.

A més, durant aquesta fase de mala connexió s'observa una pèrdua considerable de paquets, fins al punt que la imatge transmesa per la càmera es congela, indicant que el sistema no és capaç de garantir una transmissió contínua i fluida en entorns adversos. Aquest comportament posa de manifest que, malgrat la capacitat de Fast DDS per estabilitzar la latència en condicions favorables, la seva robustesa es veu compromesa sota escenaris de mala connectivitat, especialment en aplicacions sensibles com la transmissió de vídeo en temps real. El mateix li succeeix a Zenoh el qual podem veure que és el més vulnerable en condicions adverses. En quant a Cyclone podem observar que millora el seu rendiment i no es veu tan afectat.

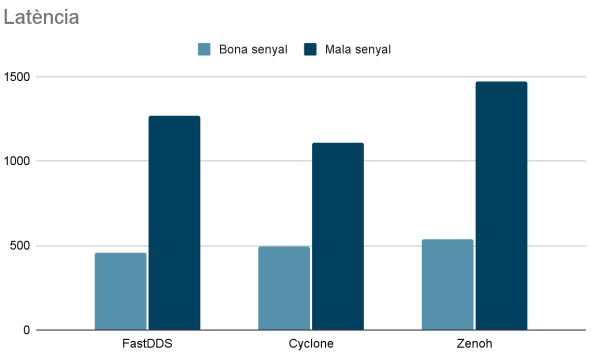


Fig. 9: Diagrama de latència

Nombre de paquets
En la prova amb la càmera s'ha observat un comportament similar al de la prova anterior. Durant l'enviament de dades, s'han detectat paquets fragmentats amb mides de 422 i 1410 bytes que contenien submissatges del tipus DATA_FRAG associats al tema rt/camera_frame, així com missatges de control HEARTBEAT per al mateix tema. Quan la connexió ha empitjorat, s'ha interromput l'enviament de missatges i han començat a aparèixer paquets ICMP amb l'error "Time-to-live exceeded", indicador que els paquets no podien arribar correctament a destinació. Aquesta interrupció ha tingut un efecte visible en temps real, ja que la imatge de la càmera s'ha quedat congelada. Un cop restaurada la connexió, el sistema ha tornat a enviar els missatges que havien quedat pendents, reprenent així la comunicació i recuperant l'estat normal de transmissió.

Consum de CPU i memòria
Pel que fa al processament, s'observa un increment generalitzat en el consum de CPU en escenaris de mala connexió[Fig.10]. Cyclone és el més exigent en tots dos casos (51,4% en bona connexió i 73,8% en mala), degut a la seva arquitectura intensiva en control d'estat i fiabilitat. Fast se situa en segon lloc (41,75% i 60,3%), mentre que Zenoh destaca per ser el més lleuger (34,6% i 49,2%). Finalment, un cop el senyal es normalitza, el consum de CPU es redueix fins al 37,9% en el cas de Fast, de 47,3% en el cas de Cyclone i per Zeno un 31,4%, indicant que el sistema recupera eficiència i torna a un ús més baix de recursos.

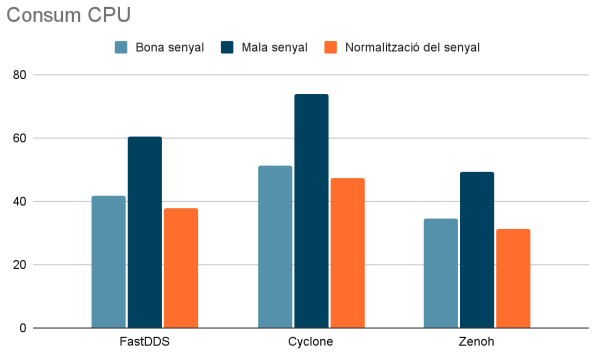


Fig. 10: Diagrama de CPU

Quant a la memòria, les diferències són més moderades però significatives[Fig.11]. En bona connexió, Cyclone DDS torna a ser el més exigent (72,8%), seguit de Zenoh (68,9%) i Fast DDS (64,7%). Amb mala connexió, tots tres mostren una lleugera reducció de consum (entre 1 i 2 punts percentuals), fet que pot explicar-se per la pèrdua de paquets que provoca que es gestionin menys dades o es descarreguin els buffers.

