



УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У
НОВОМ САДУ



Ратко Стиковић

**Софтверски адаптер између окружења
за федеративно учење PTV-FLA и
мидлвера DDS**

МАСТЕР РАД

Нови Сад, 2026



УНИВЕРЗИТЕТ У НОВОМ САДУ • ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
21000 НОВИ САД, Трг Доситеја Обрадовића 6

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, РБР :	
Идентификациони број, ИБР :	
Тип документације, ТД :	Монографска документација
Тип записа, ТЗ :	Текстуални штампани материјал
Врста рада, ВР :	Дипломски – мастер рад
Аутор, АУ :	Ратко Стиковић
Ментор, МН :	Проф. Др. Мирослав Поповић
Наслов рада, НР :	Софтверски адаптер између окружења за федеративно учење PTB-FLA и мидлвера DDS
Језик публикације, ЈП :	Српски / латиница
Језик извода, ЈИ :	Српски
Земља публикација, ЗП :	Република Србија
Уже географско подручје, УГП :	Војводина
Година, ГО :	2026
Издавач, ИЗ :	Ауторски репринт
Место и адреса, МА :	Нови Сад; Трг Доситеја Обрадовића 6
Физички опис рада, ФО : (поглавља/страна/ цитата/табела/слика/графика/прилога)	7/40/13/2/19/0/0
Научна област, НО :	Електротехника и рачунарство
Научна дисциплина, НД :	Рачунарска техника
Предметна одредница/Кључне речи, ПО :	Федеративно учење, PTB-FLA, DDS, RTSP
УДК	
Чува се, ЧУ :	У библиотеци Факултета техничких наука, Нови Сад
Важна напомена, ВН :	
Извод, ИЗ :	Овај рад нуди имплементацију транспортног адаптера за PTB-FLA окружење за развој алгоритама федеративног учења, заснованог на DDS протоколу. У овом раду је урађена и експериментална евалуација приложеног решења, његово поређење са претходном верзијом транспортног слоја, као и анализа и поређење перформанси Fast DDS имплементације DDS стандарда које нуди организација eProsima.
Датум прихватања теме, ДП :	
Датум одбране, ДО :	
Чланови комисије, КО :	Председник: Проф. Др. Јелена Радић
	Члан: Проф. Др. Илија Башичевић
	Члан, ментор: Проф. Др. Мирослав Поповић

Потпис ментора



KEY WORDS DOCUMENTATION

Accession number, ANO :	
Identification number, INO :	
Document type, DT :	Monographic publication
Type of record, TR :	Textual printed material
Contents code, CC :	Master Thesis
Author, AU :	Ratko Stiković
Mentor, MN :	Miroslav Popović, PhD
Title, TI :	A Software Adapter Between PTB-FLA Federated Learning Framework and DDS Middleware
Language of text, LT :	Serbian
Language of abstract, LA :	Serbian
Country of publication, CP :	Republic of Serbia
Locality of publication, LP :	Vojvodina
Publication year, PY :	2026
Publisher, PB :	Author's reprint
Publication place, PP :	Novi Sad, Trg Dositeja Obradovica 6
Physical description, PD : (chapters/pages/ref./tables/pictures/graphs/appendixes)	7/40/13/2/19/0/0
Scientific field, SF :	Electrical Engineering
Scientific discipline, SD :	Computer Engineering, Engineering of Computer Based Systems
Subject/Key words, S/KW :	Federated Learning, PTB-FLA, DDS, RTSP
UC	
Holding data, HD :	The Library of Faculty of Technical Sciences, Novi Sad, Serbia
Note, N :	
Abstract, AB :	This thesis presents implementation of DDS based transport adapter for PTB-FLA framework for development of federated learning applications. This thesis provides implementation and experimental evaluation of DDS based transport adapter, comparison between new adapter and old transport adapter used by PTB-FLA and comparison with performance of Fast DDS implementation of DDS standard provided by eProsima organization.
Accepted by the Scientific Board on, ASB :	
Defended on, DE :	
Defended Board, DB :	President: Jelena Radić, PhD
	Member: Ilija Bašičević, PhD
	Member, Mentor: Miroslav Popović, PhD
	Mentor's sign



KEY WORDS DOCUMENTATION

Студијски програм:	Рачунарство и аутоматика		
Студент:	Ратко Стиковић	Број индекса:	E2 30/2023
Степен и врста студија:	Мастер академске студије		
Област:	Електротехника и рачунарство		
Ментор:	Проф. Др. Мирослав Поповић		
НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА ИЗДАЈЕ СЕ ЗАДАТАК ЗА ЗАВРШНИ РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА: - проблем – тема рада; - начин решавања проблема и начин практичне провере резултата рада, ако је таква провера неопходна;			

НАСЛОВ ЗАВРШНОГ РАДА:

Софтверски адаптер између окружења за федеративно учење PTB-FLA и мидлвера DDS

ТЕКСТ ЗАДАТКА:

У оквиру овог мастер дипломског рада потребно је урадити следеће:

1. Упознати се са PTB-FLA и eProsim Fast DDS (скраћено DDS).
2. Израдити пројекат софтверског адаптера између PTB-FLA и DDS.
3. Имплементирати софтверски адаптер према израђеном пројекту.
4. Тестирати софтверски адаптер.
5. Еволуирати перформансу софтверског адаптера на: (1) једном рачунару и (2) локалној мрежи рачунара.
6. Уочити предности и мане имплементираног софтверског адаптера.
7. Написати текст мастер дипломског рада.
8. Поставити изворни код софтверског адаптера са упутством за инсталирање и коришћење на GitHub репозиторијум.

Руководилац студијског програма:	Ментор рада:

Примерак за: ☐ - Студента; ☐ - Ментора



UNIVERSITY OF NOVI SAD • FACULTY OF TECHNICAL SCIENCES
21000 NOVI SAD, Trg Dositeja Obradovića 6

KEY WORDS DOCUMENTATION

Изјављујем да нисам у сукобу интереса у односу ментор – кандидат и да нисам члан породице (супружник или ванбрачни партнер, родитељ или усвојитељ, дете или усвојеник), повезано лице (крвни сродник ментора/кандидата у правој линији, односно у побочној линији закључно са другим степеном сродства, као ни физичко лице које се према другим основама и околностима може оправдано сматрати интересно повезаним са ментором или кандидатом), односно да нисам зависан/на од ментора/кандидата, да не постоје околности које би могле да утичу на моју непристрасност, нити да стичем било какве користи или погодности за себе или друго лице било позитивним или негативним исходом, као и да немам приватни интерес који утиче, може да утиче или изгледа као да утиче на однос ментор-кандидат.

У Новом Саду, дана _____

Ментор

Кандидат

Захвалност

Највећу захвалност у изради овог рада, на стручној подршци и саветовању дугујем свом ментору професору Мирославу Поповићу.

Захваљујем се и свим пријатељима и колегама са којима сам много тога прошао у протеклих неколико година рада на факултету и у индустрији.

Посебну захвалност дугујем мојим родитељима, мајци Роси и оцу Љубиши, на несебичној љубави и пожртвовању. Све најбоље што носим са собом, почело је управо од њих.

Много ствари је донео период од мог дипломирања, знања, искустава и успомене. Искрено се радујем свим приликама за будуће напредовање и усавршавање, свим новим знањима и искуствима, као и чињеници да желим да их поделим са свима вама.

Захвалан сам на свему.

САДРЖАЈ

1. Увод.....	1
2. Теоријске основе.....	3
2.1 Федеративно учење.....	3
2.1.1 Изазови у системима федеративног учења.....	4
2.1.2 Топологија и класификација система федеративног учења.....	5
2.2 PTB-FLA окружење.....	6
2.2.1 MPT-FLA окружење.....	6
2.2.2 Архитектура PTB-FLA окружења.....	7
2.2.3 Топологија PTB-FLA мреже.....	8
2.2.4 Архитектура MPT-FLA окружења.....	9
2.2.5 Топологија MPT-FLA мреже.....	10
2.2.6 Транспортни слој MPAPI.....	11
2.3 DDS протокол.....	12
2.3.1 Појам ентитета и модел комуникације.....	12
2.3.2 Политике квалитета сервиса.....	15
2.3.3 Откривање учесника и RTSP протокол.....	15
2.4 Преглед сродних радова.....	16
2.4.1 Федеративно учење са освртом на мрежне перформансе.....	16
2.4.2 Перформансе DDS-а у односу на друге протоколе.....	17
3. Концепт решења.....	19
3.1 Преглед тренутног стања PTB-FLA транспортног слоја.....	19
3.2 Преглед примитива eProsimа имплементације DDS стандарда.....	20
3.3 Предлог решења транспортног адаптера.....	22

4.	Програмско решење.....	24
4.1	Дефиниција интерфејса	24
4.2	Имплементација адаптера	25
4.2.1	Имплементација класе уписивача.....	26
4.2.2	Имплементација класе читача.....	26
4.3	Интеграција адаптера у слој за комуникацију.....	27
4.3.1	DDS објекти у оквиру комуникационог API-а	27
5.	Експериментална евалуација	29
5.1	Методологија експеримената.....	29
5.2	Резултати мерења	31
5.3	Анализа и поређење резултата.....	33
6.	Закључак	37
7.	Литература.....	39

СПИСАК СЛИКА

Слика 1. Приказ централизованог федеративног учења	5
Слика 2. Приказ децентрализованог федеративног учења	5
Слика 3. UML дијаграм слојевите архитектуре PTB-FLA система	8
Слика 4. Приказ архитектуре PTB-FLA дистрибуиране апликације	9
Слика 5. Архитектура MPT-FLA окружења	10
Слика 6. Архитектура MPT-FLA дистрибуиране апликације	11
Слика 7. Приказ DDS домена.....	13
Слика 8. Објављивач.....	14
Слика 9. Претплатник.....	14
Слика 10. Генерисање Python модула на основу C++ библиотеке.....	21
Слика 11. Генерисање интерфејса на основу дефиниције	21
Слика 12. Поређење вредности брзина слања порука по интервалу	34
Слика 13. Поређење минималног кашњења по интервалу	35
Слика 14. Поређење просечног кашњења по интервалу	35
Слика 15. Поређење процентуалног кашњења на 50% узорака (медијана) по интервалу	35
Слика 16. Поређење процентуалног кашњења на 99% узорака по интервалу.....	36
Слика 17. Поређење максималног кашњења по интервалу	36

СПИСАК ТАБЕЛА

Табела 1. Резултати тестова спроведених над DDS адаптером.....	32
Табела 2. Резултати тестова спроведених над TCP адаптером	32

СКРАЋЕНИЦЕ

PTB-FLA – *Python Testbed for Federated Learning Algorithms*, окружење за развој алгоритама федеративног учења засновано на Python-у

MPT-FLA – *MicroPython Testbed for Federated Learning Algorithms*, окружење за развој алгоритама федеративног учења засновано на MicroPython-у

API – *Application Programming Interface*, интерфејс за развој апликација

FL – *Federated learning*, федеративно учење

OMG – *Object Management Group*, организација за стандаризовање објеката

DDS – *Data Distribution Service*, сервис за дистрибуцију података

RTPS – *Real Time Publish Subscribe protocol*, протокол за комуникацију у реалном времену заснован на механизму објављивања и претплате

QOS – *Quality of Service*, политика гаранције квалитета сервиса

TCP – *Transmission Control Protocol*, протокол за контролу слања

UDP – *User Datagram Protocol*, протокол корисничких датаграма

PDP – *Participant Discovery Protocol*, протокол за откривање учесника

EDP – *Endpoint Discovery Protocol*, протокол за откривање крајњих тачака

IDL – *Interface Definition Language*, протокол за дефиницију интерфејса

1. Увод

Премештање модела машинског учења са централизованих сервера ка уређајима на ивици мреже донело је са собом многе погодности, али и практичне проблеме. Приватни подаци, који су веома често осетљиве природе, остају на уређајима на ивици мреже. Тиме се делимично отежава процес обуке модела, јер модел и даље мора да се обучи над свим подацима, који се у овом случају налазе на различитим физичким уређајима. Федеративно учење (енгл. federated learning, FL) уводи решење проблема приватности и локализације података, тако што сервер не прикупља сирове податке од уређаја, већ уређаји шаљу само локално израчуната ажурирања модела, која потом сервер прикупља и агрегира у заједнички модел [1].

