УНИВЕРЗИТЕТ У БЕОГРАДУ
МАТЕМАТИЧКИ ФАКУЛТЕТ


Срђан Лазаревић

# UART КАО МЕДИЈУМ ЗА КОМУНИКАЦИЈУ ИЗМЕЂУ УРЕЂАЈА СА УГРАЂЕНИМ РАЧУНАРОМ

мастер рад

Београд, 2022.

**Ментор**:

др Милена Вујошевић Јаничић, ванредни професор
Универзитет у Београду, Математички факултет

**Чланови комисије**:

др Александар Картељ, доцент
Универзитет у Београду, Математички факултет

др Мирко Спасић, доцент
Универзитет у Београду, Математички факултет

**Датум одбране**:

*Клари*

**Наслов мастер рада**: *UART* као медијум за комуникацију између уређаја са уграђеним рачунаром

**Резиме**: Све већем броју уређаја са уграђеним рачунаром потребан је медијум за комуникацију, како би добавили податке које поседују други уграђени рачунари или мреже уграђених рачунара. Један од најпопуларнијих медијума за комуникацију је Етернет, али протоколи засновани на Етернету захтевају велику брзину преноса података и брзину процесора, које нису увек доступне. Стога поред Етернета, често су у употреби додатни, мање интензивни, медијуми за комуникацију као што је, на пример, *UART*. *UART* припада слоју везе референтног модела *OSI*, те је за поуздану комуникацију неопходно имплементирати слојеве вишег нивоа. Овај рад описује имплементацију протокола *YAP3* који користи *UART* као медијум за комуникацију. Протокол *YAP3* имплементира механизме ретрансмисије, детекције и опоравка од грешке у оквиру слојева виших нивоа референтног модела *OSI* како би пружио поуздан канал за комуникацију.

**Кључне речи**: рачунарске мреже, комуникациони протоколи, протокол *UART*, уређаји са уграђеним рачунаром

# Садржај

# Глава 1

# Увод

Развој нових технологија у области хардвера и софтвера све више утиче на друштво. Димензије рачунара постају све мање што омогућава њихову уградњу у различите типове уређаја. Модерна архитектура хардвера често предлаже употребу више уграђених рачунара (енг. *embedded devices*) у оквиру једног уређаја. Предлог овакве архитектуре проистиче из потребе за паралелизацијом операција, како би омогућили боље перформансе уређаја. Паралелизација операција на нивоу уграђених рачунара доноси проблем синхронизације и размене информација између уграђених рачунара. Како би се уграђени рачунари синхронизовали или разменили било какве информације неопходно је повезати рачунаре медијумом за комуникацију. Поред Етернета, који представља један од најчешће коришћених медијума за комуникацију, често су у употреби и други медијуми као што је, на пример, *UART* (енг. *universal asynchronous receiver-transmitter*).

*UART* припада медијумима за серијску комуникацију, при чему серијска комуникација подразумева слање и примање података у виду низова битова. Како су медијуми за серијску комуникацију подложни грешкама и губитцима података током транспорта, неопходно је имплементирати механизме ретрансмисије, детекције и опоравка од грешке. Имплементација претходно наведених механизама подразумева имплементацију протокола који имплементира наведене механизме. *HDLC* (енг. *high level data link control*) и *PPP* (енг. *point-to-point protocol*) представљају концептуалне моделе протокола заснованих на серијској комуникацији који описују како имплементирати претходно наведене механизме. Један од недостатака ових концептуалних модела је да не постоји референтна имплементација која се може користити или надоградити

за потребе комуникације између уграђених рачунара.

У оквиру овог рада имплементиран је протокол *Још један протокол за директну комуникацију*, односно *YAP3* (енг. *yet another peer to peer protocol*), који користи *UART* као медијум за комуникацију, при чему се може генерализовати и за остале медијуме за серијску комуникацију. Протокол имплементира неке од механизама ретрансмисије, детекције и опоравка од грешке описаних у оквиру концептуалних модела *HDLC* и *PPP*. Протокол се може користити за претходно наведене проблеме синхронизације и размене протокола између уграђених рачунара. У оквиру рада приказано је и теоријско поређење протокола *YAP3* са концептуалним моделима *HDLC* и *PPP*.

Наставак рада који описује коришћене концепте и имплементацију организован је на следећи начин. У глави 2 описан је референтни модел *OSI* (енг. *open systems interconnection reference model*) који представља један од најзначајнијих теоријских модела описа слојевите архитектуре мрежа. Слојевита архитектура представља један од главних механизама енкапсулације функционалности како мрежа тако и протокола који се извршавају у оквиру мреже. Глава 3 описује *UART* као медијум за комуникацију. Обрађен је општи принцип рада и размене података, као и елементарни концепти детекције грешке на основу парности. У глави 4 описани су концепти ретрансмисије и опоравка од губитка порука помоћу технике клизних прозора. Дат је преглед неколико различитих подешавања технике које различито утичу на пропусност медијума за комуникацију. Глава 5 описује протокол *YAP3*. Описана је имплементација наведених механизама ретрансмисије, детекције и опоравка од грешке. Поред описа имплементације, описан је и начин развоја и тестирања протокола у локалном окружењу. Додатно, дато је теоријско поређење протокола *YAP3* са концептуалним моделима. У оквиру главе 6 изнети су закључци овог рада.

# Глава 2

# Референтни модел *OSI*

Референтни модел *OSI*, односно референтни модел за отворено повезивање система представља један од најкоришћенијих апстрактних описа архитектуре рачунарских мрежа [16, 4]. Описује интеракцију уређаја (хардвера), програма, сервиса (софтвера) и протокола при мрежним комуникацијама. Користе га произвођачи при пројектовању мрежа, стручњаци при изучавању мрежа, као и професори у сврхе подучавања [3].

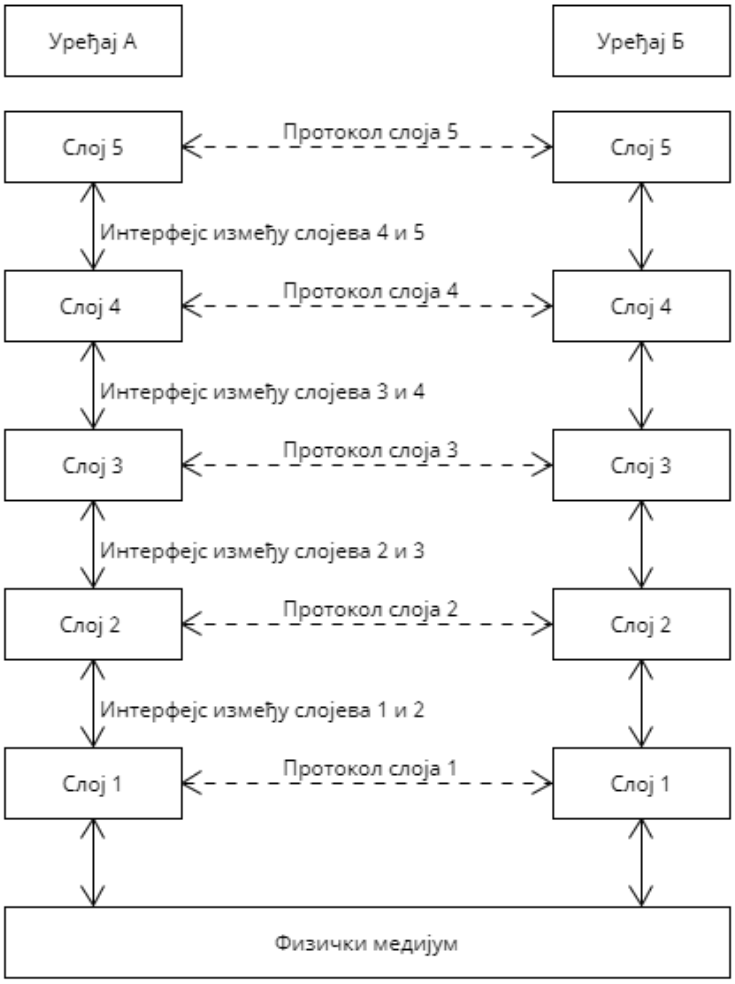
Већина сервиса и протокола, у контексту комуникационих мрежа, функционишу као јединствена целина. Да би се описала веза између њих, њихова функција у сваком од процеса комуникације, а исто тако и да би се лакше разумео сам процес, уведена је концептуална скица, тј. референтни модел. Још један од разлога увођења референтног модела је и да се изврши стандардизација самих протокола. У самим почецима настајања мрежа, различите фирме су примењивале своја решења, тако да је комуникација била могућа само између рачунара у оквиру мреже истог произвођача.