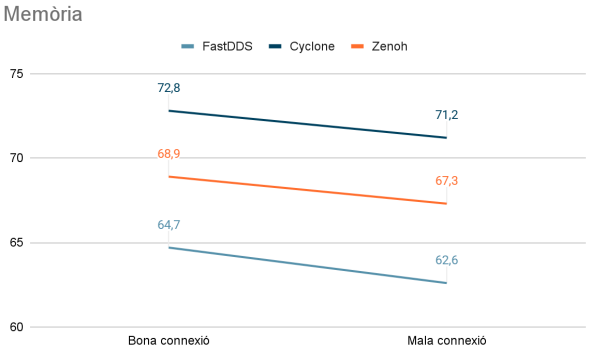


Fig. 11: Diagrama de memòria

6.4. Prova 4: Utilització VPN
En la realització de la quarta prova, la qual consisteix en la comunicació entre el Publisher i Subscriber amb l'enviament de missatges curts de tipus string a través de la connexió per VPN, s'han obtingut els següents resultats en les diferents mètriques d'estudi:

Latència de transmissió
Pel que fa a l'anàlisi de la latència en les diferents implementacions[Fig.12], s'ha pogut comprovar que Fast DDS manté una latència relativament estable fins i tot quan es produeixen variacions en la qualitat de la connexió WiFi. El seu increment ha estat d'un 32,64%, passant de 74,33 ms a 98,6 ms, la qual cosa indica una resistència raonable davant les fluctuacions de xarxa.
D'altra banda, Zenoh és la implementació que ha experimentat un empitjorament més notable de la latència, passant de 69,55 ms a 104,24 ms, la qual cosa representa un increment del 49,87%. Aquest resultat suggereix que Zenoh és força sensible als canvis en la qualitat de la xarxa i pot patir una degradació significativa del rendiment en entorns amb connectivitat inestable.
Finalment, Cyclone DDS ha mostrat un increment més moderat del 20,89%, passant de 79,3 ms a 95,88 ms, cosa que el posiciona com una solució intermèdia: no tan robust com Fast DDS, però més estable que Zenoh en condicions de degradació de la connexió.

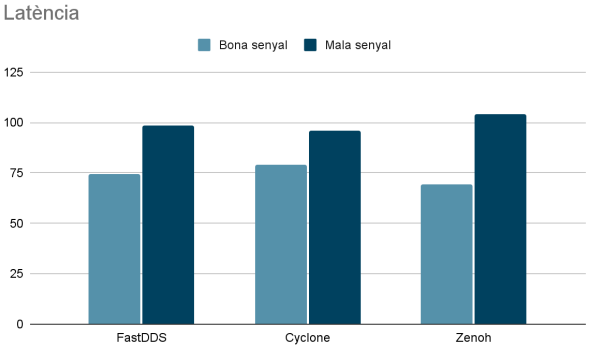


Fig. 12: Diagrama de latència

Nombre de paquets
Durant l'anàlisi de la comunicació, s'han observat

paquets amb una mida de 320 bytes corresponents a la transmissió d'informació pròpia dels missatges. També s'han detectat paquets de 700 bytes associats a la gestió de la connexió entre nodes.
Aquest increment de bytes respecte a l'enviament de paquets realitzat a la prova 1 és degut a l'encapsulament VPN.
En l'escenari de bona connexió, s'han detectat algunes pèrdues de paquets. Quan la connexió empitjora, les implementacions activen les polítiques de Qualitat de Servei (QoS), principalment la política de fiabilitat (Reliability), que garanteix l'entrega correcta dels missatges. En el cas de Fast DDS, la retransmissió no és necessària, ja que el middleware conserva els paquets fins que són confirmats pel receptor.
Per contra, Cyclone DDS i Zenoh sí que mostren retransmissions. Cyclone DDS, amb la configuració de Reliability establerta a Reliable, garanteix la fiabilitat mitjançant el protocol RTPS sobre UDP, retransmetent els missatges quan no reben una confirmació (ACK). Zenoh, malgrat no ser una implementació DDS pròpiament dita, també ofereix retransmissions automàtiques per garantir la fiabilitat quan es configura adequadament.