PTB-FLA (Python Testbed for Federated Learning Algorithms) је радно окружење настало као основа за развој апликација федеративног учења [2]. По свом пројекту PTB-FLA представља једноставно, али комплетно и целовито, окружење засновано на Python-у, без додатних спољних зависности, које нуди генерички оквир за развој централизованих и децентрализованих алгоритама федеративног учења. Та једноставност чини PTB-FLA погодним за окружења на ивици интернета за која већина других FL окружења није погодна. Окружење, дакле, добро решава алгоритамски део проблема какви су FL алгоритми и како их писати, као и распоређивање и оркестрацију тих алгоритама на стварну мрежу уређаја.

Увођење PTB-FLA окружења оставља отворено питање комуникације једног ивичног система, како чворови система на ивици међусобно комуницирају, какав је квалитет такве комуникације, да ли постоји и колико је кашњење у једном таквом систему. Овакви проблеми долазе до изражаја нарочито у окружењима попут индустријских постројења или мреже IoT уређаја, где комуникациона инфраструктура није ни изблиза толико једноставна и поуздана колико то претпостављају радна окружења коришћена у центрима података.

У оквиру овог мастер рада урађена је евалуација тренутног стања PTB-FLA окружења, преглед карактеристика и предности DDS и RTPS, пројектовање и имплементација транспортног слоја PTB-FLA окружења заснованог на поменутим протоколима, као и евалуација израђеног транспортног слоја у једном потпуном PTB-FLA систему. DDS, заједно са својим физичким протоколом, RTPS-ом [4] [5], управо даје решење за неке отворене проблеме система на ивици интернета: аутоматско, децентрализовано откривање учесника, конфигурабилне политике квалитета сервиса, комуникацију засновану на темама, као и обезбеђивање чврстих временских рокова, што систему отвара могућност за рад у реалном времену.

У следећем поглављу, *Теоријске основе*, сумирају се сви неопходни извори о PTB-FLA окружењу и његовом слоју за размену порука, DDS/RTPS стандарду, који су неопходни за разумевање овог рада, као и неке сродне радове на ову тему. Треће поглавље, *Концепт решења*, описује архитектуру и кључне одлуке са становишта пројектовања транспортног адаптера. Четврто поглавље, *Програмско решење*, приказује имплементацију адаптера, док пето поглавље *Експериментална евалуација* приказује резултате његове експерименталне евалуације у поређењу са оригиналним транспортом. Шесто поглавље, *Закључак*, даје осврт на цео рад као и преглед недостатака представљеног решења чиме се уједно и даје преглед неких будућих проширења решења. Поглавље седам *Литература*, наводи коришћене изворе.

2. Теоријске основе

У овом поглављу дат је преглед и разрада термина неопходних за разумевање овог рада као што су: федеративно учење, PTB-FLA окружење и његов транспортни слој, DDS и RTPS транспортни протоколи. На крају овог поглавља дат је и преглед сродних радова релевантних за ову тему.

2.1 Федеративно учење

Федеративно учење (енгл. *federated learning*, FL) је парадигма машинског учења у којој се модел обучава над подацима расподељеним на већи број учесника, односно уређаја, а при том приватни подаци (односно скуп података за обучавање) се никада не шаљу ван уређаја на коме се користе. За разлику од централизованог приступа машинског учења у коме се сви подаци прво прикупљају на једном месту (нпр. серверу или центру података) и тек онда користе за обуку модела, федеративно учење користи другачији приступ: модел се дистрибуира ка уређајима на којима се налазе подаци, а не подаци ка серверу који чува модел. На сервер који оркестрира моделом шаљу се искључиво резултати локалне обуке, односно ажурирани параметри модела или израчунати градијенти, које сервер потом агрегира у један, заједнички глобални модел [1].

Типичан процес федеративног учења одвија се кроз низ узастопних рунди, при чему свака рунда обухвата следеће кораке:

- 1) Иницијализација и дистрибуција модела – уколико сервер оркестратор постоји, шаље тренутну верзију глобалног модела свим учесницима или неком одабраном подскупу учесника.

- 2) Локална обука - сваки учесник над приватним, локалним подацима изводи одређени број корака или епоха обуке, полазећи од примљеног модела, чиме добија локално ажуриран модел.
- 3) Слање ажурираног модела - учесници шаљу натраг само разлику у односу на почетни модел односно ажуриране тежине и градијенте, а не локалне податке на основу којих је ажурирање израчунато.
- 4) Агрегација - сервер прикупља пристигле разлике ажурираних модела у односу на почетни модел и комбинује их у нову верзију глобалног модела, најчешће пондерисаним усредњавањем у односу на количину локалних података сваког учесника.

Овај циклус се понавља док модел не исконвергира до задовољавајуће тачности или док се не испуни неки други критеријум. Пошто се шаљу само ажурирања модела, а не сирови подаци, овакав приступ директно смањује ризик по приватност учесника и, у већини случајева, значајно смањује количину података коју је потребно пренети мрежом у поређењу са слањем целокупног скупа за обуку.

2.1.1 Изазови у системима федеративног учења

Практична примена федеративног учења носи низ изазова који се уобичајено групишу у четири категорије [10]:

- скупа комуникација - учесници по правилу шаљу ажурирања преко спорих, нестабилних или повремено недоступних мрежних веза, а сама мрежа федеративног учења може обухватати већи број учесника, тако да комуникација, поред осталих проблема, постаје критично уско грло система.
- системска хетерогеност - учесници се међусобно разликују по рачунарским, меморијским и комуникационим капацитетима, па систем мора да толерише учеснике који су спори, привремено недоступни или у потпуности нестану усред једне рунде обуке.
- статистичка хетерогеност - подаци код различитих учесника по правилу нису независно и идентично распоређени, што отежава и успорава конвергенцију модела у односу на класичну, централизовану обуку.
- приватност - иако размена искључиво ажурираних коефицијената модела, већ сама по себи смањује ризик по приватност, ни ажурирања нису потпуно безбедна, где злонамерни сервер или учесник може, техникама попут инверзије градијента, делимично реконструисати податке на основу којих је ажурирање израчунато, док злонамерни учесник слањем нарочито обликованог

ажурирања може покушати да у глобални модел угради скривено, нежељено понашање [11].

Прва два изазова комуникациони трошак и системска хетерогеност директно су везани за начин на који учесници у мрежи федеративног учења проналазе једни друге, размењују поруке и обезбеђују договорени ниво поузданости. То питање ретко је предмет самог алгоритма федеративног учења, већ се препушта транспортном слоју система за федеративно учење, што је управо домен проблематике коју овај рад обрађује.

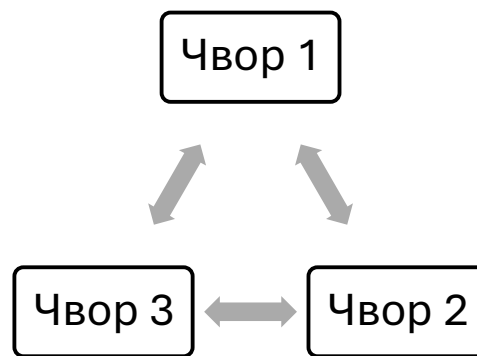
2.1.2 Топологија и класификација система федеративног учења

Федеративно учење није један јединствен приступ проблему децентрализованог обучавања модела вештачке интелигенције, већ породица сродних архитектура које се разликују по неколико независних критеријума.

Према топологији оркестрације разликују се централизовано и децентрализовано федеративно учење. У централизованом моделу постоји један оркестрациони сервер задужен за дистрибуцију модела и агрегацију ажурирања, док у децентрализованом моделу такав сервер не постоји, наиме учесници директно размењују ажурирања међу собом, а сваки учесник самостално врши агрегацију приспелих ажурирања у сопственом моделу [8]. Децентрализована оркестрација најчешће се остварује путем посебног протокола где сваки учесник ажурирања размењује само са неколико суседа, а информација се постепено шири кроз мрежу, прстенасте топологије или, ређе, механизма заснованих на блокчејн технологији [11]. Ова подела је директно релевантна за овај рад, будући да PTB-FLA окружење, представљено у наредном поглављу, нуди генеричке алгоритме за оба случаја.



Слика 1. Приказ централизованог федеративног учења



Слика 2. Приказ децентрализованог федеративног учења

Према расподели података разликују се хоризонтално и вертикално федеративно учење [9]. Код хоризонталног федеративног учења учесници деле исти скуп обележја

података, а разликују се по узорцима. Код вертикалног федеративног учења учесници деле исте узорке, али различита обележја.

Према обиму и природи учесника разликују се федеративно учење између уређаја и између центара података [8]. Федеративно учење између уређаја подразумева велики број, ресурсно ограничених уређаја (нпр. мобилни телефони, IoT сензори) који су повремено доступни и појединачно незнатно доприносе моделу. Федеративно учење између центара података подразумева мали број организација или центара података, сваки са знатно већим и поузданијим ресурсима, а самим тим и другачијим захтевима у погледу поузданости комуникације и толеранције на прекиде.

Према временском усклађивању агрегације федеративно учење може бити синхроно, где сервер чека да пристигну ажурирања унапред одређеног скупа учесника пре него што изврши агрегацију, или асинхроно, где се врши агрегација прерачунатих коефицијената чим пристигну, без чекања на остале учеснике, по цену веће сложености у обезбеђивању конвергенције модела [10].

2.2 PTB-FLA окружење

PTB-FLA (Python Testbed for Federated Learning Algorithms) је окружење за развој и извршавање алгоритама федеративног учења, писано у потпуности у програмском језику Python [2]. Део назива сугерише да је реч о тестном окружењу за проверу готових система федеративног учења, PTB-FLA је заправо окружење за федеративно учење, док testbed овде сугерише алгоритаМСКО окружење у коме се развијају и проверавају нови алгоритми федеративног учења, а не систем за тестирање готових решења [2]. У свом инкременталном процесу настанка, PTB-FLA окружење је проширивано до целовитог окружења за развој и валидацију алгоритама федеративног учења, међутим транспортни слој на који се PTB-FLA тренутно ослања је ограничен на извршавање свих инстанци апликације на једном рачунару, док проширење на локалну мрежу, као и евентуално извршавање у ивичним системима јесте предмет овог рада.

PTB-FLA зависи искључиво од стандардних Python пакета, пре свега пакета multiprocessing, без икаквих спољних зависности. Овакав начин имплементације окружења као предност има примарно једноставност, наиме окружење нема конкретних спољашњих зависности, што даље имплицира да уз измене транспортног слоја апликације писане у овом окружењу своју примену могу наћи и у IoT уређајима.

2.2.1 MPT-FLA окружење

MicroPython Testbed for Federated Learning Algorithms (MPT-FLA) је еволуција PTB-FLA радног оквира, настао као директан одговор на његово ограничење на један физички

рачунар [3]. Узрок тог ограничења може се наћи у самом избору транспортних примитива: PTB-FLA се ослања на Python-ов пакет multiprocessing (процес, ред, Слушаоца и Клијента), а MicroPython, минималистичка имплементација Python-а намењена микроконтролерима, не пружа еквивалентне примитиве, па PTB-FLA у изворном облику не може да се изврши на MicroPython платформи [3]. MPT-FLA то ограничење превазилази заменом транспортних примитива Python asyncio апстракцијама, чиме окружење постаје способно да ради непромењено и на регуларном Python-у (персонални рачунари, односно рачунари опште намене) и на MicroPython-у (нпр. Raspberry Pi Pico W плочице), уз прихватљиве перформансе на обе платформе [3].

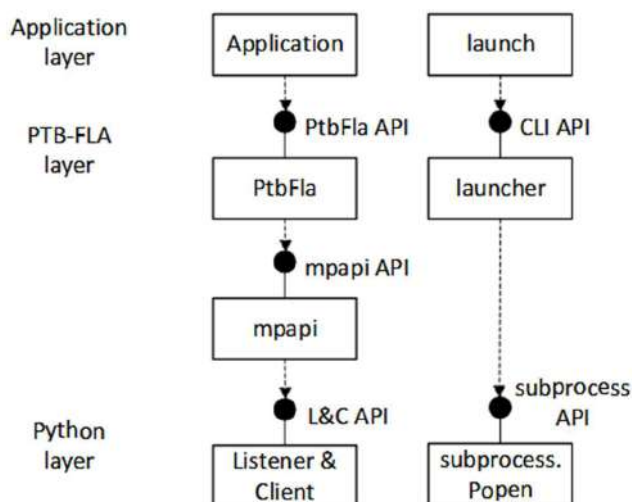
Експериментална евалуација MPT-FLA-а открила је, међутим, и практична ограничења транспорта заснованог на једноставним TCP везама преко бежичне мреже: поновљени покушаји повезивања, WiFi сметње изазивале су експоненцијално растуће кашњење, као и потпун прекида комуникације у изразито бучним окружењима. Поуздани резултати добијени су тек у готово идеалним лабораторијским условима [3]. Овакав налаз додатно поткрепљује мотивацију овог мастер рада, транспорт без стандардизованог механизма откривања учесника и без подесивог квалитета сервиса тешко подноси услове стварне ивичне мреже.