Крајем 1979. године, међународна организација за стандардизацију ISO (енг. *international organization for standardization*), креирала је референтни модел *OSI* да би се превазишли ови проблеми, док је 1983. године представљена ревизија овог модела која је постала међународни стандард и водич за умрежавање [6, 4].

## 2.1 Слојевита архитектура

Да би упростили комплексан дизајн, већина мрежа је оранизована слојевито, тако да је сваки слој заснован на слојевима изнад и испод. Приказ

претходно наведене архитектуре, илустрован је сликом 2.1. Број слојева, име сваког слоја и његова функција се разликују од мреже до мреже. Сврха сваког слоја је да имплементира одређене функционалности, при чему не излаже детаље имплементације, слојевима вишег и нижег нивоа.




Слика 2.1: Слојеви, протоколи и интерфејси [16].

Референтни модел *OSI* има седам слојева [4]. Правила и критеријуми на основу којих су слојеви изабрани се могу укратко дефинисати на следећи начин [16]:

1. Слој се креира када је потребно апстраховати неку функционалност.

2. Сваки слој треба да омогућава прецизно дефинисану функционалност.

3. Функционалност сваког слоја мора узети у обзир станардизацију те функционалности на интернационалном нивоу.

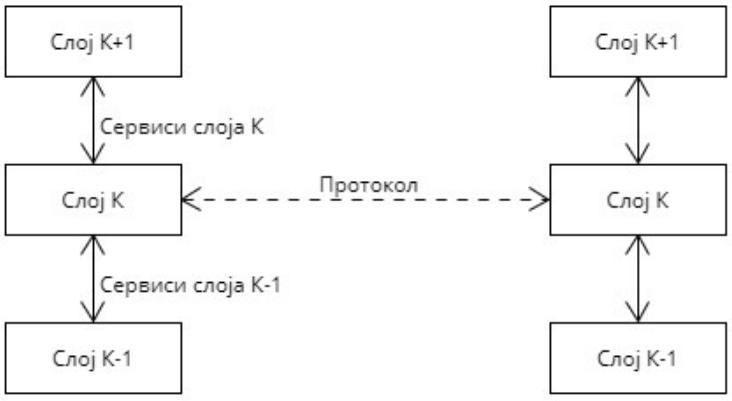
4. Сваки слој мора да има јасно дефинисане границе, како би се максимално умањило дељење информација са другим слојевима.

5. Број слојева мора бити довољно велики да различите функционалности не буду у оквиру истог слоја уколико то није изричито неопходно, али и довољно мали тако да архитектура не постане гломазана.

Модел *OSI* не представља архитектуру мреже, јер не наводи које тачно сервисе и протоколе треба користити на сваком од задатих нивоа [4, 16]. Модел само наводи шта који слој треба да ради.

## 2.2 Сервиси и протоколи

Сервиси и протоколи представљају различите концепте [16, 3]. Илустрација односа сервиса и протокола приказана је на слици 2.2.




Слика 2.2: Однос сервиса и протокола [16].

Сервис представља операцију које слој пружа слоју изнад себе, при чему не излаже ни један детаљ о томе како је операција имплементирана. Сваки слој може имати више сервиса. Скуп сервиса је заправо интерфејс слојева,

при чему слој нижег нивоа пружа одређену операцију или више њих, док је слој изнад њега корисник тих операција.

Протокол, за разлику од сервиса, представља скуп правила који дефинишу формат и значење структура података унутар слоја које се размењују између уређаја. Како протокол дефинише формат и значење, протокол може мењати интерно формат или значење неке структуре података све док не мења сервисе који припадају том слоју. На овај начин сервиси и протоколи су комплетно раздвојени.

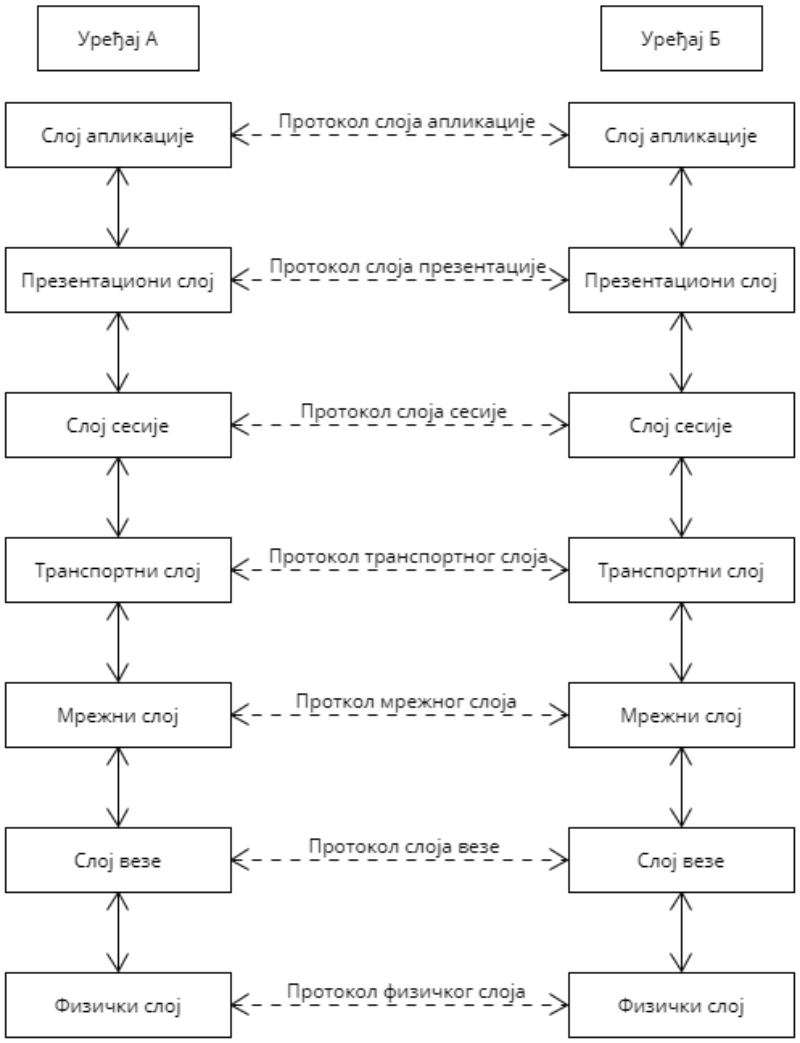
За илустрацију односа сервиса и протокола може се применити аналогија из програмских језика. У тој аналогији, сервис би представљао неку апстрактну структуру података или инстанцу неког објекта у случају објектно оријентисаног програмског језика. Та струкура података, односно објекат, дефинише које операције је могуће урадити са тим објектом, али не прецизира како су те операције имплементиране. Насупрот сервису, протокол описује интерну имплементацију тих операција, при чему имплементација није директно изложена крајњем кориснику [16].