Consum de CPU i memòria
Pel que fa al consum de CPU[Fig.13], s'han mesurat els valors en els escenaris de bona i mala connexió i posteriorment la seva normalització per fer una comparació eficient.
En l'escenari amb bon senyal, Fast DDS presenta un consum de 8,65%, sent el més eficient. Cyclone DDS mostra un consum superior de 12,5%, mentre que Zenoh se situa en una posició intermèdia amb un 10,78%. En quant la qualitat empitjora, totes les implementacions es veuen afectades augmentant el seu consum de CPU. On Fast DDS arriba a un 15%, Cyclone a un 24,29% i Zenoh un 20,3%. Aquestes dades mostren que Fast DDS, tot i duplicar pràcticament el consum, encara es manté per sota de les altres dues implementacions. Cyclone DDS, en canvi, mostra una forta dependència del rendiment respecte a la qualitat de la connexió, amb un augment del 94%. Zenoh també mostra un increment significatiu, però menor que el de Cyclone.
En quant els valors normalitzats podem observar com Fast obté un 9,21%, Cyclone un 11,46% i Zenoh un 10,34%.
Aquestes xifres indiquen clarament que Fast DDS és l'opció més eficient pel que fa a l'ús de CPU, mentre que Cyclone DDS presenta un consum relativament elevat.

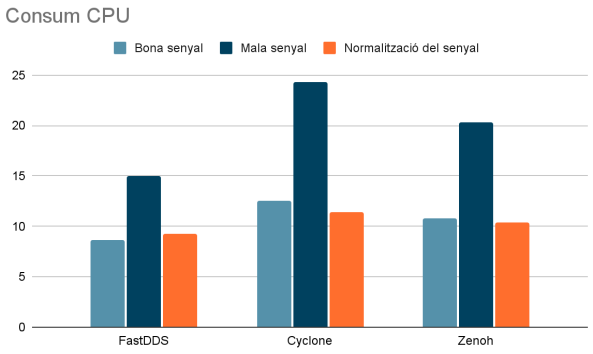


Fig. 13: Diagrama de CPU

Pel que fa a l'ús de memòria, en condicions de bona connexió[Fig.14], Fast consumeix un 53,7%, Cyclone consumeix un 63,5% i Zenoh es manté en 58,3%. Quan la qualitat empitjora, l'ús de memòria augmenta lleugerament fins a un 55% en Fast, en un 65,78% en Cyclone, i en Zenoh fins a 60,45%.
Aquestes variacions són poc significatives, cosa que indica que l'ús de memòria es manté relativament estable en totes les implementacions, independentment de la qualitat de la connexió WiFi.

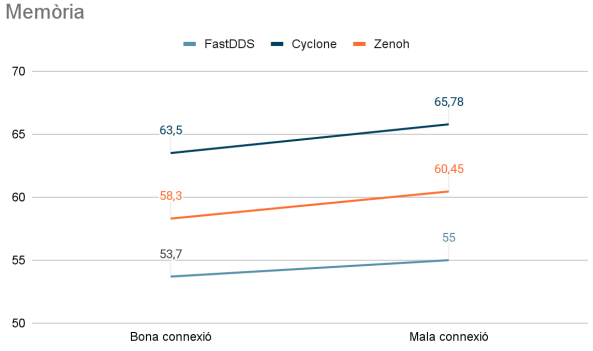


Fig. 14: Diagrama de memòria

8 CONCLUSIONS

Un cop dutes a terme les quatre proves amb les diferents implementacions de DDS, es poden extreure conclusions clares respecte a latència, consum de CPU i consum de memòria, tenint en compte l'estat de la connexió en cada escenari. A continuació, es presenten les observacions més rellevants per a cada categoria d'anàlisi.
En la primera prova, orientada a l'enviament simple de missatges, Fast DDS mostra la latència més baixa tant en condicions de bona com mala connexió. Per contra, Zenoh és el que més empitjora quan la qualitat de la connexió disminueix. Pel que fa al consum de CPU, Fast DDS torna a ser el més eficient, mentre que Cyclone DDS experimenta un increment notable del consum amb senyal deficient, fet que denota una baixa resiliència. Quant a la memòria, Fast DDS també destaca pel seu baix consum, mentre que Cyclone DDS és la implementació que més recursos consumeix. En conjunt, Fast DDS es posiciona com la solució més