Главна предност MPT-FLA-а над PTB-FLA-ом је, дакле, способност да инстанце апликације изврши на различитим мрежним чворовима, укључујући и IoT уређаје, по цену напуштања стандардног CPython окружења и преласка на MicroPython и asyncio модел извршавања [3].

2.2.2 Архитектура PTB-FLA окружења

Архитектура PTB-FLA система организована је у три слоја [2]:

- Апликативни слој на врху, обухвата апликативне Python модуле за развој клијентских и серверских апликација и конзолни лансер, преко ког се систем покреће.
- PTB-FLA слој, обухвата класу PtbFla, као и модуле MPAPI (message passing API) слој за преношење порука, са својих пет функција (server_fun, sendMsg, rcvMsg, broadcastMsg, rcvMsgs), који је од посебног интереса за овај рад.
- Python слој, обухвата класе Process, Queue, Listener и Client из стандардног пакета multiprocessing.



Слика 3. UML дијаграм слојевите архитектуре PTB-FLA система
(Уз дозволу аутора преузето из референце [2])

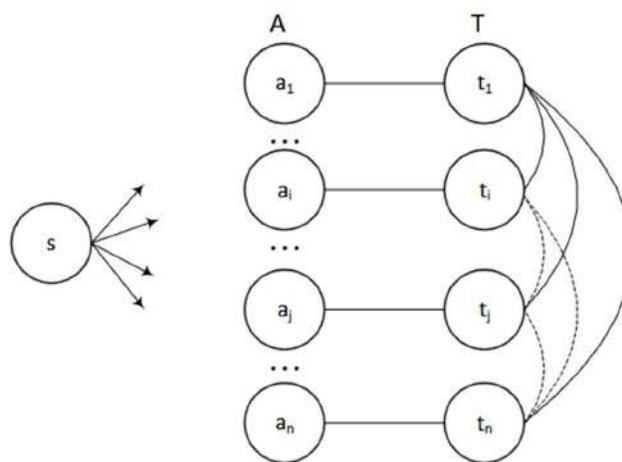
API класе PtbFla се састоји од четири функције [2], конструктора и деструктора, као и од две генеричке API функције које су посебно интересантне:

1. fl_centralized, извршава генерички централизовани алгоритам федеративног учења, преко повратно позивних функција сервера и клијента,
2. fl_decentralized, која извршава генерички децентрализовани алгоритам федеративног учења, такође преко повратно позивних функција.

Током развоја апликација федеративног учења употребом PTB-FLA окружења довољно је обезбедити повратно-позивне функције за улогу сервера и клијента, које генерички алгоритми, које PTB-FLA енкапсулира, позивају у одговарајућем тренутку извршавања.

2.2.3 Топологија PTB-FLA мреже

Систем се покреће тако што процес лансера инстанцира n инстанци апликације и покреће их као n независних процеса, где сваки процес чини пар инстанци: апликативну инстанцу и одговарајућу инстанцу окружења (testbed). Апликативни слој користи PtbFla API (конструктор/деструктор и функције fl_centralized и fl_decentralized) да прави тестбед инстанцу и учествује у извршавању дистрибуираног алгоритма, док PtbFla класа интерно, преко модула MPAPI, комуницира са осталим инстанцама.



Слика 4. Приказ архитектуре PTB-FLA дистрибуиране апликације
(Уз дозволу аутора преузето из референце [2])

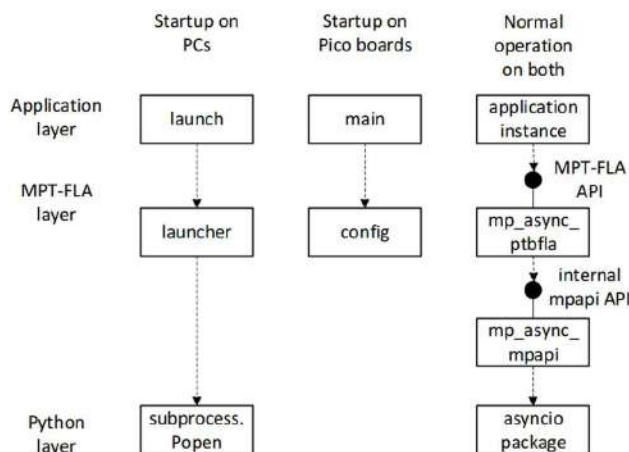
Генерички централизовани FL алгоритам извршава се над топологијом звезде, где сви чворови који учествују у комуникацији имају улоге клијената, док један оркестрациони чвор узима улогу сервера [2]. Алгоритам се извршава у три фазе, у првој фази сервер шаље свој локални модел свим клијентима, а сваки клијент позива своју повратно позивну функцију да добије ажурирани модел, у другој фази сервер прима ажуриране моделе од свих клијената, у трећој фази сервер позива своју повратно позивну функцију над примљеним подацима.

Генерички децентрализовани FL алгоритам извршава се над топологијом потпуно повезаног графа, пошто свака инстанца наизменично преузима улогу сервера и улогу клијента [2]. Алгоритам се извршава у три фазе, у првој фази свака инстанца, у улози сервера, шаље свој локални модел свим суседима, у другој фази свака инстанца, у улози клијента, прима поруке (могуће и ван редоследа, будући да се шаљу асинхроно) и позива своју повратно позивну функцију, у трећој фази свака инстанца поново игра улогу сервера и позива своју функцију повратног позива над примљеним ажурирањима.

2.2.4 Архитектура MPT-FLA окружења

Полазећи управо од PTB-FLA архитектуре, MPT-FLA у више развојних фаза долази до сопствене архитектуре, при чему задржава исти концептуални модел [3]. У својој коначној форми, MPT-FLA уводи Python `asyncio` апстракције: `asyncio` токови података користе се за комуникацију, корутине за асинхроно управљање процесима, уведено је блокирајуће чекање на `asyncio` догађаје [3].

Архитектура задржава исти, трослојни облик (слој апликације, MPT-FLA слој, Python слој), али Python слој, у случају MPT-FLA, чини `asyncio`, а MPT-FLA слој модули `mp_async_ptbfla` и `mp_async_MPAPI` [3].

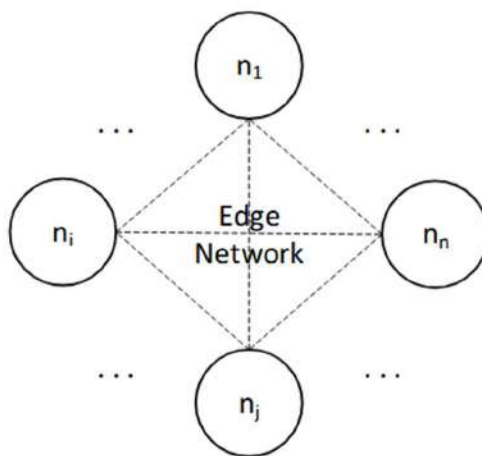


Слика 5. Архитектура MPT-FLA окружења
(Уз дозволу аутора преузето из референце [3])

API MPT-FLA окружења проширен је у односу на PTB-FLA API, на шест функција чланова, при чему су три функције сада асинхроне корутине. Додата је `start` корутина за учешће чвора у покретању система, `fl_centralized` и `fl_decentralized` задржавају исту улогу као у PTB-FLA, само су сада корутине, а нови члан `get1Meas` реализује TDM размену података са одређеним парњаком (детали у наредном одељку). Интерни MPAPI слој задржава истих пет функција PTB-FLA транспорта, сада реализованих као `'asyncio'` корутине [3].

2.2.5 Топологија MPT-FLA мреже

MPT-FLA систем представља се као граф чворова, при чему су чворови процеси, које рачунари, IoT уређаји или други мали системи, извршавају, а гране TCP везе, успостављене динамички по потреби и раскинуте након размене поруке, за разлику од PTB-FLA фиксног скупа веза унутар једног рачунара [3]. Окружењу је притом додат и трећи генерички алгоритам без парњака у PTB-FLA, TDM (Time Division Multiplexing) размена података међу парњацима, при чему топологија комуникације добија облик произвољног графа, за разлику од звезде (централизовани алгоритам) или потпуно повезаног графа (децентрализовани алгоритам) [3].



Слика 6. Архитектура MPT-FLA дистрибуиране апликације
(Уз дозволу аутора преузето из референце [3])

Генерички централизовани алгоритам извршава се над топологијом звезде, кроз m итерација, при чему свака итерација пролази кроз исте три фазе описане за PTB-FLA.

Генерички децентрализовани алгоритам извршава се над топологијом потпуно повезаног графа, такође кроз m итерација, при чему сваки чвор наизменично игра улогу сервера (фазе 1 и 3) и клијента (фаза 2) [3].

Трећи, TDM сценарио извршавања нема парњака у PTB-FLA. Чворови размену података организују кроз периодичан низ временских одсечака, при чему сваки чвор у датом одсечку комуницира са највише једним чвором, топологија комуникације је, у општем случају, произвољан граф [3].

2.2.6 Транспортни слој MPAPI

Функције `fl_centralized` и `fl_decentralized` за комуникацију са осталим инстанцама интерно користе модул MPAPI (Message Passing API), искључиво интерни API, намењен само класи `PtbFla`, који апликативни код никада не позива директно [2]. Модул MPAPI инстанцира Python multiprocessing класе `Listener` и `Client`, чиме остварује TCP комуникацију између сервера и клијента, уз важну напомену да се овде не ради о улогама сервера и клијента на апликативном нивоу (улога сервера и клијента између инстанци апликација), већ о интерним конструктима за пријем и слање порука [2].

На нивоу кода, MPAPI обезбеђује пет функција преко којих генерички алгоритми користе транспортни слој:

- `server_fun` покреће пријемну страну инстанце,
- `sendMsg` шаље поруку једном, конкретном одредишту,
- `rcvMsg` прима једну поруку,
- `broadcastMsg` шаље поруку свим осталим инстанцама,

- `rcvMsgs` прима више порука заредом.

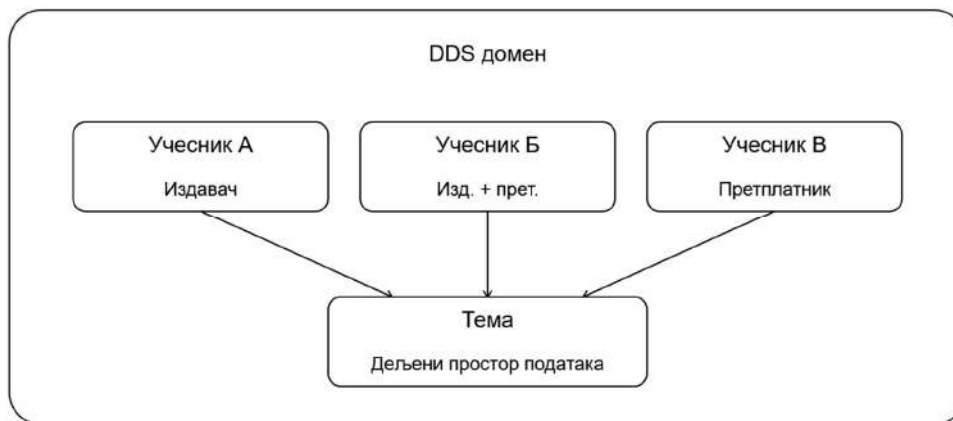
Пошто је овај слој изграђен непосредно над `multiprocessing` механизмима, комуникација међу инстанцама своди се на TCP везе адресиране паровима (`host`, `port`), што директно намеће и главно ограничење транспорта на који се ослања PTB-FLA: све инстанце морају да деле исти физички рачунар, да би међусобно успоставиле везу. Управо овај транспортни слој, без стандардизованог механизма откривања учесника или подесивог квалитета сервиса представља почетну тачку овог мастер рада.