## 2.3 Модел

Слојеви модела OSI, приказаног на слици 2.3, описани су у наставку текста.

**Физички слој** (енг. *physical layer*) је задужен за пренос сировог сигнала, у виду битова, преко комуникационог канала. Како су вредности битова бинарне, задатак физичког слоја је да осигура поуздану трансимисију, односно ако пошиљалац пошаље бит са вредношћу један онда и прималац мора да прими бит са вредношћу један, а не бит са вредношћу нула.

**Слој везе** (енг. *data link layer*) омогућава локалну размену података између уређаја на истом нивоу мреже. Структуре података које се размењују на овом нивоу мреже се називају оквири (енг. *frame*). Оквири не прелазе границе локалне мреже, што омогућава протоколима слоја везе да се фокусирају на пренос, адресирање и контролу приступа медијуму за комуникацију.




Слика 2.3: Референтни модел *OSI* [16, 4].

На овом слоју се најбоље види смисао слојевите архитектуре, јер без слоја везе, протоколи вишег нивоа, као што је на пример *IP* (енг. *internet protocol*), морали би да дефинишу приступ различитим типовима медијума за комуникацију [2].

**Мрежни слој** (енг. *network layer*), за разлику од слоја везе, омогућава раз-

мену података између уређаја који не припадају истом нивоу мреже, односно не деле исту локалну мрежу. Мрежа, која може обухватати више повезаних локалних мрежа уређаја, представља медијум за комуникацију на овом нивоу.Како сваки уређај на мрежи има своју адресу, то омогућава повезаним уређајима пренос података који садрже само адресу примаоца и садржај поруке, при чему се ослањају на мрежу да пронађе начин да испоручи поруку до примаоца.

Протоколи мрежног слоја не гарантују достављање порука примаоцу, те стога комуникација на овом слоју није у потпуности поуздана [3, 15]. Може се десити да у одређеним ситуацијама, као што је повећани мрежни саобраћај (енг. *network traffic load*), дође до губитка поруке, поновног слања исте поруке или генералног кашњења у достави порука.

**Транспортни слој** (енг. *transport layer*) имплементира потребне механизме како би размена порука преко мрежног слоја била поуздана. Основна улога транспортног слоја је да прихвати структуре података са вишег слоја, по потреби их подели у структуре погодне за транспорт преко мрежног слоја, те осигура да су сви делови структуре података примљене са вишег слоја успешно достављене примаоцу.

Најпопуларнији протоколи транспортног слоја су протокол *TCP* (енг. *transmission control protocol*) који имплементира секвенцијално слање порука који обезбеђује да прималац прими поруке редоследом којим је пошиљалац слао [2], као и протокол *UDP* (енг. *user datagram protocol*) који имплементира транспорт изолованих порука без гаранције о редоследу испоруке [15].

Треба имати у виду да је немогуће постићи комплетно поуздан канал за комуникацију, те да је оно што термин *поуздан канал за комункацију* представља, заправо канал чија је стопа грешка довољно ниска да се занемарује у пракси [16].

**Слој сесије** (енг. *session layer*) пружа механизме за успостављање, прекидање и управљање везом, односно сесијом, за комуникацију између корисничких програма на различитим уређајима. Комуникациона веза се састоји од порука, односно захтева и одговара, који се дешавају између корисничких програма, при чему сервиси овог слоја брину о синхронизацији комуникационе везе тако што онемогућавају неке критичне опе-

рације да се одвијају на оба уређаја истовремено или воде евиденцију о томе који уређај је послао поруку последњи. Слој сесије често пружа и сервисе који осигуравају квалитет везе, те у случају проблема са везом обавештавају слојеве вишег нивоа.

**Презентациони слој** (енг. *presentation layer*) се често назива и слој *синтаксе и семантике*. Осигурава да структуре података које долазе са слоја апликације имају одговарајућу репрезентацију која одговара уређајима на којима се извршавају програм пошиљаоца и програм примаоца, при чему ти уређаји могу бити тотално различити. Презентациони слој је задужен за превођење поруке у стандардну репрезентацију коју пошиљалац и прималац разумеју. Стандардна репрезентација коју пошиљалац и прималац разумеју се најчешће огледа у својеврсном шифровању (енг. *encryption*) и дешифровању (енг. *decryption*) порука неком од метода шифровања, односно дешифровања, при чему само шифровање представља и један вид заштите од злоупотребе података.

**Слој апликације** (енг. *application layer*), за разлику од осталих слојева референтног модела *OSI*, доставља сервисе директно крајњим корисницима уређаја, односно апликацијама које се налазе ван модела *OSI*. Због разноврсности програма који размењују поруке на различите начине, овом слоју припада највећи број протокола.

# Глава 3

# Комуникациони протокол *UART*

Избор протокола има значајну улогу у управљању комуникацијом између уређаја. Протокол се дизајнира према неопходним захтевима система (енг. *system requirements*) и подразумева се да уређаји који користе протокол имају унапред задато подешавање која је компатибилна са свим уређајима који учествују у комуникацији.

Универзални асинхрони прималац-пошиљалац, односно *UART* [10], представља један од најкоришћенијих комуникационих протокола за директну серијску комуникацију (енг. *point to point serial communication*) између уређаја. Уређаји са уграђеним рачунарима, микроконтролери, као и конвенционални рачунари често користе протокол *UART* за комуникацију са другим уређајима са уграђеним рачунаром или микроконтролером. *UART* представља згодно решење у претходно наведеној ситуацији, јер су за комуникацију неопходне само две жице, једна за слање и једна за примање података [1].

По дефиницији, *UART* је хардверски комуникациони протокол који се заснива на асинхроној серијској комуникацији са подесивом брзином слања (енг. *baud rate*) података [11]. Комуникација је асинхрона јер не користи сат као механизам за синхронизацију између уређаја који шаље и уређаја који прима податке.

Често се мисли да протокол *UART* припада физичком слоју референтног модела *OSI*. Међутим, према моделу *OSI*, протокол *UART* припада слоју везе података. Припадност слојевима референтног модела *OSI* биће дискутована у поглављу 3.4, након упознавања са детаљима протокола *UART*.

## 3.1 Принцип рада

Како је раније наведено, за *UART* комуникацију потребне су две жице. Жице представљају медијум за комуникацију и често се називају серијске линије (енг. *serial line*) за комуникацију, односно линије са серијско слање и примање података [1].

Сваки *UART* се састоји од линије која шаље податке (енг. *transmitter*) и линије која прима податке (енг. *receiver*) [10]. Линија која је задужена за слање података на страни пошиљаоца представља линију задужену за примање на страни примаоца. Слично важи и за комуникацију у обрнутом смеру.

Серијско слање и примање података се реализује помоћу слања и примања електричног сигнала преко линија за слање и примање података. Свака порука која се шаље, шаље се у виду серијализоване структуре података бит по бит. Бит може имати само две вредности, логичку јединицу или логичку нулу.

UART користи напонске нивое (енг. *voltage levels*) за репрезентацију логичке јединице или логичке нуле на линији слања, односно примања. Један ниво представља логичку јединицу, док други ниво представља логичку нулу. Тачан опсег напонских нивоа зависи од стандарда дефинисаних на физичком слоју. Неки од најкоришћенијих стандарда и напонских нивоа су:

1. Већина уређаја користи 0V (енг. *Volt*) за логичку нулу и 3.3V за логичку јединицу, док неки уређаји користе 0V за логичку нулу и 1.8V за логичку јединицу.

2. RS232 (енг. *Recommended Standard 232*) прописује коришћење негативог напона за репрезентацију логичке јединице, тачније напона у оспегу од $[-15, -3]$V [14]. Позитивни напон у опсегу $[3, 15]$V представља логичку нулу.

3. Ардуино Уно (енг. *Arduino Uno*) платформа користи 0V за логичку нулу и 5V за логичку јединицу [8].