eficient i estable per a enviaments senzills, ja que manté una letència, ús de CPU i memòria baixos, fins i tot amb les variacions de qualitat de la connexió.
A la segona prova, centrada en l'enviament de cadenes de text d'1MB, Fast DDS torna a oferir els millors valors de latència, mentre que Zenoh presenta una latència inferior a Cyclone DDS en escenaris de mala connexió, tot i continuar per darrere de Fast DDS. Pel que fa al consum de CPU i memòria, Fast DDS continua sent la més eficient, mentre que Cyclone DDS arriba a valors molt elevats, sent la implementació més exigent. Per tant, Fast DDS torna a ser l'opció més recomanable per a l'enviament de volums moderats de dades.
En la tercera prova, corresponent a l'enviament en viu d'imatges de càmera, Fast DDS obté la millor latència amb bona connexió, però amb connexió deficient, Cyclone DDS ofereix millors resultats en aquest aspecte. En termes de consum de CPU, Zenoh és l'opció més eficient, mentre que Cyclone DDS torna a destacar negativament pel seu elevat consum. Quant a la memòria, Cyclone DDS continua sent el més pesat, mentre que Fast DDS presenta el consum més baix. Així doncs, en aquest escenari no hi ha una implementació clarament guanyadora: si es prioritza una bona latència en condicions òptimes, Fast DDS és preferible; si es dona importància a l'eficiència de CPU, Zenoh és l'opció més adequada.
Finalment, a la quarta prova, realitzada mitjançant una connexió VPN, Zenoh presenta la millor latència en condicions favorables, però empitjora significativament quan el senyal és deficient. En canvi, Fast DDS manté una latència més estable i amb valors relativament propers. En relació amb el consum de CPU i memòria, Fast DDS és novament la implementació més eficient. En aquest cas, tot i que Fast DDS ofereix un millor equilibri global entre latència i ús de recursos, Zenoh continua sent una bona alternativa quan es prioritza la latència, tot i que a cost d'un consum més elevat.
En conclusió, Fast DDS és la implementació més eficient i equilibrada, destacant en gairebé totes les proves per la seva baixa latència, estabilitat i consum moderat de recursos. Zenoh sobresurt per la seva eficiència en CPU i per oferir bones prestacions en escenaris amb xarxes distribuïdes, com en la prova amb VPN. En canvi, Cyclone DDS presenta un comportament inconsistent i un alt consum de recursos, fet que la fa menys recomanable en entorns amb variabilitat en la connexió o limitacions de maquinari.