2.3 DDS протокол

DDS (Data Distribution Service), односно сервис за дистрибуцију података, је стандард организације OMG за комуникацију засновану на подацима (`data-centric`), по моделу објављивања и претплате, усредсређеном на податке, а не на канале комуникације [4]. То практично значи да апликације не адресирају једна другу директно, нити међу собом отварају везе, оне објављују или се претплаћују на именоване, типизирани податке, а мидлвер преузима на себе да сваки објављен податак испоручи свим тренутно заинтересованим претплатницима, у складу са декларисаним захтевима квалитета сервиса. Овај стандард првенствено је намењен системима у којима велики број учесника у реалном времену размењује податке, без централног посредника. Своју примену налази у индустријским и роботским системима управо због комбинације аутоматског откривања учесника, стандардизоване семантике размене података и конфигурабилне гаранције квалитета сервиса (QoS).

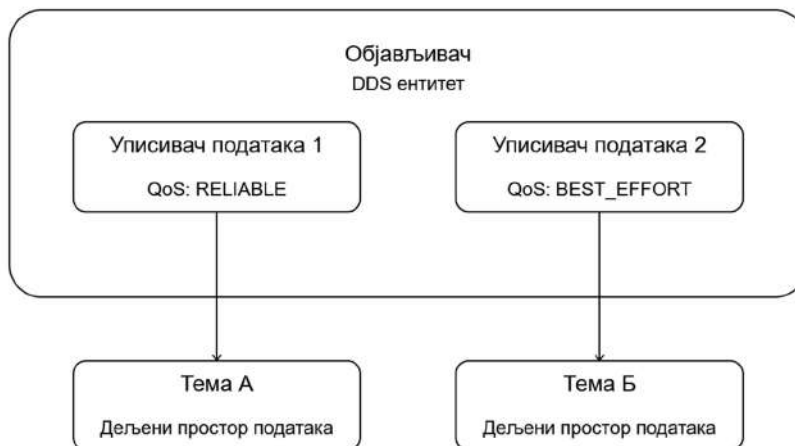
2.3.1 Појам ентитета и модел комуникације

DDS дефинише групу међусобно повезаних ентитета кроз које апликације приступају услугама мидлвера [4]. Доменски учесник представља улазну тачку у мидлвер и локално чланство апликације у именованом домену. Домен представља комуникациону равну која повезује све инстанце способне да међусобно комуницирају, само учесници пријављени у истом домену могу да размењују податке. Сви остали ентитети представљају неку конкретизацију ентитета доменског учесника.

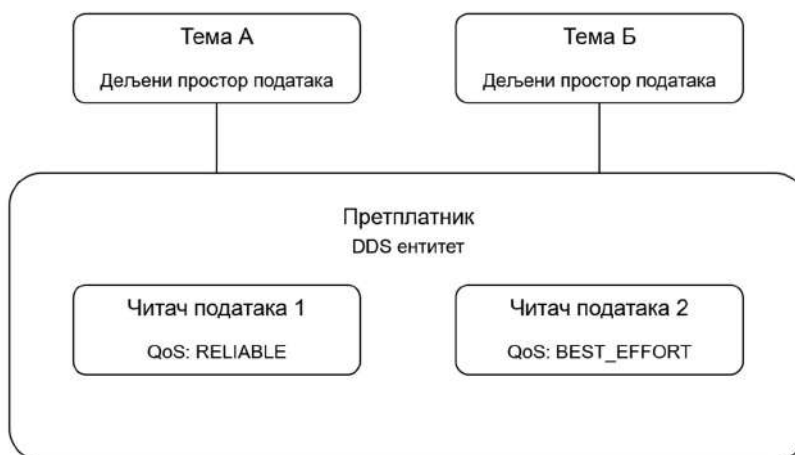


Слика 7. Приказ DDS домена

Комуникациони пут на страни пошиљаоца чине Објављивач (Publisher) и Уписивач (DataWriter). Објављивач је сам по себи генеричан (може се посматрати као интерфејс) може истовремено да дистрибуира податке различитих типова и одговоран је за саму дистрибуцију, у складу са важећим квалитетом сервиса. Апликација, међутим, никада не позива операције директно над Објављивачем, за сваки тип податка прави се посебан Уписивач, везан искључиво за тај тип (одатле и назив „типизиран приступ“), и управо над њим апликација позива операцију писања кад год жели да објави нову вредност. Објављивач тек тада, посредно, преузима на себе стварну дистрибуцију те вредности до претплатника. На страни примаоца, симетрично, налазе се Претплатник (Subscriber) и Читач података (DataReader). Претплатник је такође генеричан конструкт, који прима објављене податке и чини их доступним апликацији, док је Читач типизиран, односно везан за тачно један тип, преко кога апликација те податке чита [4].



Слика 8. Објављивач



Слика 9. Претплатник

Између ове две стране стоји Тема (Topic), јединствено именована у оквиру домена, типизиран канал податка, коме је придружен и сопствени скуп политика квалитета сервиса. Улога теме је управо у томе да раздвоји пошиљаоца од примаоца, где се подаци не адресирају директно од једног ка другом, него преко заједничке, именоване теме коју мидлвер препознаје и повезује одговарајуће уписиваче и читаче [4]. Уписивач и Читач над истом темом се упарују аутоматски, чим мидлвер утврди да су њихове политике квалитета

сервиса међусобно сагласне и све то без потребе да апликација икада сазна мрежну адресу супротне стране.

2.3.2 Политике квалитета сервиса

Сваки од наведених ентитета носи сопствени скуп политика квалитета сервиса (Quality of Service, QoS). Квалитет сервиса је за све ентитете конфигурабилан на нивоу ентитета, без потребе за директним управљањем политикама на апликативном нивоу [4]. Посебно су издвојене следеће политике:

- ПОУЗДАНОСТ, односно поузданост услуге, бира се између НАЈБОЉИ_ПОКУШАЈ (мидлвер не покушава да поново пошаље изгубљени узорак) и ПОУЗДАНО (мидлвер памти и, по потреби, поново шаље узорке све док прималац не реконструише потпун низ порука),
- ТРАЈНОСТ, односно трајност услуге, одређује да ли, и колико уназад, Читач који се придружи касније (након што су подаци већ објављени) добија приступ претходно објављеним вредностима,
- ИСТОРИЈА, односно чување претходних порука, одређује колико се претходних вредности сваке инстанце податка чува и доставља, САЧУВАЈ_ПОСЛЕДЊЕ чува само последњих n вредности, док САЧУВАЈ_СВЕ чува све узорке,
- КРАЈЊИ_РОК, односно граница крајњег рока, успоставља очекивани максимални период између узастопних ажурирања инстанце податка, мидлвер проверава сагласност понуђеног и захтеваног периода приликом упаривања писца и читаоца, а свако кашњење преко договореног рока пријављује се апликацији.

2.3.3 Откривање учесника и RTPS протокол

Откривање парова апликација које садрже ентитете који припадају истом домену у DDS-у је аутоматско и децентрализовано, учесници објављују своје присуство и своје теме на мрежи, а мидлвер сам повезује одговарајуће ентитете уписивача и читача, без потребе за фиксном адресом или унапред дефинисаним редоследом руковања [5]. Стандардом је дефинисано да се за потребе такве функционалности користи RTPS (Real-Time Publish-Subscribe) протокол, односно физички протокол, чија је сврха да апликације засноване на DDS имплементацијама различитих произвођача остваре међусобну интероперабилност.

RTPS откривање учесника у оквиру домена, као и њихово упаривање преко тема дели се на два независна протокола: протокол за откривање учесника (Participant Discovery Protocol) и протокол откривања крајњих тачака (Endpoint Discovery Protocol) [5]. Први

омогућава учесницима да објаве своју присутност на мрежи, а други, тек након што су два учесника открила један другог, омогућава учесницима да објаве своје карактеристике, односно које теме поседују, као и уписиваче и читаоце које сваки од њих поседује. Ради интероперабилности, свака RTPS имплементација мора да подржи бар основну варијанту сваког протокола: Simple Participant Discovery Protocol (SPDP) и Simple Endpoint Discovery Protocol (SEDP) [5].

SPDP ради на једноставном принципу где сваки учесник поседује ентитет уграђеног уписивача који периодично, унапред конфигурисаном скупу локатора (уникаст или мултикаст адреса), објављује податак о сопственом присуству и особинама, периодично понавља ту објаву и тако омогућава да учесници приступе мрежи било којим редоследом [5]. Симетрично, уграђени читалац прикупља такве објаве са мреже и тако сазнаје за присуство осталих учесника, укључујући и то које протоколе откривања крајњих тачака сваки од њих подржава.

Након откривања учесника путем SPDP-а, започиње извршавање SEDP, коришћењем сопственог, унапред дефинисаног скупа уграђених писаца и читалаца, учесници размењују податке о конкретним ентитетима читача и уписивача података и придруженим политикама квалитета сервиса које поседују [5]. Тек на основу ових података DDS имплементација локално утврђује да ли постоји сагласан пар уписивача и читаоца за сваку локалну тему и, ако постоји, успоставља упаривање описано у претходном одељку. Цео процес откривања и упаривања одвија се, дакле, аутоматски и децентрализовано, без централног регистра или унапред познатих адреса учесника.

RTPS размену порука типично преноси преко UDP/IP транспорта, мада спецификација не намеће овај транспорт као једини дозвољени [5].

2.4 Преглед сродних радова

У оквиру овог поглавља дат је преглед радова чија је тематика била од посебног значаја за израду овог рада. Издвојени радови обрађују различите теме, али се могу груписати у неколико логички одвојених група, па ће тако и бити раздвојени у наредним одељцима.

2.4.1 Федеративно учење са освртом на мрежне перформансе

У раду [12] урађена је анализа комуникационих проблема у парадигми федеративног учења, као и имплементација и евалуација референтног окружења заснованог на мрежи именованих података (Named Data Networking, у даљем тексту NDN). NDN комуницира преко имена садржаја, а не IP адреса уређаја, где сваки NDN чвор одржава локални бафер

и табелу захтева. Пошто је NDN изворно заснован на механизму повлачења података (слање података се ради тек након примљеног захтева), аутори морају да га прошире механизмом нотификација интереса (енгл. interest notifications) да би омогућили комуникацију засновану на гурању података, каква је потребна за федеративно учење. Резултати показују 23% временског побољшања у обуци модела у односу на TCP/IP при минималној маргини грешке. Трајање рунде обучавања је редуковано са 1600 секунди за TCP/IP, на 150 секунди за NDN. Саобраћај је редукован за до 74% због опција баферовања захтева, док је број скокова од 4 за TCP/IP редукован на 2.4 у просеку. Посебно занимљиво код овог рада је тврдња да решења која су оријентисана према домаћину, базирана на адресама и оријентисана према успостави везе (какав је TCP/IP), по структури и перформансу заостају за решењима која су оријентисана према подацима и користе механизме именованих података (имена или теме) са уграђеним кашњењем и мултикастом. Те разлике су у неким случајевима цео ред величине. Наведена запажања иду у прилог увођења DDS-а као мидлвера транспортног адаптера PTB-FLA.

Рад [11] приказује резултате експерименталне евалуације алатом "FedComm", јавно доступан, изграђен на бази FedAdapt имплементације окружења федеративног учења. Тестирање је обухватило пет протокола: TCP, UDP, MQTT, AMQP и ZMQ. Мрежни услови симулирани преко ограничења протока података (3G=5Mbps, 4G=20Mbps, WiFi=60Mbps), стрес мреже (~250Mbit/s) и степен губитка пакета (tc: 0/5/10/20%). Овај рад даје резултате који иду у прилог протокола који користе механизам рачвања, где се подаци директно дистрибуирају до свих уређаја, уместо TCP модела слања N засебних порука. Постојање брокера уводи додатан трошак у систем по питању процесорског времена, због потребе да уређаји константно слушају брокера. Наведени резултати могу бити добар аргумент за увођење DDS у систем федеративног учења, с обзиром да подржава један ка више дистрибуцију података, али не користи засебног брокера, па у том контексту се не би правио трошак заузимања додатног процесорског времена.

2.4.2 Перформансе DDS-а у односу на друге протоколе

У раду [10] урађен је преглед литературе о експерименталној евалуацији перформанси DDS имплементација, као и DDS перформансама (кашњење, количина података у времену) у поређењу са другим технологијама и међу DDS имплементацијама. Резултати овог рада указују да DDS генерално надмашује HLA, MQTT, RabbitMQ и AMQP по throughput-у и jitter-у, али га надмашују традиционална socket комуникација и ZeroMQ, посебно за поруке мање од 1KB. Перформансе по величини поруке OpenSplice и OpenDDS боље се показују за мале податке, док RTI DDS и FastRTPS/FastDDS боље скалирају за веће дужине података

(>10KB до 1MB). Иако је у току истраживања рађено поређење различитих имплементација DDS-а, различите политике квалитета сервиса (трајност, крајњи рок, буџет кашњења и историја) нису тестиране у контексту перформанси. Рад је показао да DDS већим делом надмашује остале протоколе на принципу објављивања и претплате и то по различитим критеријумима, као и да даје велики потенцијал за скалабилност система.