Да би се избегло прегоревање уређаја, неопходно је да оба уређаја користе исте или приближне напонске нивое за репрезентацију логичке нуле или логичке јединице.

Како је слање и примање података асинхроно, потребно је унапред одредити брзину слања и примања података. Брзина слања, односно примања, се дефинише као количина битова које се може послати, односно примити, у јединици времена. За јединицу времена се најчешће користи секунд, док су подржане брзине слања најчешће `110, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 38400, 57600, 115200, 128000` или `256000` битова у секунди [10, 1, 11].

UART на страни пошиљаоца (енг. *transmitting UART*) шаље податке према такту свог интерног сата, док *UART* на страни примаоца (енг. *receiving UART*) прима податке према такту свог интерног сата. Због тога је неопходно ускладити брзину слања на обе стране, како би се избегле грешке у интерпретацији послатих порука.

## 3.2 Пренос података

Пренос података се врши у форми пакета. Сваки пакет који се шаље је записан у формату приказаном на слици 3.1.

| Старт бит<br>1 бит | Битови података<br>5 до 9 битова | Бит парности<br>0 или 1 бит | Стоп битови<br>1 или 2 бита |
|---|---|---|---|

Слика 3.1: Формат *UART* пакета [11].

**Почетни бит** (енг. *start bit*) означава почетак сваког пакета који се шаље.

Линија задужена за слање се обично држи на високом напонском нивоу када нема размене података. *UART* на страни пошиљаоца, спушта напон на низак ниво током једног циклуса системског сата како би сигнализирао почетак трансмисије. Управо тај сигнал представља почетни бит. Када *UART* на страни примаоца примети промену нивоа напона на линији задуженој за примање, тада почиње да чита битове података задатом брзином.

**Битови података** (енг. *data bits*) представљају поруку која се шаље. У случају да се користе битови парности, порука може бити од 5 до 8 битова

дужине. Уколико се не користе битови парности дужина поруке може бити 9 битова. У већини случајева, поруке се шаљу тако да се бит најмање тежине (енг. *the least significant bit*) шаље први.

**Бит парности** (енг. *parity bit*) означава парност укупног броја логичких јединица у битовима података. Уколико је укупан број логичких јединица у битовима података паран, бит парности се поставља на логичку нулу (енг. *even parity*), у даљем тексту 0, у супротном бит парности се поставља на логичку јединицу (енг. *odd parity*), у даљем тексту 1. Бит парности представља механизам за детакцију грешке, уз помоћ којег *UART* на страни примаоца може уочити ако се вредност неког бита променила током транспорта преко жице. Вредност бита се може променити под утицајем електромагнетног зрачења, не подударања брзине слања и бризине примања или у случају оштећења физичког медијума, односно жице.

Након што *UART* на страни примаоца прими битове података, *UART* на страни примаоца израчунава парност укупног броја логичких јединица и упоређује израчунату парност и парност израчунату на страни пошиљаоца. Уколико се израчуната парност и парност израчуната на страни пошиљаоца подударају, *UART* на страни примаоца прихвата поруку и преслеђује је на даљу обраду. Уколико се парности не подударају, *UART* на страни примаоца одбацује поруку.

Недостатак оваквог приступа представља случај у којем паран број битова промени своју вредност током транспорта. На пример, послати битови података су `10101010` и парност је постављена на 0, јер имамо паран број 1. Уколико под утицајем спољног фактора два бита промене вредности и *UART* на страни примаоца прими `01101010`, израчуната парност ће и даље бити 0, те ће *UART* на страни примаоца прихватити поруку иако порука није исправна.

**Зауставни битови** (енг. *stop bits*) представљају сигнал за крај једног пакета. Заустављање преноса се реализује тако што *UART* на страни пошиљаоца подиже напон на жици задуженој за слање током једног или два, у зависности од дужине зауставног сигнала, циклуса системског сата.

## 3.3 Заглавље `termios.h`

Приступ протоколу *UART* одвија се кроз менаџер ресурса (енг. *resource manager* или *driver*) који пружа оперативни систем. За сваки нови уређај који се повеже, менаџер ресурса оперативног система креира нову приступну тачку (енг. *access point*) [7].

Приступна тачка се најчешће назива терминал (енг. *terminal*), при чему име терминал има историјско значење. У раним фазама развоја рачунара корисници су приступали уређајима са UNIX системима помоћу терминала користећи серијску комуникацију попут протокола *UART* заснованог на стандарду RS232. Терминал је најчешће био монитор са катодном цеви (енг. *cathode ray tube*) који је могао да прикаже карактере и, у неким случајевима, примитивну графику, док су се у неким случајевима као терминали користили телепринтери (енг. *teleprinter* или *teletype*).

На UNIX системима, за сваки повезани уређај оперативни систем креира приступну тачку у виду виртуелног терминала. Виртуелни терминал је назван именом формата `/dev/ttyN`, при чему `N` представља редни број терминала [7]. Један такав терминал представља и командна линија. Да би проверили који виртуелни терминал користи командна линија, може се искоритити команда `tty` [18].

```
typedef unsigned long   tcflag_t;

struct termios {
  tcflag_t c_iflag;     /* maska bitova za podesavanje
                           ulaznih podataka */
  tcflag_t c_oflag;     /* maska bitova za podesavanje
                           izlaznih podataka */
  tcflag_t c_cflag;     /* maska bitova za podesavanje
                           parametara kontrole toka */
  tcflag_t c_lflag;     /* maska bitova za napredno
                           podesavanje ulaznih podataka */
  cc_t     c_cc[NCCS]; /* specijalni karakteri */
};
```

Листинг 3.1: Структура података `termios`.

Заглавље `termios.h`, програмског језика C, пружа интерфејс за подешавање виртуелних терминалима [17, 12]. `termios.h` користи структуру података `termios`, приказану у листингу 3.1, за подешавање и добављање подеша-

вања виртуелних терминала. Функције `tcgetattr` и `tcsetattr`, приказане у листингу 3.2, омогућавају добављање и промену подешавања терминала.

```
/* Dobavljanje trenutnog podesavanja */
int tcgetattr(int fd, struct termios *termios_p);
/* Postavljanje novog podesavanja */
int tcsetattr(int fd, int optional_actions,
            const struct termios *termios_p);
```

Листинг 3.2: Подешавање терминала.

Помоћу претходно наведене структуре и функција може се мењати подешавање терминала за серијску комуникацију. На примеру, приказаном у листингу 3.3, демонстриран је програм за унос шифре преко командне линије. Како би обезбедили да се шифра не приказује на екрану током уноса потребно је променити подешавање терминала тако да се у маски битова за напредна подешавања улазних података, `c_lflag`, бит на позицији 8, дефинисан макроом `ECHO`, постави на 0.

```
#include <stdio.h>
#include <termios.h>

int main(int argc, char *argv[])
{
    printf("Unesite sifru: ");
    struct termios term;
    tcgetattr(fileno(stdin), &term);
    term.c_lflag &= ~ECHO;
    tcsetattr(fileno(stdin), 0, &term);
    char passwd[32];
    fgets(passwd, sizeof(passwd), stdin);
    term.c_lflag |= ECHO;
    tcsetattr(fileno(stdin), 0, &term);
    printf("\nVasa sifra je: %s\n", passwd);
    return 0;
}
```

Листинг 3.3: Пример измене подешавања терминала.