9 Agraïments

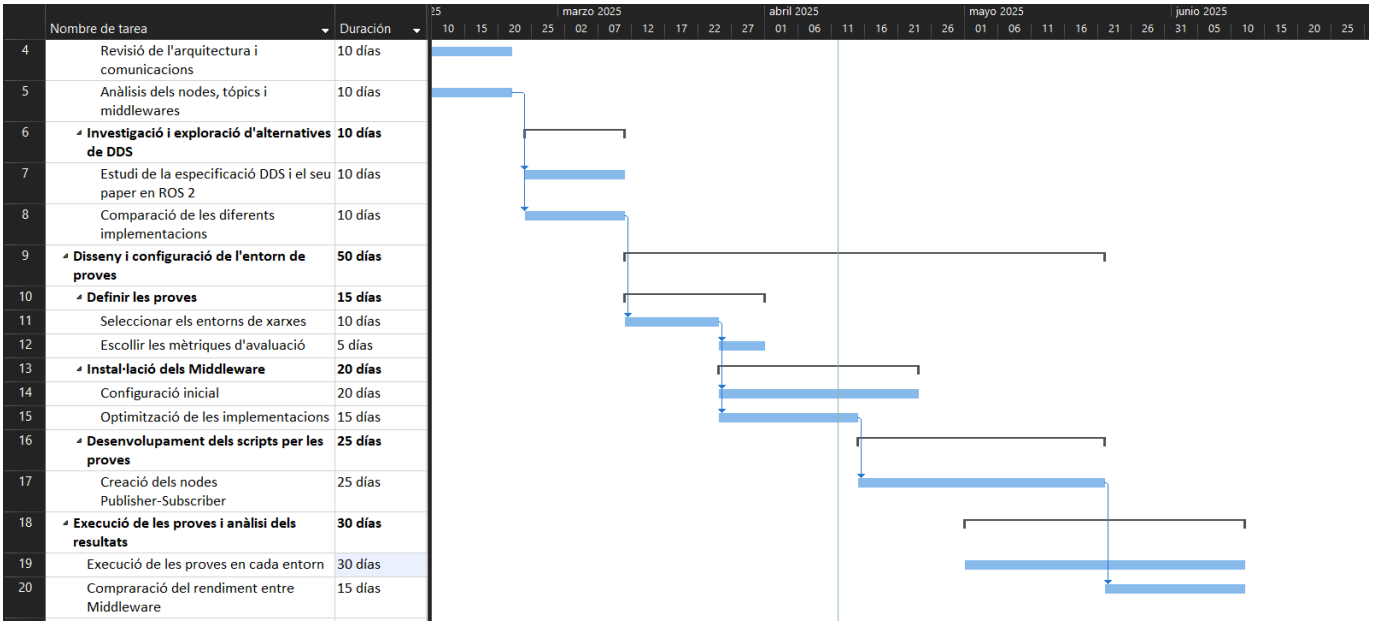
Vull expressar els meus més sincers agraïments al Bernat Gaston, tutor del TFG, per la seva apreciada orientació i consells en tot el procés i per haver estat tan implicat.

10 Bibliografia

[1] ROS 2 Documentation: Jazzy documentation https://docs.ros.org/en/jazzy/index.html
[2] Integrating ROS2 with Eclipse zenoh https://zenoh.io/blog/2021-04-28-ros2-integration/
[3] eProsima Fast DDS — ROS 2 Documentation https://docs.ros.org/en/rolling/Installation/DDS-Implementations/Working-with-eProsima-Fast-DDS.html
[4] eProsima Fast DDS Documentation https://fast-dds.docs.eprosima.com/en/latest/
[5] Eclipse Cyclone DDS — ROS 2 Documentation: Jazzy documentation https://docs.ros.org/en/jazzy/Installation/DDS-Implementations/Working-with-Eclipse-CycloneDDS.html
[6] Cómo comprobar el valor RSSI con NetSpot https://www.netspotapp.com/es/wifi-signal-strength/what-is-rssi-level.html
[7] Cómo comprobar la intensidad de la señal WiFi https://www.acrylicwifi.com/blog/como-comprobar-intensidad-senal-wifi/
[8] Cómo probar la intensidad de la señal WiFi en casa y mejorar la cobertura - Infobae https://www.infobae.com/tecno/2024/08/23/como-probar-la-intensidad-de-la-senal-wifi-en-casa-y-mejorar-la-cobertura/
[9] Creating an XML profiles file — Fast DDS 3.1.2 documentation https://fast-dds.docs.eprosima.com/en/latest/fastdds/xml_configuration/making_xml_profiles.html
[10] Configuration guide — Eclipse Cyclone DDS, 0.11.0 https://cyclonedds.io/docs/cyclonedds/latest/config/index.html
[11] Github Zenoh https://github.com/eclipse-zenoh/zenoh
[12] es/ROS/Introduccion - ROS Wiki https://wiki.ros.org/es/ROS/Introduccion
[13] ROS 2 Quality of Service policies https://design.ros2.org/articles/qos
[14] DDS Protocol Architecture Basics | RF Wireless World https://www.rfwireless-world.com/terminology/iot/dds-protocol-architecture-basics
[15] Zenoh https://zenoh.io/
[16] Protocols/rtps - Wireshark Wiki https://wiki.wireshark.org/Protocols/rtps
[17] Cyclone DDS | OMG DDS | ADLINK https://www.adlinktech.com/en/CycloneDDS
[18] Driving Autoware with Zenoh https://autoware.org/driving-autoware-with-zenoh
[19] ROS1 vs ROS2, Practical Overview For ROS Developers - The Robotics Back-End. https://roboticsbackend.com/ros1-vs-ros2-practical-overview/#:~:text=In%20ROS1%2C%20for%20Cpp%20you,ROS2%20has%20more%20layers
[20] Free VPN Accounts • 100% Free PPTP and OpenVPN Service https://www.vpnbook.com/freevpn