3. Концепт решења

У овом поглављу дат је преглед тренутног API-а транспортног слоја PTB-FLA, преглед DDS примитива дефинисаних у eProsima имплементацији DDS стандарда, као и предлог решења заменских API функција.

3.1 Преглед тренутног стања PTB-FLA транспортног слоја

Транспортни слој PTB-FLA окружења, модул MPAPI, на нивоу кода изложен је кроз пет функција, представљених у Поглављу 2, чији тренутни изглед је потребно детаљно размотрити пре предлагања замене.

`server_fun(localPort, queue)` покреће пријемну страну транспортног адаптера инстанце, прављење Слушаоца, који је везан за адресни пар ("localhost", localPort) и улазак у бесконачну петљу чиме започињу рутину прихватања долазећих захтева. За сваки прихваћен захтев, функција прима поруку, врши десеријализацију из JSON формата записа, и уписује је у дељени ред, дат као queue параметар `server_fun` функције. Значајно је да је адреса чвора у општем случају постављена на "localhost", што директно потврђује ограничење транспорта који је у употреби у PTB-FLA окружењу на један физички рачунар. Овај проблем се може заобићи директним уносом физичке адресе чвора као параметра покренуте инстанце, али то би захтевало додатне измене на апликативном нивоу, што захтева потпуно познавање топологије физичке мреже односно IP адресе чворова, као што и у потпуности одступа од установљеног правила за развој транспортног адаптера за PTB-FLA, апликативни ниво не сме да познаје било какве промене везане за адаптер, API мора бити једнообразан без обзира на вид транспорта.

`sendMsg(remoteServerAddress, msg)` ствара Клијента ка датој адреси, серијализује поруку у JSON формат и шаље је. Уколико веза није могућа (нпр. одредишни Слушалац још

увек не прихвата везе), функција понавља покушај успостављања везе до програмски дефинисане границе (`MAX_RETRY_COUNT = 100`), уз фиксну паузу од 200ms између покушаја. Овако дефинисана програмска рутина, са стриктним алгоритмом поновних покушаја успостављања везе, без икаквог механизма за откривање учесника или конфигурабилног рока за слање порука, представља проблем за који DDS стандард нуди решења, уграђени механизам откривања и подесива политика квалитета сервиса са учинком на степен поузданости представљају алтернативу.

`rcvMsg(queue)` позива блокирајућу методу из Python-а за добављање узорка из реда, где је целокупна логика пријема порука премештена у `server_fun` API функцију, преко истог дељеног објекта реда. Параметар `queue` се функцији `rcvMsg` прослеђује преко `server_fun` функције, којој је исти `queue` објекат претходно прослеђен.

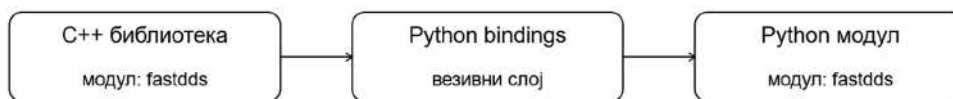
`broadcastMsg(listOfRemoteServerAddress, msg)` бродкаст механизам је овде остварен петљом која позива `sendMsg` за сваку адресу са листе дате као први параметар функције појединачно. Другим речима, у RTB-FLA транспортном слоју `broadcast` семантички представља низ узастопних `unicast` позива, а не једну операцију на нивоу транспорта. Овде, такође, DDS стандардом дефинисани `broadcast` смањује оптерећење слања N засебних унікаст порука, слањем порука на дељену тему за све учеснике у комуникацији.

`rcvMsgs(queue, noOfMessagesToReceive)` позива `rcvMsg` API функцију онолико пута колико то други параметар функције дефинише и примљене поруке се смештају у листу коју потом функција враћа преко повратне вредности.

Заједнички недостатак свих API функција је одсуство било каквог механизма откривања учесника (адресе морају бити унапред познате и прослеђене као параметри), одсуство конфигурабилног квалитета сервиса, као и текстуална JSON серијализација порука без формализованог типа порука. Управо те наведене тачке: адресирање, откривање, поузданост и типизираност порука представљају конкретне тачке на којима DDS примитиве, представљене у наредном одељку, замењују постојеће решење.

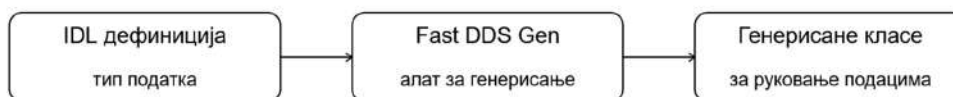
3.2 Преглед примитива eProsimа имплементације DDS стандарда

eProsimа Fast DDS је бесплатна и јавно доступна C++ имплементација DDS/RTPS стандарда. Приступ свим функционалностима дефинисаним у имплементацији остварује се преко библиотеке `fastdds`. eProsimа имплементација, примарно нуди C++ библиотеку, међутим употребом Python везивног слоја (Python bindings) омогућен је директан приступ свим функцијама, класама и ентитетима описаним у Поглављу 2.3 овог рада, кроз модул `fastdds`.



Слика 10. Генерисање Python модула на основу C++ библиотеке

Тип података који се преноси преко теме дефинише се у језику за дефинисање интерфејса (Interface Definition Language (IDL)), из дефинисаног интерфејса се алатом Fast DDS Gen генеришу се класе које представљају апстракције неопходне за руковање задатим интерфејсом.



Слика 11. Генерисање интерфејса на основу дефиниције

Успостављање учесника и теме, готово је идентично на страни уписивача и на страни читаоца. Преко објекта шаблона фабрике доменског учесника прави се доменски учесник над неким доменом, затим се над њим региструје одређени тип (тип дефинисан преко језика за дефиницију интерфејса), а потом се преко доменског учесника прави тема са задатим именом. Тек након тога свака страна прави сопствени ентитет вишег нивоа, уписивач прави Објављивача, а читалац Претплатника, обома ентитетима придружују се политике квалитета сервиса преко подразумеваних политика квалитета сервиса добијених од основног доменског учесника (`get_default_publisher_qos` и `get_default_subscriber_qos` поља базне класе).

Класа Уписивач енкапсулира страну уписивача тако што над Објављивачем прави Уписивач података над темом, чија се политика квалитета експлицитно конфигурише подразумеваним вредностима. Уз Уписивача података се везује и Слушалац, чији `on_publication_matched` повратни позив, при сваком упаривању или раскиду упаривања са читаоцем, ажурира бројач упарених читалаца. Метода `wait_discovery` блокира позиваоца све док бројач не постане различит од нуле, дајући уписивачу експлицитан начин да сачека да

га бар један читалац открије пре слања прве поруке. Сама операција слања, метода уписа, пуни инстанцу структуре која наслеђује интерфејс пољима дефинисаним у IDL-а и позива методу за упис над њом.

Класа Читача, симетрично, енкапсулира страну читаоца. Над претплатником се прави Читалац података над истом темом, са истом комбинацијом политика квалитета као у парњаку Објављивача. Уз њега се везује Слушалац, чији `on_data_available` повратни позив, позван кад год мидлвер испоручи нов узорак, узима податак операцијом за добављање, проверава да ли је реч о важећем податку. Пријем порука је, дакле, потпуно вођен догађајима, за разлику од блокирајућег у оригиналном MPAPI транспортном адаптеру. Из виших нивоа се не позива ништа од наведених функција повратног позива да би се примила порука, мидлвер сам позива `on_data_available` чим порука стигне.

3.3 Предлог решења транспортног адаптера

Централни циљ предложеног решења је потпуна замена транспортног слоја описаног у одељку 3.1 DDS примитивама из одељка 3.2, а да при том пет API функција транспортног модула задрже исте потписе и исто значење за позиваоца. На тај начин, PTB-FLA API и генерички алгоритми који су потпуно агностични према транспортном протоколу који транспортни слој енкапсулира, не морају да уведу промене свог тренутног изворног кода како би се нови транспорт применио, довољно је да се заменски транспортни модул импортује уместо оригиналног MPAPI модула.

Испод јавног API-а, решење треба да уведе два објекта који морају постојати тачно једном по процесу: Слушалац порука, који поседује све објекте читалаца, и Уписивач порука, који поседује све објекте уписивача. Како би се обезбедила јединственост оба објекта наведених класа, треба обезбедити заједничку класу дизајн обрасца `singleton`, чиме се осигурава да ће се из различитих делова апликације, увек руковати истим DDS ентитетима и да се неће стварати нови при сваком позиву.

Слушалац порука при прављењу прима дељени ред и локални порт чвора, и на основу њих прави два Читача, један претплаћен на тему под именом Броудкаст на који се сви чворови претплаћују, односно тему заједничку за све чворове, као и један претплаћен на тему учесник/{локални_порт}, наменски резервисану за поруке упућене баш том чвору. Оба читаоца конструисана су над истим параметром реда, тако да порука примљена на било којој од две теме мора бити уписана у један јединствени дељени ред, односно идентичан ток података којим су генерички алгоритми и до сада очекивали да добију поруке.

Уписивач порука треба да поседује информације о свим уписивачима који су придружени сваком учеснику, уписивачи морају бити идентификовани по идентификатору

чвора на који се односе, уз то потребан је и један заједнички писац за Броудкаст тему. За сваки чвор који је у интересу за размену података са одређеним чвором, потребан је једнократан корак регистрације где се за дати идентификатор чвора прави одговарајући Уписивач над наменском темом учесник/{порт}, тек након тога `sendMsg` може да пронађе Уписивача за тог учесника.

Имплементације свих пет API функција транспортног слоја свODE се на адекватно коришћење наведених објеката:

- `server_fun(local_port, queue)` треба да прави синглтон Слушаоца порука (чиме се покрећу оба Читача), након чега само остаје активна, будући да се сада пријем порука обавља асинхроно, у позадини,

- `sendMsg(remote_server_address, msg)` треба да пронађе регистрованог Уписивача за дату адресу и да позове над њим методу за упис, тако да остатак PTB-FLA окружења не мора да зна да се испод API-ја налази другачији транспортни механизам,

- `recvMsg(queue)` остаје непромењена, где и даље само чита из дељеног реда, при чему сада поруке стижу и убацују се у ред асинхроно,

- `broadcastMsg(msg)` треба да обави позив методе за упис над јединим, дељеним уписивачем броудкаст теме,

- `recvMsgs(queue, no_of_messages_to_receive)` остаје потпуно непромењена.

Овако постигнута заменљивост није, међутим, безусловна: захтева поменути једнократни корак регистрације Уписивача по учеснику, који оригинални транспорт није захтевао, будући да DDS, за разлику од директног отварања везе на захтев, унапред тражи да писац и одговарајућа тема постоје пре него што порука уопште може бити послата.

4. Програмско решење

У овом поглављу дат је опис програмске имплементације транспортног адаптера, преглед свих помоћних класа и функција, као и њихов начин употребе на нивоу API функција. Цео изворни код је јавно доступан на репозиторијуму [13].

4.1 Дефиниција интерфејса

Полазна тачка имплементације транспортног адаптера је IDL дефиниција интерфејса типа податка који ће се преносити преко DDS теме. Дефинисани интерфејс, у фајлу `Adapter.idl`, представљен је структуром:

Дефиниција интерфејса типа Адаптер

```
01: struct Adapter
02: {
03:     string message;
04: };
```

Поље Порука носи стварни садржај поруке. На основу овакве дефиниције интерфејса, алат `fastddsgen` не генерише директно Python код, него C++ заглавље Адаптер са дефиницијом C++ класе Адаптер, као и `AdapterPubSubTypes` са екстензијама `.hpp` и `.cxx` и `AdapterTypeObjectSupport` `.hpp` и `.cxx`, у којима је имплементирана CDR серијализација и десеријализација овог типа, односно механизам преко кога се инстанца класе Адаптер припрема за преношење преко RTPS-а. Овај C++ код се, заједно са SWIG интерфејсним записом, помоћу CMake-а компилира у две дељене библиотеке: `libAdapter.so`, која садржи саме C++ типове повезане са `fastcdr` и `fastdds` библиотекама, и `_AdapterWrapper.so`, SWIG-ом генерисано Python проширење (C екстензија) које омогућава да C++ типови могу да се позивају из Python-а. Тек над овим проширењем стоји танак, такође аутоматски генерисан Python модул који излаже класе Адаптер и Типови_објаве_и_претплате_Адаптера. Поља

класе Адаптер нису изложена као обични атрибути, него као методе које обједињују читање и упис. Класа Типови_објаве_и_претплате_Адаптера имплементира интерфејс Тип_податка_теме (са методама попут серијализуј, десеријализуј, припреми_податке) и управо њену инстанцу је потребно регистровати над учесником (методом за регистрацију, описану у 3.2) пре него што тема заснована на овом типу може бити направљена.