Функције библиотеке `termios.h` приказане у листингу 3.4 омогућавају контролу брзине слања и примања података на серијским линијама.

```
/* Dobavljanje trenutne brzine primanja podataka */
speed_t cfgetispeed(const struct termios *termios_p);
/* Dobavljanje trenutne brzine slanja podataka */
speed_t cfgetospeed(const struct termios *termios_p);
/* Postavljanje brzine primanja podataka */
int cfsetispeed(struct termios *termios_p, speed_t speed);
/* Postavljanje brzine slanja podataka */
int cfsetospeed(struct termios *termios_p, speed_t speed);
/* Postavljanje brzine primanja i slanja podataka
   na istu vrednost */
int cfsetspeed(struct termios *termios_p, speed_t speed);
```

Листинг 3.4: Контрола брзине примања и слања података.

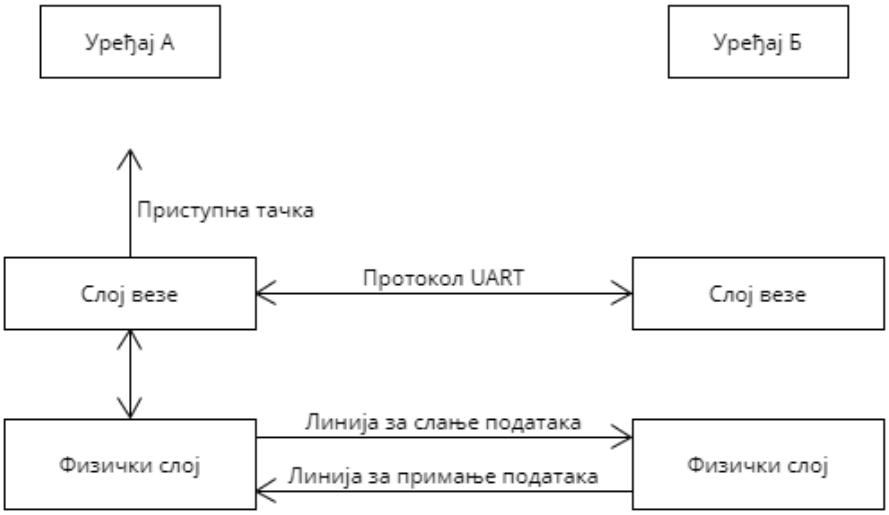
Функције библиотеке `termios.h`, приказане у листингу 3.5, омогућавају контролу преноса података преко серијских линија.

```
/* Zaustavljanje komunikacije tokom zadatog perioda */
int tcsendbreak(int fd, int duration);
/* Zaustavljanje slanja dok se ne isprazni interni
   bafer za slanje */
int tcdrain(int fd);
/* Praznjenje internih bafera terminala */
int tcflush(int fd, int queue_selector);
/* Zaustavljanje ili ponovno pokretanje komunikacije */
int tcflow(int fd, int action);
```

Листинг 3.5: Контрола преноса података.

## 3.4 Класификација према референтном моделу *OSI*

Илустрација припадности протокола *UART* слојевима референтног модела *OSI* приказана је на слици 3.2. Физичком слоју припадају серијске линије које шаљу податке у виду битова, док се протокол *UART* апстрахује у виду менаџера ресурса који се извршава на слоју везе. Менаџер ресурса је задужен за формирање пакета, описаних у поглављу 3.2, као и интеракцију са физичким слојем.




Слика 3.2: протокол *UART* према моделу *OSI*.

Протоколи слоја везе не пружају механизме за ретрансмисију или детекцију и опоравак од грешке. Потребно је имплементирати протокол, односно слојеве вишег нивоа референтног модела *OSI*, који би се бавили наведеним концептима и омогућили поуздану комуникацију.

Један механизам транспортног слоја за ретрансмисију или детекцију и опоравак од грешке представљен је у наредног поглављу, док је предлог једног комплетног протокола представљен је у поглављу 5.

# Глава 4

# Техника клизних прозора

Техника клизних прозора (енг. *sliding window*) представља један од механизама, најчешће транспортног слоја, за ретрансмисију или детекцију и опоравак од грешке током комуникације која се заснива на пакетима [13, 19]. Пакет представља низ битова произвољне, али ипак ограничене, дужине. Техника се користи у ситуацијама када је неопходно осигурати да прималац прима пакете у редоследу у којем су послати. Пример једног протокола који користи наведену технику је протокол *TCP* [2].

Комуникација заснована на пакетима почива на идеји слања података подељених у пакете. Сваки пакет се шаље заједно са додатним битовима који би помогли примаоцу да детектује грешку у случају да примљени пакет није идентичан послатом пакету. Када прималац прими пакет и на основу додатно послатих битова потврди да је примљени пакет идентичан послатом пакету, прималац шаље пакет који потврђује успешан пријем поруке. Такав пакет се најчешће назива потврдни пакет (енг. *acknowledgment*), у даљем тексту *ACK*. У случају да прималац примети неку грешку у примљеном пакету на основу додатно послатих битова, прималац одговара са одричним пакетом (енг. *negative acknowledgment*), у даљем тексту *NACK*.

Пошиљалац који не добија никакав одговор од примаоца не може знати да ли је прималац заправо примио пакет или се пакет изгубио или оштетио током транспорта. Такође, одговор примаоца се може изгубити или оштетити током транспорта, те би у том случају пошиљалац послао пакет поново. У том случају, прималац мора имати механизме за октривање ретрансмисије пакета који је већ примио раније, те да исти пакет игнорише и поново пошаље *ACK*.

Пример једног протокола, односно његове варијанте, који имплементира

комуникацију засновану на пакетима је *ARQ* (енг. *automatic repeat request protocol*) [5] . Пошиљалац након сваког послатог пакета чека потврду успешног пријема пакета, односно *ACK*, пре него што пошаље нови пакет. На тај начин осигурава се да је редослед пријема пакета исправан. Оваквим начином имплементације комуникације засноване на пакетима не може се искористити максимална пропусност (енг. *bandwidth*) медијума за комуникацију, јер се бар половина времена извршавања потроши на чекање *ACK* пакета.

Један начин решавања претходно наведеног проблема искоришћености максималне пропусности представља техника клизних прозора која је описана у наставку.

## 4.1 Основна идеја

Техника клизних прозора омогућава слање унапред задатог броја пакета без чекања на потврду о пријему. Задати број пакета који се може послати без чекања на потврду о пријему се назива прозор (енг. *window*).

Сваком послатом пакету додају се додатни битови, који представљају редни број пакета. Прималац пакета користи редни број пакета како би обрађивао пакете у редоследу у којем су послати, те евентуално елиминисао дупликате или детектовао пропуштен пакет. За сваки успешно примљен пакет шаље се *ACK* пакет који садржи информацију о редном броју пакета за који се потврђује пријем. У случају пријема пакета који не одговара редоследу пријему пакета, шаље се *NACK* као одговор. У случају пријема *ACK* пакета, прозор се помера даље, клиза (енг. *slide*), по низу пакета спремних за слање и шаље следећи пакет или више њих у зависности од дужине прозора. Пошиљалац чека *ACK* пакет за сваку поруку задати период времена. У случају да *ACK* пакет не стигне на време, пакет се шаље поново. Процедура поновног слања најчешће се понаља ограничен број пута.

Дужина прозора зависи од пропусности медијума за комуникацију и најчешће се бира на основу експеримената са различитом дужином прозора. У случају да је прозор прекратак, пропусност ће бити мала. Док у случају предугачког прозора, прималац можда неће бити у могућности да обради све поруке у задатом року за слање *ACK* пакета.

## 4.2 Синхронизација пошиљаоца и примаоца

Синхронизација између пошиљаоца и примаоца се остварује помоћу чувања информације о редном броју пакета који треба да се пошаље, односно прими, следећи. $n_t$ представља редни број пакета који треба да се пошаље следећи, док $n_r$ представља редни број пакета који треба да се прими следећи. Оба броја, $n_t$ и $n_r$, монотоно се увећавају током извршавања.