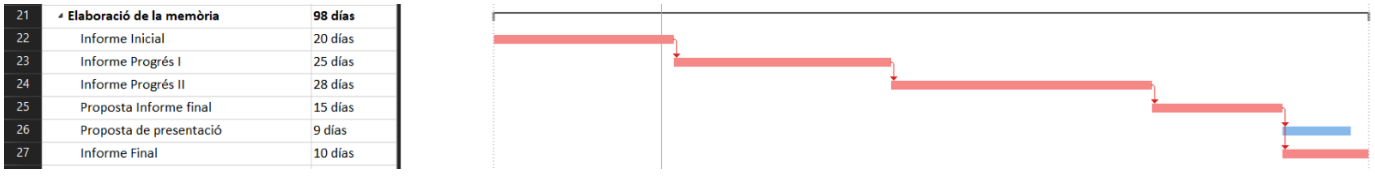
APÈNDIX

A1. Diagrama de Gantt



	Nombre de tarea	Duración
4	Revisió de l'arquitectura i comunicacions	10 días
5	Anàlisis dels nodes, tópics i middlewares	10 días
6	**Investigació i exploració d'alternatives de DDS**	**10 días**
7	Estudi de la especificació DDS i el seu paper en ROS 2	10 días
8	Comparació de les diferents implementacions	10 días
9	**Disseny i configuració de l'entorn de proves**	**50 días**
10	**Definir les proves**	**15 días**
11	Seleccionar els entorns de xarxes	10 días
12	Escollir les mètriques d'avaluació	5 días
13	**Instal·lació dels Middleware**	**20 días**
14	Configuració inicial	20 días
15	Optimització de les implementacions	15 días
16	**Desenvolupament dels scripts per les proves**	**25 días**
17	Creació dels nodes Publisher-Subscriber	25 días
18	**Execució de les proves i anàlisi dels resultats**	**30 días**
19	Execució de les proves en cada entorn	30 días
20	Compraració del rendiment entre Middleware	15 días

A2. Diagrama de Gantt Memòria



21	**Elaboració de la memòria**	**98 días**
22	Informe Inicial	20 días
23	Informe Progrés I	25 días
24	Informe Progrés II	28 días
25	Proposta Informe final	15 días
26	Proposta de presentació	9 días
27	Informe Final	10 días

A3. Configuració de la VPN

Per tal de poder realitzar la prova de la comunicació del Publisher/Subscriber a partir de la connexió d'una VPN, s'ha utilitzat un servidor VPN extern gratuït. L'opció escollida ha sigut VPNBook, ja que és fiable i senzilla d'instal·lar. Aquesta plataforma ofereix diferents servidors OpenVPN gratuïts arreu del món sense necessitat de registre.

Per tal de baixar la configuració del servidor s'ha d'anar a la web de VPNBook[7] escollir el .zip del servidor que millors s'adapti a nosaltres, en el nostre cas podria ser qualsevol servidor Europeu, un cop escollit s'hauria de descarregar i executar la següent comanda:

openvpn –config <fitxer>.ovpn

Ens demanaran un usuari i contrasenya que apareixen en la propia web i un cop introduïts ja tindrem la VPN creada.