Разлог зашто је интерфејс сведен на једно опште поље Порука, уместо да прецизно моделује облике порука какве генерички алгоритми PTB-FLA окружења стварно размењују (један податак код централизованог алгоритма; итерација, фаза, адреса извора и податак код децентрализованог, како је описано у одељку 2.2.3), лежи у самом дизајну оригиналног транспорта: МРАPI слој не познаје експлицитно облик поруке коју преноси, тумачење садржаја поруке искључиво тумаче генерички алгоритми. Транспортни адаптер задржава исти принцип, где облик поруке остаје одговорност на страни Python слоја, `sendMsg` и `recvMsg`, како је описано у одељку 3.3, серијализација и десеријализација JSON записа пре него што порука уђе у `message` поље, тако да један, непромењен DDS тип подржава све облике порука PTB-FLA окружења, без потребе да IDL интерфејс прати сваку будућу измену облика поруке на апликативном нивоу.

4.2 Имплементација адаптера

На основу генерисаног интерфејса, као и Python екстензија C++ библиотека, извршена је енкапсулација интерфејса, на тај начин да дефинисани модули имају тачно одређену улогу коју обављају (концепт из објектно-оријентисане парадигме правило јединствене одговорности). Пакет `adapter`, као и модули `AdapterPublisher.py` и `AdapterSubscriber.py` које обухвата, представља ниво имплементације на коме су DDS примитиве, описани у одељку 3.2, енкапсулирају у интерфејс погодан за директну употребу у транспортном слоју описаном у одељку 3.3. Главна замисао је раздвајање одговорности: цела логика везана непосредно за DDS, односно прављење учесника, регистрација типа, теме, QoS политике, механизми слушаца, остаје затворена унутар две класе, Уписивач и Читалац, тако да остатак адаптера, модул `msg_passing_api`, са њима комуницира искључиво преко неколико метода уског обима (конструктор, метода за уписивање, приступ дељеном реду), без потребе да икада директно позове `fastdds API`. Оваква подела директно служи циљу целог адаптера, постављеном у Поглављу 3, дефинисању понашања интерфејса без измена на страни PTB-FLA окружења. Јавни интерфејс транспортног слоја мора да остане непромењен, имплементација специфична за DDS мора да буде енкапсулирана, а не једноставно премештена на неки виши ниво, управо ту улогу играју Уписивач и Читач. Оваквим приступом остварује се и неколико предности. Пре свега, модул `msg_passing_api`,

а самим тим ни PTB-FLA, као ни апликативни код изнад њега, уопште не користе нити познају fastdds библиотеку. Читава зависност од DDS-а сведена је на ове две класе, чиме употпуњује апстракција транспорта. Друго, будућа измена начина на који се DDS користи, прилагођавање другачијим QoS политикама, промена верзије eProsima имплементације, остају локализовани искључиво на класама Уписивача и Читача, без последица по остатак адаптера или на PTB-FLA окружења изнад њега. Треће, будући да класе Уписивача и Читача изван конструктора нуде по једну операцију (за писање, односно пријем преко дељеног реда), виши слој према сваком учеснику, било да је реч о дељеној теми или наменској теми појединачног учесника, поступа на потпуно исти начин.

4.2.1 Имплементација класе уписивача

Конструктор класе Уписивач, дефинисане у AdapterPublisher.py, прима име типа и назив теме и над њима спроводи читав низ корака описан у одељку 3.2: прави доменског учесника, региструје тип AdapterPubSubType, прави тему, Објављивача, а потом и Уписивача података, чија се политика квалитета ПОУЗДАНОСТ експлицитно поставља на ПОУЗДАНО и ИСТОРИЈА на САЧУВАЈ_СВЕ.

Уз уписивача се везује и Слушалац уписивача, посебна класа, чија функција повратног позив упарена_објава при сваком упаривању или раскиду упаривања са читаоцем увећава или умањује бројач упарених читалаца.

Метода сачекај_откривање користи условну променљиву да блокира позиваоца све док бројач упарених читалаца не постане различит од нуле.

Метода за упис је једина операција коју остатак адаптера позива у нормалном раду: прави инстанцу структуре Адаптер, поставља јој поље Порука, и над њиме позива метод за упис.

4.2.2 Имплементација класе читача

Класа Читача, дефинисана у AdapterSubscriber.py, симетрична је класи Уписивача, где конструктор пролази исти низ корака: прављење доменског учесника, регистрација типа, тема, прављење Претплатника и Читача података, са истом ПОУЗДАНО и САЧУВАЈ_СВЕ QoS комбинацијом. Конструктор Читача прима и опциони параметар ред, дељени ред у који ће примљене поруке бити уписиване, и прослеђује га Слушаоцу.

Уз читаоца се везује Слушалац читаоца.

Метода повратног позива податак_је_доступан, позван кад год мидлвер испоручи нов узорак, преузима податак операцијом преузми_следећи_узорак, проверава стање валидности и, ако је узорак важећи, само поље Порука уписује у дељени ред.

4.3 Интеграција адаптера у слој за комуникацију

Модул за преношење порука `msg_passing_api.py` представља тачку у којој се класе Уписивача и Читача, повезују у конкретну имплементацију пет API функција транспортног слоја. Управо овај модул апликативни слој користи уместо оригиналног MPAPI модула да би остварио прелазак на DDS транспорт.

Јединственост објеката Читач и Слушалац порука обезбеђује заједничка метакласа обрасца синглтон Комуникациони синглтон.

Свака класа која за своју базну класу наведе класу Комуникациони синглтон добија понашање синглтона: први позив конструктора, прави инстанцу и памти је у речнику инстанци, док сваки наредни позив само враћа већ упамћену инстанцу, а сам конструктор се уопште поново не извршава.

Класа Слушалац порука користи базну класу Комуникационог синглтона да при првом позиву направи два читаоца, једног на дељеној теми Броудкаст, други на теми учесник/локални_порт, односно наменској за тај чвор. Оба слушаоца порука се праве над истим дељеним редом.

Класа Уписивача порука, која такође наслеђује класу Комуникацион синглтон, при сопственом настајању прави дељеног уписивача за броудкаст тему и одмах позива његову покрену методу, чиме конструисање овог објекта блокира док се бар један читалац не упари на броудкаст тему. Метода додај_уписивача је метода која за дати идентификатор чвора прави наменског уписивача над темом учесник/{број_порта} и уписује га у речник учесника.

4.3.1 DDS објекти у оквиру комуникационог API-а

У овом одељку дат је опис комуникационих функција дефинисаних у оквиру интерфејса за пренос порука, али измењених тако да користе DDS дефинисане објекте за комуникацију са мидлвером. Како је описано у претходним одељцима API транспортног слоја је већ дефинисан и већ је у употреби и као такав не сме да се мења.

API метода `server_fun` ствара (или, ако већ постоји у процесу, само преузима) објекат класе Слушаоца порука, чиме се покрећу оба читаоца описана изнад, један општи за броудкаст тему, и један локални само намењен том учеснику у комуникацији, а потом остаје у петљи мировања, сав стварни пријем порука одвија се асинхроно, преко методе повратног позива `подаци_су_доступни`.

API метода `sendMsg` тражи уписивача регистрованог за дату адресу у речнику учесника који је поље синглтона Уписивача порука, и над њим позива методу за упис, при чему поруку претходно серијализује у JSON запис.

API метода `rcvMsg` симетрично примењује десеријализацију JSON записа над оним што прочита из дељеног реда, чиме се затвара круг серијализације отворен у `sendMsg`.

API метода `broadcastMsg` више не користи свој први параметар листу адреса свих учесника, која је задржана искључиво ради сагласности потписа са оригиналним транспортним слојем. У овој имплементацији се једноставно позива метод за упис над дељеним броудкаст Објављивачем.

API метода `rcvMsgs` остаје непромењена, где се и даље само позива `rcvMsg` задати број пута.

5. Експериментална евалуација

У овом поглављу описана је експериментална евалуација транспортног адаптера имплементираних у Поглављу 4. У наредним одељцима биће објашњена методологија мерења и тестно окружење, на који начин су спроведени тестови, поред тога дати су и експериментални резултати и урађено је поређење са оригиналним, МРАPI транспортом PTB-FLA окружења.

5.1 Методологија експеримената

Мерења која су спроведена за потребе овог рада настала су у оквиру самосталног тест окружења које транспорт покреће искључиво преко пет API функција, описаних у одељку 4.3. на исти начин како би свака од њих била позвана у оквиру PTB-FLA окружења. Резултати добијени на тај начин одражавају стварни трошак који апликација плаћа, а не само идеализовани трошак DDS примитива или оригиналног транспорта. Поред тога на овај начин оба вида транспорта су тестирана на истом нивоу, односно на нивоу транспортног API-а, а не сваки на свом нивоу апстракције. Једнообразност остварена на тај начин омогућава да поређење резултата буде равноправно за оба транспорта. Разлике измерене између два транспорта потичу из начина на који сваки од њих остварује исти уговорени интерфејс, као и из тога што су мерени различити слојеви саме имплементације.

Процес објављивача и процес претплатника, покренути на два одвојена рачунара, размењују узорке постојеће теме типа Адаптер, при чему су и њихови Уписивачи података као и Читачи података подешени да прате политике квалитета сервиса ПОУЗДАНОСТ са вредношћу ПОУЗДАНО као и ИСТОРИЈА са вредношћу ЧУВАТИ СВЕ ОДБИРКЕ.

Објављивач у постојеће поље поруке уписује редни број поруке и сопствени локални временски печат T_{pub} , где је резултујућа величина садржаја поруке реда 20 бајтова. Претплатник, одмах након што поруку преузме из реда, бележи сопствени временски печат

T_{sub} и разлику два временска печата бележи као једносмерно кашњење $L = T_{sub} - T_{pub}$. Порукe се шаљу у тачно одређеним интервалима који логаритамски варирају од узастопног слања без паузе (интервал = 0) до 10ms између узорака (100 порука/с), где се по једном интервалу пошаље 1000 порука, након чега се прелази на наредни интервал. Непосредно пре преласка на нови интервал, локални часовници на оба рачунара синхронизовани су протоколом РТР (Precision Time Protocol), како би овакво мерење преко две различите машине било ваљано. Кашњење при интервалу слања I , за поруку i дато је изразом:

$$L(I)_i = T(I)_{sub_i} - T(I)_{pub_i},$$

где $I \in \{0, 0.000001, 0.000002, 0.000005, 0.00001, \dots, 0.01\}$ представља скуп унапред дефинисаних интервала и где $i \in \{0, 1, 2, \dots, 999\}$ представља редни број мерења у оквиру једног од интервала. На овај начин, за сваки интервал слања, пошаље се тачно 1000 порука, прати се да ли су све поруке пристигле, као и кашњење између слања и пријема сваке поруке. Након завршетка слања свих порука по свим интервалима ради се прерачунавање броја примљених порука, процена брзине слања порука, проналажење минималног, средњег, максималног, као и процентуалног кашњења.

Број примљених порука се мери увећавањем бројача порука приликом пристизања порука на страни примаоца порука.

Процена брзине порука се рачуна тако што се број примљених порука по једном интервалу подели временом од слања прве поруке до пријема последње поруке. Процена је дата следећим изразом:

$$\omega'(I) = \frac{n}{T(I)_{sub_n} - T(I)_{pub_0}},$$

Где n представља број примљених порука за интервал, $T(I)_{sub_n}$ време пријема последње поруке, а $T(I)_{pub_0}$ време слања прве поруке.

За рачунање просечног кашњења на датом интервалу слања, вредности кашњења свих примљених порука се сумирају, након чега се добијена сума дели бројем примљених порука. Рачунање просечног кашњења дато је следећим изразом:

$$L_{avr}(I) = \frac{\sum_{i=0}^n L_i(I)}{n},$$

где $L_i(I)$ представља кашњење i -те поруке, а n представља број примљених порука.