Прималац чува додатну информацију о највећем редном броју пакета примљеном током извршавања, који се означава са $n_s$. Уз помоћ информације о највећем радном број пакета примљеном током извршавања, може се извести закључак:

1. сви пакети редног броја мањег од $n_r$ су успешно примљени,
2. ниједан пакет редног броја већег од $n_s$ није примљен и
3. примљени су неки пакети редних бројева између $n_r$ и $n_s$.

Када прималац прими поруку, провера се да ли је редни број примљене поруке $n_r$. Уколико јесте, $n_r$ се увећава за један и шаље се $ACK$ пакет који садржи нову вредност $n_r$.

Пошиљалац чува информацију о највећем редном броју пакета примљеном у оквиру $ACK$ пакета, који се означава са $n_a$. На основу $n_a$ пошиљалац зна да су сви пакети редних бројева мањег од $n_a$ успешно примљени.

Дужина клизних прозора пошиљаоца и примаоца се означава са $w_t$ и $w_r$. Дужина клизних прозора може бити различита за пошиљаоца и примаоца, али се углавном користи клизни прозор исте дужине. При чему прозор мора бити дужи од 0.

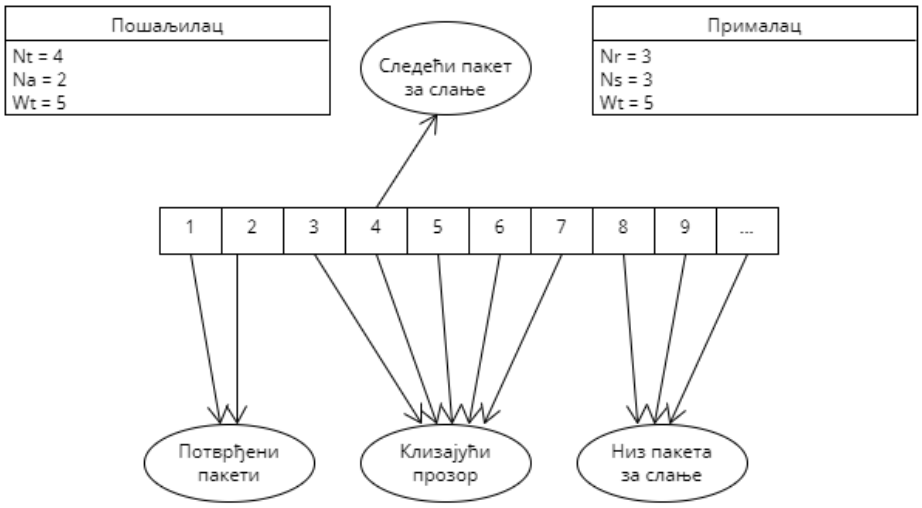
Информације о редним бројевима послатих и примљених пакета морају увек задовољавати релацију $n_a \leq n_r \leq n_s \leq n_t \leq n_a + w_t$, односно:

1. $n_a \leq n_r$ — највећи редни број $ACK$ пакета не сме бити већи од редног броја пакета који прималац очекује да прими као следећи пакет,
2. $n_r \leq n_s$ — редни број пакета који који прималац очекује да прими следећи не сме бити већи од највећег редног броја пакета примљеног током извршавања,

3. $n_s \leq n_t$ — највећи редни број пакета примљен на стани примаоца не сме бити већи од највећег редног броја послатог пакета,

4. $n_t \leq n_a + w_t$ — највећи редни број послатог пакета је ограничен збиром највећим редним бројем $ACK$ пакета и дужине прозора за слање.

Илустрација претходно наведене релације приказана је на слици 4.1 [13, 19].




Слика 4.1: Синхронизација пошиљаоца и примаоца.

## 4.3 Протокол *Врати-Се-N*

*Врати-Се-N* (енг. *Go-Back-N*) представља унапређену верзију протокола $ARQ$ описаног у уводу 4. поглавља. Унапређење се огледа у коришћењу технике клизних прозора. Дужина клизног прозора примаоца увек је 1, док дужина клизног прозора пошиљаоца зависи од пропусности медијума за комуникацију, као и брзине обраде примљених пакета на страни примаоца.

Прималац чува информацију о редном броју пакета који треба да се прими следећи. За сваки пакет који прими, прималац проверава да ли редни број пакета одговара редном броју пакета који се очекује као следећи и, уколико

пакет нема оштећења током транспорта и редни број одговара следећем очекиваном, прослеђује на даљу обраду уз слање $ACK$ пакета пошиљаоцу.

У случају да редни број пакета одговара редном броју пакета који се очекује следећи, али су детектована оштећења пакета током транспорта тада се шаље $NACK$ пакет пошиљаоцу као индикација да је пакет оштећен. Уколико радни број не одговара редном броју пакета који се очекује следећи, порука се игнорише.

Пошиљалац шаље пакете у оквиру клизног прозора према унапред задатом ритму и за сваки пакет очекује $ACK$ пакет у унапред задатом временском интервалу. Након што пошаље све пакете у оквиру клизног прозора, пошиљалац се враћа на почетак прозора и чека $ACK$ пакет за први послати пакет из клизног прозора. Уколико $ACK$ пакет стигне у предвиђеном временском интервалу, пошиљалац помера клизни прозор за једну позицију. Након померања клизног прозора отвара се простор за слање додатног пакета, те пошиљалац шаље додатни пакет. Након слања додатног пакета пошиљалац се поново враћа на почекат прозора како би чекао следећи $ACK$ пакет. Уколико $ACK$ пакет не стигне у предвиђеном временском интервалу или се загуби или оштети током транспорта, пошиљалац шаље све пакете у оквиру клизног прозора поново.

Мана оваквог приступа је да се неки пакети могу послати више пута, чак и ако је прималац успешно примио пакет, услед претходно описаних проблема који се могу догодити током транспорта. Један начин за превазилажење проблема који се могу догодити током транспорта описан је наредном поглављу.

# Глава 5

# Протокол за директну комуникацију *YAP3*

Како *UART* комуникација припада слоју везе референтног модела *OSI*, неопходно је апстраховати овај вид комуникације и пружити корисницима поуздан канал за комуникацију (енг. *communication channel*). *Још један протокол за директну комуникацију*, односно *YAP3* развијен је у оквиру овог рада и представља протокол за директну комуникацију између уређаја са уграђеним рачунаром. *YAP3* имплементира слојеве апликације, презентације, транспорта и слој везе референтног модела *OSI* у оквиру једног процеса. Да би обезбедио поуздан канал за комуникацију протокол *YAP3* имплементира различите механизме ретрансмисије, детекције и опоравка од грешака током комуникације. Крајњем кориснику се не излаже ни један деталь имплементације, већ само интерфејс за приступ каналу за комуникацију.

Протокол *YAP3* је имплементиран у програмским језицима *C* и *C++*. Имплементација, заједно са тестовима јединица кôда, може се пронаћи на сервису *GitHub* на адреси `https://github.com/lazarsrkic/yap3`.

Тренутна имплементација предвиђа коришћење *UART*-а као медијума за комуникацију, али модуларност која произилази из имплементације према референтном моделу *OSI* оставља могућност да се протокол прошири тако да користи и неке друге медијуме за серијску комуникацију. Детаљи дизајна, имплементације и коришћених механизама ретрансмисије, детекције и опоравка од грешака описани су у наставку текста.

## 5.1 Опис архитектуре

Софтверска архитектура протокола може се поделити у три логичке целине.

**Апстраховање приступа медијуму за комуникацију** имплементира слој везе података протокола *YAP3*. Слој везе података енкапсулира иницијализацију, подешавање и размену порука преко медијума.

**Контрола тока података** је имплементирана у оквиру транспортног слоја. Контрола тока података се огледа у имплементираном проширењу протокола *Врати-Се-N* како би протокол обезбедио робустнији проток порука. Детаљи проширења описани су у секцији 5.4.