Екстреми кашњења као и процентуални резултати кашњења рачунати су на следећи начин. Низ свих забележених кашњења на датом интервалу се сортира у растућем поретку, тако да важи пресликавање:

$$L(I) \in \{L_i(I) \mid i \in \{0, 1, 2, \dots, 999\}\} \xrightarrow{sort} L'(I) \in \{L'_i(I) \mid i \in \{0, 1, 2, \dots, 999\}\},$$

при чему важи следећи поредак $L'_1 \leq L'_2 \leq \dots \leq L'_n$, чиме се постиже растући поредак вредности кашњења у сортираном низу $L'(I)$. На основу тако сортираног низа проналазе се екстремни вредности кашњења. Минимално кашњење представља први елемент низа $L'(I)$, дат изразом $L_{min}(I) = L'(I)[0]$. На сличан начин максимално кашњење се проналази као последњи елемент низа $L'(I)$, коришћењем израза $L_{max}(I) = L'(I)[n - 1]$, где n представља величину низа. За рачунање процентуалних резултата кашњења користи се метода најмањег ранга дата следећим изразом:

$$L_p(I) = L'(I)[p * n],$$

Где p представља проценат броја одбирака, а n број одбирака у низу. Овим се постиже, с обзиром на чињеницу да је низ $L'(I)$ сортиран у растућем поретку, да се добије вредност кашњења која је већа од p процената одбирака вредности кашњења из низа $L'(I)$.

Поступак описан у овој секцији се затим понавља 100, при чему се бележе резултати добијени на описане начине. Резултати који су приказани у наредном одељку представљају усредњена мерења у 100 поновљених поступака.

Важно је напоменути да се доменски учесник на страни издавача уништава 5 секунди након завршетка слања, како би RTPS механизам поновног слања имао времена да опорави сваки узорак евентуално изгубљен на нивоу транспорта.

5.2 Резултати мерења

У оквиру експеримената спроведених на описан начин мерено је кашњење порука у систему, као и интензитет слања порука у оквиру система.

Резултати су прерачунати на основу усредњавања резултата више поновљених сесија мерења. Урађено је мерење кашњења сваке поруке, број порука које су успешно пристигле по интервалу по сесији, као и процена пропусности система по секунди. У табели испод дат је преглед резултата. У колони Интервал представљен је променљиви параметар интервал, који представља време које прође између слања две узастопне поруке. Колона Примљених порука представља колико је у просеку примљено порука у свакој сесији од тачно 1000 послатих порука. Колона брзина слања представља процену колико порука је послату у секунди у току сесије тестирања. У осталим колонама представљена су кашњења порука, али на такав начин да могу да дају бољу слику о перформансама система у различитим условима рада. Минимално и максимално кашњење представљају средње вредности минималног, односно максималног кашњења на одређеном интервалу у сесијама мерења. Просечно кашњење представља просечну вредност кашњења у свим сесијама на задатом интервалу. Процентуално кашњење даје просечну вредност кашњења за које наведени проценат учесника има мању вредност кашњења на задатом интервалу.

У табели испод дати су резултати мерења спроведени над адаптером базираном на DDS мидлверу, у више сесија мерења на 14 унапред дефинисаних интервала слања порука.

Интервал (s)	Примљених порука (од 1000)	Брзина пријема (порука/s)	Минимално кашњење (ms)	Просечно кашњење (ms)	Процентуално кашњење 50% (ms)	Процентуално кашњење 99% (ms)	Максимално кашњење (ms)
0.0	1000.000	4009.344	1.190	10.860	10.978	19.746	19.746
0.000001	1000.000	4475.753	1.048	51.927	53.294	96.981	97.771
0.000002	1000.000	4494.546	1.118	52.648	54.952	97.537	98.217
0.000005	1000.000	4396.282	3.768	51.834	53.429	94.375	160.279
0.00001	1000.000	4552.452	1.066	47.629	49.122	88.543	89.122
0.00002	1000.000	4528.542	0.965	42.422	43.644	79.046	79.547
0.00005	1000.000	4490.348	0.937	26.905	27.318	49.505	49.798
0.0001	1000.000	4112.808	0.172	0.322	0.239	1.176	1.848
0.0002	1000.000	2908.417	0.175	0.223	0.211	0.495	1.185
0.0005	1000.000	1469.799	0.183	0.248	0.236	0.440	0.972
0.001	1000.000	747.538	0.210	0.447	0.429	0.802	1.206
0.002	1000.000	419.004	0.226	0.523	0.522	0.905	1.295
0.005	1000.000	184.113	0.257	0.575	0.557	0.950	1.526
0.01	1000.000	95.705	0.311	0.586	0.561	0.987	1.582

Табела 1. Резултати тестова спроведених над DDS адаптером

У табели испод дати су резултати идентичног мерења спроведеног над транспортним API-ем заснованим на чистом TCP транспорту какав је примарно коришћен у PTB-FLA.

Интервал (s)	Примљених порука (од 1000)	Брзина пријема (порука/s)	Минимално кашњење (ms)	Просечно кашњење (ms)	Процентуално кашњење 50% (ms)	Процентуално кашњење 99% (ms)	Максимално кашњење (ms)
0.0	1000.000	4.497	0.229	222.831	0.496	1024.417	1028.308
0.000001	1000.000	744.351	0.130	1.495	0.152	0.424	1017.934
0.000002	1000.000	759.565	0.130	1.434	0.158	0.407	1020.327
0.000005	1000.000	767.677	0.131	1.430	0.155	0.399	1018.265
0.00001	1000.000	816.487	0.130	1.268	0.160	0.396	1021.778
0.00002	1000.000	795.959	0.131	1.302	0.165	0.371	1021.546
0.00005	1000.000	913.762	0.132	1.289	0.161	0.365	979.113
0.0001	1000.000	2648.247	0.132	0.592	0.166	0.506	406.812
0.0002	1000.000	2732.730	0.134	0.232	0.185	0.355	34.888
0.0005	1000.000	1356.511	0.151	0.275	0.252	0.633	1.529
0.001	1000.000	721.040	0.186	0.446	0.450	0.823	1.427
0.002	1000.000	412.401	0.191	0.493	0.493	0.861	1.438
0.005	1000.000	182.659	0.232	0.545	0.528	0.869	1.549
0.01	1000.000	95.219	0.279	0.572	0.543	0.979	1.573

Табела 2. Резултати тестова спроведених над TCP адаптером

5.3 Анализа и поређење резултата

Резултати приказани у Табели 1 и Табели 2 показују начине понашања оба адаптера под истим условима у истом тестном окружењу. Адаптер заснован на DDS мидлверу и оригинални МРАPI транспорт понашају се на суштински различит начин у тестовима који представљају веће оптерећење, односно код учесталог слања порука, док на интервалима већим од приближно 1ms њихове перформансе практично конвергирају.

У опсегу интервала од 0 до 0,00005s адаптер остварује стабилну пропусност у приближном распону од 4000 до 4500 порука у секунди (Табела 1, колона брзина пријема порука), при чему просечно кашњење износи 52,648ms на интервалу 0,000002s, да би на интервалу 0,00005s опало на 26,905ms секунди (Табела 1, колона просечно кашњење). Овај почетак опадања просечног кашњења може се уочити на слици 14 у тренутку 50μs. Максимално кашњење на истом опсегу од 0 до 0,00005s остаје прилично уједначено, што се може видети на слици 17.

На интервалу 0,0001s кашњење нагло опада (просек 0,322ms, максимум 1,848ms, табела 1, интервал 0,0001s), након чега се пропусност адаптера смањује сразмерно променљивом интервалу слања, што показује да од те тачке надаље транспортни трошак адаптера практично престаје да буде ограничавајући фактор, где систем постаје ограничен темпом којим тест сам намеће слање порука, а не брзином преноса. Овај нагли пад кашњења се може пратити на сликама 13, 14, 15, 16 и 17 у тренуцима након 50μs. Остварена брзина открива границу на страни пријема која важи за интервале од 0 до 50μs, на отприлике 4.000–4.550 порука/с. Тек од 100μs навише остварена брзина почиње да прати задати интервал, уместо границе. Кашњење прати исту поделу, где унутар тог опсега, политика сервиса ПОУЗДАНО поновним слањем опорави сваки узорак уместо да га изгуби, али је просечно кашњење при том свуда повишено од 11 до 53ms (Табела 1, колона просечно кашњење на опсегу од 0 до 50μs).

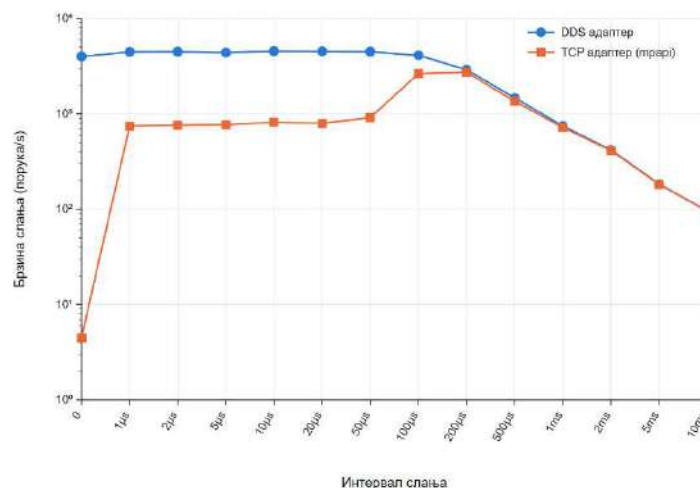
Понашање МРАPI транспорта на интервалу 0 битно одступа од утврђених образаца, где пропусност пада на свега приближно 4,5 порука у секунди, а просечно кашњење скаче на 222,831ms, упркос томе што медијана кашњења износи само 0,496ms (Табела 2, интервал 0). Овако велики раскорак између медијане и просека указује на расподелу кашњења, где већина од 1000 послатих порука стиже готово тренутно, док мањи број порука трпи кашњење блиско једној секунди (максимум 1028,308ms, процентуално кашњење за 99% чак 1024,417ms).

На најмањем ненултом интервалу 0,000001s пропусност МРАPI транспорта нагло расте на 744,351 порука у секунди, а просечно кашњење пада на свега 1,495ms (Табела 2, интервал 0,000001s) и те вредности током даљег тестирања остају у сличном опсегу (744 до

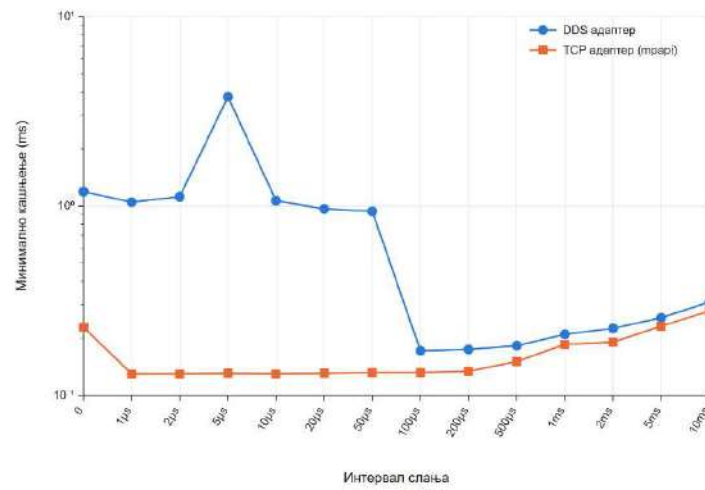
914 порука у секунди (слика 12) уз просечно кашњење испод 1,5ms) све до интервала 0.00005s. Максимално кашњење у истом опсегу интервала остаје изнад 979ms, готово идентично вредностима измереним на интервалу 0 (Табела2, максимално кашњење на интервалу од 0 до 50 μ s). То показује да мали број порука и даље наилази на заузету везу.

Од интервала 0.0001s максимално кашњење почиње приметно да опада (406,812ms), затим 34,888ms на интервалу 0.0002s, да би од интервала 0.0005s пало испод 2ms (слика 17). Из овога се може закључити да је интервал потребан да опорави просечну пропусност око 100 пута дужи од оног интервала који уклања ризик од појединачних кашњења блиских једној секунди.