**Интеракција са корисницима** је имплементирана у оквиру слоја апликације. Слој апликације пружа корисницима интерфејс за размену порука, при чему су детаљи о медијуму за комуникацију скривени од корисника.

Енкапсулација слојева протокола имплементирана је помоћу класа програмског језика *C++*. Сваки слој протокола *YAP3* дефинисан је као засебна класа у оквиру заглавља који одговара тој класи. Класе `Datalink`, `Transport`, `Presentation`, `Application` дефинисане у заглављима `datalink.h`, `transport.h`, `presentation.h`, `application.h` представљају слој везе, транспортни слој, презентациони слој и слој апликације, респективно. Класа сваког слоја садржи референце на инстанце класа које имплементирају слојеве испод и изнад те класе. На тај начин се осигурава да само суседни слојеви размењују поруке.

Класа `Protocol`, дефинисана у оквиру заглављу `protocol.h`, садржи инстанце класа свих слојева и представља класу која апстрахује протокол *YAP3*. Извршавање протокола *YAP3* паралелизовано је у оквиру две нити (енг. *threads*). Покретање и контрола извршавања нити имплементирана је у оквиру класе `Protocol`. Једна нит извршавања задужена је за интеракцију са корисницима протокола, док је друга нит извршавања задужена за одржавање комуникације између два уграђена рачунара. Обе нити извршавају своје задатке (енг. *tasks*) периодично, према унапред задатом временском интервалу, како би кашњење слања и примања порука било минимално.

Временски интервали као и остала глобална подешавања протокола дефинисана је помоћу макроа у оквиру заглавља `configuration.h`. У оквиру

фолдера `utils` дефинисане су неопходне помоћне класе и структуре података које енкапсулирају употребу различитих алгоритама. Коришћене помоћне структуре и класе биће описане у наставку текста.

## 5.2 Формат порука

Сваки слој протокола проширује поруку заглављем (енг. *header*) дефинисаним за тај слој. Заглавље садржи додатне информације на основу којих прималац може закључити коју операцију је неопходно применити на примљену поруку. На пример, заглавље може да садржи информацију о типу поруке која се шаље, редни број поруке или информацију о типу коришћеног шифровање.

Протокол *YAP3* третира поруку, заједно са заглављем, као низ бајтова, при чему на сваком слоју очекује заглавље на унапред задатој позицији у низу бајтова. Уколико заглавље није на предвиђеној позицији, порука се игнорише и пријављује се грешка. Сваки слој протокола дефинише минималну и максималну дужину поруке заједно са заглављем.

Формат порука слоја апликације приказан је на слици 5.1. Дефиниција формата поруке слоја апликације налази се у оквиру заглавља `application.h`. Заглавље поруке се састоји од два бајта. Заглавље идентификације уписује корисник заједно са поруком која се шаље и представља информацију о пошиљаоцу и примаоцу поруке. Четири бита најмање тежине представљају идентификациони број примаоца, док четири бита највеће тежине представљају идентификациони број пошиљаоца поруке. Тренутни формат поруке предвиђа максимално шеснаест различитих пошиљалаца, као и максимално шеснаест различитих прималаца поруке.

| 1 бајт | 1 бајт | Од 0 до 100 бајтова |
|---|---|---|
| Контролно заглавље слоја апликације | Заглавље идентификације | Порука |

Слика 5.1: Формат поруке слоја апликације.

Четири бита најмање тежине контролног заглавља слоја апликације представљају параметар подешавања протокола *YAP3* и дефинишу којом методом

је потребно шифровати поруке које протокол шаље преко медијума. Слој апликације контролише метод шифровања да би се омогућио различит тип шифровања за различите пошиљаоце, односно примаоце. У случају да сви пошиљаоци и примаоци користе исти метод шифровања, тада би метод шифровања контролисао презентациони слој. Четири бита највеће тежине тренутно нису у употреби, те је могуће проширити протокол додатним подешавајућим параметрима.

| 1 бајт | Од 0 до 128 бајтова |
|---|---|
| Контролно заглавље слоја апликације | Порука криптована стандардом *AES128* |

Слика 5.2: Формат поруке презентационог слоја који користи стандард *AES128* за криптовање.

На слици 5.2 приказан је један формат порука презенатационог слоја. Дефиниција наведеног формата налази се у оквиру заглавља `presentation.h`. На основу метода шифровања уписаног на слоју апликације, презентациони слој би шифровао поруку уписаном методом. У случају криптовања према стандарду *AES128*, порука би се допуњавала додатним нулама док се не постигне поравнање на шеснаест бајтова.

Како шифровање и дешифровање нису у главном фокусу ове тезе, тренутна имплементација презентационог слоја само прослеђује поруку са слоја апликације ка транспортном слоју. Имплементација шифровања и дешифровања представља једно од могућих унапређења протокола.

| 2 бајта | 1 бајт | 1 бајт | Од 0 до 128 бајтова |
|---|---|---|---|
| Заглавље цикличне провере редундансе | Контролно заглавље транспортног слоја | Контролно заглавље слоја апликације | Порука криптована стандардом *AES128* |

Слика 5.3: Формат поруке транспортног слоја.

Формат поруке транспортног слоја приказан на слици 5.3, дефинисан је у оквиру заглавља `transport.h`. Транспортни слој проширује примљену поруку контролним заглављем транспортног слоја који садржи информације

неопходне за контролу трансмисије дужине једног бајта, као и заглављем цикличне провере редундансе (енг. *cyclic redundancy check* или *CRC*) дужине два бајта.

Четири бита најмање тежине контролног заглавља транспортног слоја представљају редни број поруке, док четири бита највеће тежине представљају тип поруке. Наведене информације су неопходне за имплементацију проширења протокола *Врати-Се-N* имплементираног у оквиру транспортног слоја.

Заглавља цикличне провере редундансе представљају повратну вредност алгоритма *CRC16* примењеног на преостали део поруке након заглавља цикличне провере редундансе. Први бајт заглавља цикличне провере редундансе садржи осам битова највеће тежине повратне вредности алгоритма *CRC16*, док су преосталих осам битова најмање тежине уписани у други бајт заглавља. Заглавље цикличне провере редундансе се користи као механизам за детекцију оштећења порука током транспорта. Имплементација алгоритма *CRC16* налази се у оквиру заглавља `crc16.h`. Заглавље је могуће проширити на четири бајта дужине у случају употребе *CRC32* или осам бајтова дужине у случају *CRC64*. Алгоритама цикличне провере редундансе апстрахован је у оквиру помоћне класе `crc16`, заглавља `crc16.h`.

Слој везе података проширује поруку описану на транспортном слоју додатним заглављима почетка и заустављања дефинисаним у оквиру заглавља `datalink.h`. Заглавље почетка представља сигнал за почетак *YAP3* поруке и умеће се на почетак поруке. Заглавље заустављања представља сигнал за крај *YAP3* поруке и умеће се на крај поруке. Формат поруке слоја везе приказан је на слици 5.4.

| 1 бајт | 2 бајта | 1 бајт | 1 бајт | Од 0 до 128 бајтова | 1 бајт |
|---|---|---|---|---|---|
| Заглавље почетка | Заглавље цикличне провере редундансе | Контролно заглавље транспорт-ног слоја | Контролно заглавље слоја апликације | Порука криптована стандардом *AES128* | Заглавље заустављања |

Слика 5.4: Формат порука слоја везе.

## 5.3 *UART* као медијум за комуникацију

Приступ протоколу *UART*, самим тим и медијуму за комуникацију, апстрахован је од стране менаџера ресурса оперативног система. Приступ се реализује кроз приступну тачку, односно терминал, за повезани уређај. Неопходно је знати који терминал одговара уређају са којим је потребно успоставити комуникацију како би се користио протокол *YAP3*.