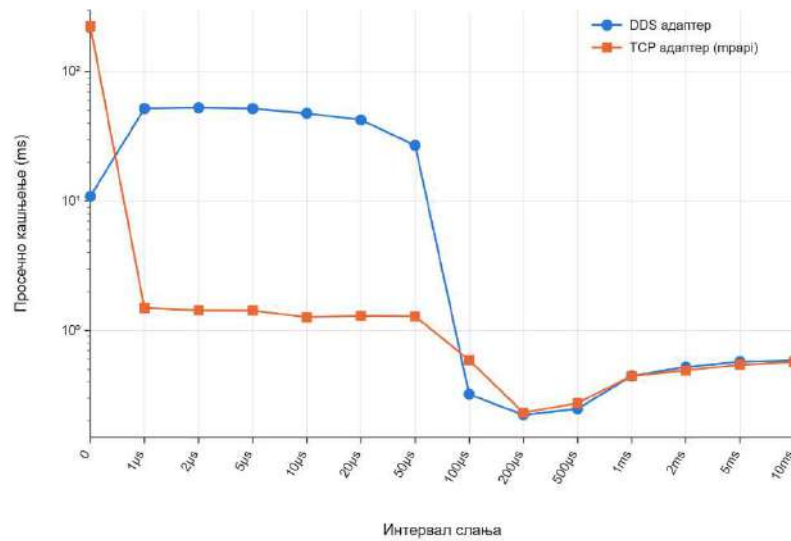
У целом опсегу интервала од 0 до 0.0005s DDS адаптер остварује већу пропусност од МРАПІ транспорта на сваком појединачном интервалу, тек од интервала 0.001s надаље разлика постаје занемарљива (упоредити криве на слици 12), док се просечна кашњења два транспорта се готово поклапају (слика 14). На овим већим интервалима оба транспорта постају ограничена темпом слања, а не сопственим трошком преноса, па разлике у имплементацији престају да буду мерљиве.



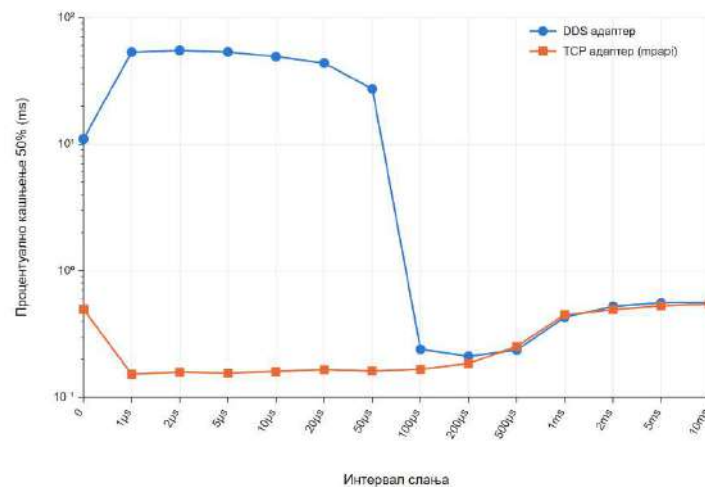
Слика 12. Поређење вредности брзина слања порука по интервалу



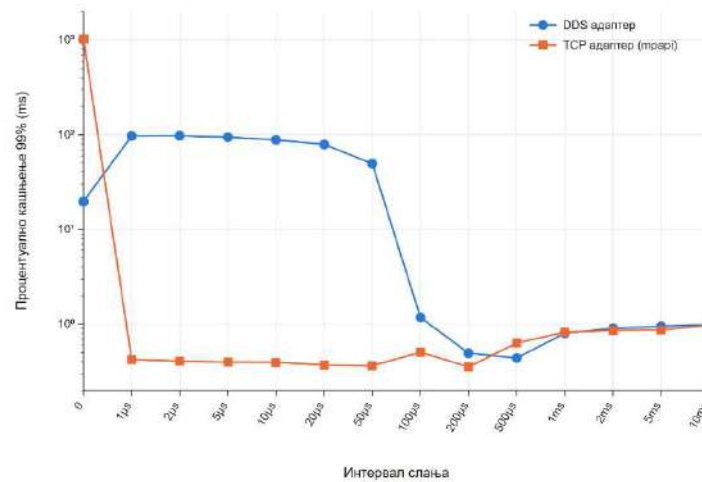
Слика 13. Поређење минималног кашњења по интервалу



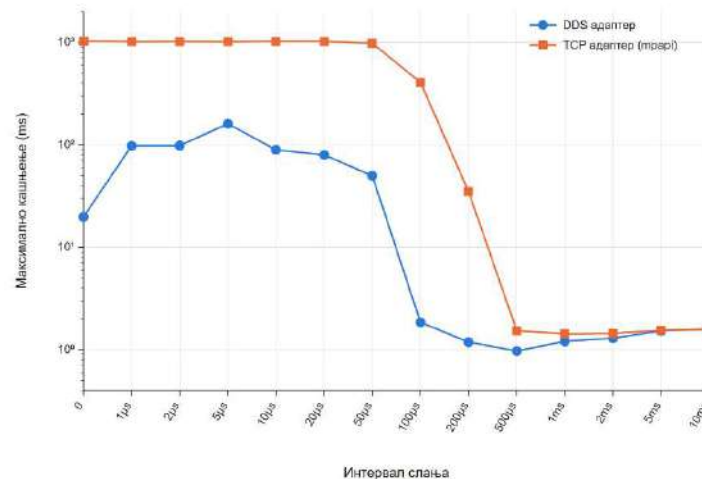
Слика 14. Поређење просечног кашњења по интервалу



Слика 15. Поређење процентуалног кашњења на 50% узорака (медијана) по интервалу



Слика 16. Поређење процентуалног кашњења на 99% узорака по интервалу



Слика 17. Поређење максималног кашњења по интервалу

Гледано у целини, ниједан узорак није изгубљен ни у једној сесији мерења, без обзира на транспорт или интервал слања. Главна разлика између обе врсте транспорта је у начину на који се та поузданост остварује под оптерећењем. Адаптер, ослањајући се на комбинацију ПОУЗДАНО и ЧУВАТИ СВЕ ОДБИРКЕ политике квалитета сервиса, у сваком тренутку показује постепену и ограничену деградацију кашњења, где чак и у најнеповољнијем измереном случају максимално кашњење није прешло 0.16 s. Механизам понављања оригиналног МРАРИ транспорта, заснованог на ТСР-у нема еквивалентну горњу границу деградације на нивоу појединачног узорака, испорука остаје загарантована. У пракси то значи да МРАРИ транспорт под оптерећењем прелази нагло из готово идеалног у готово потпуно заустављено стање, док адаптер под истим оптерећењем прелази кроз низ међустања са мерљиво растућим, али ограниченим кашњењем.

6. Закључак

Овим радом дат је преглед процеса развоја транспортног адаптера између PTB-FLA окружења за развој алгоритама федеративног учења и DDS мидлвера. Поред процеса развоја дат је и преглед имплементационих детаља, као и преглед експерименталне евалуације решења. Уз експерименталну евалуацију приложено је и поређење са постојећим транспортом PTB-FLA окружења, као и поређење са перформансама приложеним од стране организације eProxima, која је развила имплементацију DDS стандарда, коришћену за потребе овог рада.

У овом раду полази се од конкретног недостатка транспортног слоја који PTB-FLA окружење користи, а то су одсуство механизма откривања учесника и конфигурабилног квалитета сервиса, физичког протокола погодног за рад на ивици интернета у његовом транспортном слоју, услед чега постоји ограничење на извршавање свих инстанци на једном физичком рачунару. Циљ рада био је да се тај транспортни слој замени решењем заснованим на DDS/RTPS протоколу, а да при том остатак PTB-FLA окружења остане потпуно непромењен.

Експериментална евалуација потврђује да овако остварена транспарентност замене није дошла на уштрб поузданости или предвидивости понашања система. Ниједан узорак није изгубљен ни при једној тестираној брзини слања, било код адаптера базираног на DDS-у, било код оригиналног транспорта, главна разлика између њих, дакле, никада није у самој поузданости испоруке, него у начину на који се она остварује под оптерећењем. За типичан образац рада PTB-FLA окружења, у коме се размена порука између учесника одвија периодично, између кругова федеративног учења, а не у виду непрекидног низа порука послатих без паузе, већина стварних сценарија пада управо у опсег интервала у коме два транспорта показују упоредиве перформансе. Адаптер нуди транспорт чије се најгоре

измерено кашњење, чак и у услову непрекидног слања без паузе, задржава и до шест пута ниже од одговарајуће вредности измерене код оригиналног транспорта.

Ове тврдње о поузданости и пропусности адаптера треба посматрати као условна побољшања. Поузданост јесте потпуна у измереном опсегу, али цена преоптерећења система испољава се као додатно кашњење, а не као стварни губитак узорка, што сугерише да систем не отказује нагло, него да предвидиво успорава. Пропусност, са друге стране, остаје ограничена: за поруке садржаја од отприлике 20 бајтова, какве користи PTB-FLA у типичном раду, адаптер на страни пријема достиже око 4000 порука у секунди, након чега свако додатно скраћивање интервала слања повећава искључиво кашњење, а не стварно остварену брзину преноса.

Имплементација адаптера ослања се на SWIG-ом генерисане Python C-екстензије, што захтева CPython окружење и самим тим искључује директну преносивост на MicroPython платформу, коју MPT-FLA управо циља, транспортни слој предложен у овом раду тако решава проблем откривања учесника и квалитета сервиса на нивоу локалне мреже, али не и проблем извршавања на ресурсима ограниченим микроконтролерима.

У целини, овај рад показује да се DDS/RTPS протокол може увести као транспортни слој PTB-FLA радног оквира на начин који апликативном коду и генеричким алгоритмима федеративног учења остаје потпуно невидљив, уз задржану поузданост испоруке и предвидиво, ограничено понашање под оптерећењем, чиме PTB-FLA добија основу за рад у стварним ивичним мрежама, а не само на једном физичком рачунару.

7. Литература

- [1] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, B. A. y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," u Proc. 20th Int. Conf. Artificial Intelligence and Statistics (AISTATS), JMLR: W&CP vol. 54, pp. 1–10, 2017.
- [2] M. Popovic, M. Popovic, I. Kastelan, M. Djukic, S. Ghilezan, "A Simple Python Testbed for Federated Learning Algorithms," u Proc. 2023 Zooming Innovation in Consumer Technologies Conf. (ZINC), 2023, pp. 148–153.
- [3] M. Popovic, M. Popovic, I. Kastelan, M. Djukic, I. Basicovic, "MicroPython Testbed for Federated Learning Algorithms – The Extended Paper," arXiv:2405.09423, 2024.
- [4] Object Management Group, "Data Distribution Service (DDS), Version 1.4," OMG Document formal/2015-04-10, 2015.
- [5] Object Management Group, "The Real-time Publish-Subscribe Protocol (RTPS) DDS Interoperability Wire Protocol Specification, Version 2.5," OMG Document formal/2022-04-01, 2022.
- [6] P. Kairouz, H. B. McMahan, B. Avent, et al., "Advances and Open Problems in Federated Learning," arXiv:1912.04977, 2021.
- [7] M. Aledhari, R. Razzak, R. M. Parizi, F. Saeed, "Federated Learning: A Survey on Enabling Technologies, Protocols, and Applications," IEEE Access, vol. 8, pp. 140699–140725, 2020.

-
- [8] T. Li, A. K. Sahu, A. Talwalkar, V. Smith, "Federated Learning: Challenges, Methods, and Future Directions," arXiv:1908.07873, 2019.
 - [9] R. Rahman, "Federated Learning: A Survey on Privacy-Preserving Collaborative Intelligence," arXiv:2504.17703, 2025.
 - [10] K. Peeroo, P. Popov, V. Stankovic, "A Survey on Experimental Performance Evaluation of Data Distribution Service (DDS) Implementations," arXiv:2310.16630, 2023.
 - [11] G. Cleland, D. Wu, R. Ullah, B. Varghese, "FedComm: Understanding Communication Protocols for Edge-based Federated Learning," arXiv:2208.08764, 2022.
 - [12] M. Amadeo, C. Campolo, G. Ruggeri, A. Molinaro, "Improving Communication Performance of Federated Learning: A Networking Perspective," Computer Networks, vol. 267, art. 111353, 2025.
 - [13] R. Stikovic, "DDS-adapter: software adapter between PTB-FLA and DDS middleware," GitHub repository, 2026. [Online]. Available: <https://github.com/ratkost00/DDS-adapter>