Протокол *YAP3* апстрахује *UART* као медијум за комуникацију у оквиру класе `Datalink` која представља слој везе. Да би се наведена инстанца класе креирала, неопходно је проследити путању до терминала и брзину слања као аргументе конструктора. Протокол *YAP3* захтева путању до терминала и жељену брзину размене података као аргументе командне линије. Аргументи командне линије се даље прослеђују слоју везе података. Слој везе података успоставља везу за комуникацију између протокола *YAP3* и терминала помоћу системског позива `open`. Као први аргумент системског позива `open` прослеђује се задата путања.

Након успостављања иницијалне конекције, слој везе података подешава терминал помоћу операција дефинисаних у оквиру заглавља `termios.h`. Слој везе података потокола *YAP3* подешава терминал на следећи начин:

1. Брзина слања и примања података се поставља на основу аргумента командне линије.

2. Празне се интерни бафери за слање и примање података менаџера ресурса како слој везе не би руковао подацима уписаним пре покретања протокола.

3. Искључује се задржавање податка у оквиру интерних бафера менаџера ресурса како би протокол слоја везе имао минимално кашњење приликом размене података[1].

4. Сви улазни и излазни подаци се третирају као сирови низ бајтова.

5. Сва постављена подешавања постају активна пре него што извршавање интерних операција протокола почне.

[1]Кашњење се може додатно умањити уколико се менџер ресурса извршава у реалном времену. У оквиру оперативног система *Linux*, извршавање у реалном времену се остварује помоћу извршавања у оквиру посебне нити кернела задужене само за одређени менаџер ресурса.

Класа `Datalink`, односно слој везе података, садржи инстанцу помоћне класе `Serial`, заглавља `serial.h`, у оквиру које је апстрахован приступ и подешавање терминала.

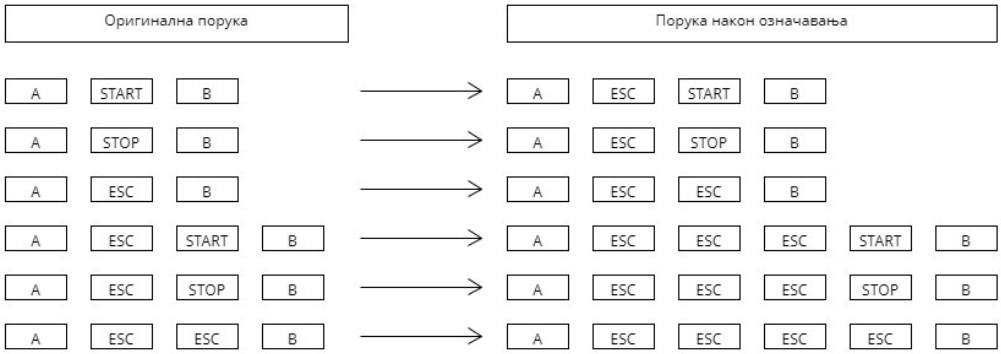
Када се иницијална веза успостави, а затим и постави жељено подешавање терминала, операције слања и примања порука користећи *UART* као медијум за комуникацију остварују се помоћу системских позива `write` и `read`. Управо операције слања и примања порука, у виду низа бајтова, представљају интерфејс слоја везе података према транспортном слоју.

Како протокол *UART* шаље податке серијски, неопходно је означити почетак и крај поруке протокола *YAP3*. Слој везе протокола *YAP3* имплементира наведено означавање помоћу додатних заглавља почетка и заустављања који се умећу на почетак и крај поруке. По пријему поруке, у виду низа бајтова, од стране протокола *UART*, слој везе података покушава да пронађе заглавље почетка претражујући поруку секвенцијално бајт по бајт. Након што пронађе заглавље почетка, слој везе података даље тражи заглавље заустављања и када га пронађе прослеђује низ бајтова, почевши од позиције заглавља почетка до позиције заглавља заустављања, транспортном слоју. Процедура се понавља све док се не обради комплетан низ бајтова. Уколико порука не садржи заглавља почетка и заустављања, порука се игнорише.

Проблем који се јавља са оваквим приступом је ситуација у којој порука примљена од стране транспортног слоја садржи бајт који је једнак заглављу почетка или заустављања. У том случају претходно наведени механизам не би могао да разликује између заглавља почетка и заустављања уметнутом од стране слоја везе и бајта који има вредности заглавља почетка или заустављања примљеног као део поруке транспортног слоја. Како би превазишао претходно наведени проблем, слој везе протокола *YAP3* уводи додатно означавање бајтова поруке који су једнаки заглављу почетка или заустављања слоја везе. Означавање је имплементирано у оквиру класе `Datalink` увођењем специјалног бајта означавања, у даљем тексту *ESC*.

Пре него што пошаље поруку на медијум, слој везе протокола *YAP3* секвенцијално претражује поруку примљену од стране транспортног слоја и уколико пронађе бајт који је једнак заглављу почетка или заустављања или бајту означавања примењује се операција означавања. Пример означавања приказан је на слици 5.5, при чему ниске *START* и *STOP* представљају бајтове поруке примљене од стране транспортног слоја чија је вредност једнака

заглављу почетка или заустављања.




Слика 5.5: Означавање заглавља почетка и заустављања у оквиру порука.

У случају пријема поруке, примењује се операција уклањања означених бајтова како би се реконструисала оригинална порука.

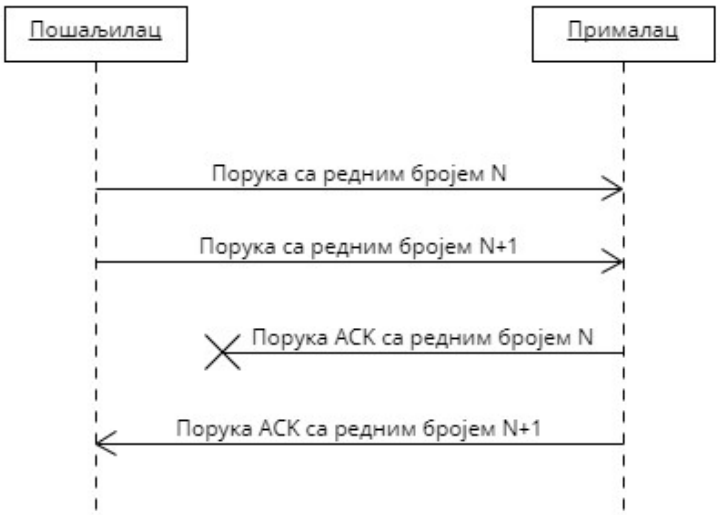
## 5.4 Проширења протокола *Врати-Се-N*

Контрола тока података представља основни задатак протокола транспортног слоја. Транспортни слој протокола *YAP3* имплементира контролу тока података проширењем протокола *Врати-Се-N*. Проширење подразумева увођење додатних предефинисаних типова порука протокола како би се одржавала комуникација између пошиљаоца и примаоца, чак и у случају одсуства корисничких порука. Поред стандардних типова порука, као што су корисничке поруке, *ACK* и *NACK*, транспортни слој уводи поруку за поновну иницијализацију конекције, *RESET*, и поруку за одржавање везе у случају одсуства корисничких порука, *KEEPALIVE*. Дефиниција сваке од порука, као и интерна имплементација транспортног слоја може се пронаћи у оквиру класе `Transport`, дефинисане у оквиру заглавља `transport.h`.

Стандардна имплементације протокола *Врати-Се-N* подразумева поновно слање целог клизног прозора у случају изгубљене *ACK* поруке током транспорта. У случају медијума који је склон губитку порука током транспорта, претходно описано понашање може узроковати вишеструко слање порука чак и када је прималац успешно примио поруку.

Како би избегли поновно слање пакета који је можда успешно примљен на страни примаоца, али се одговор загубио током транспорта, транспортни

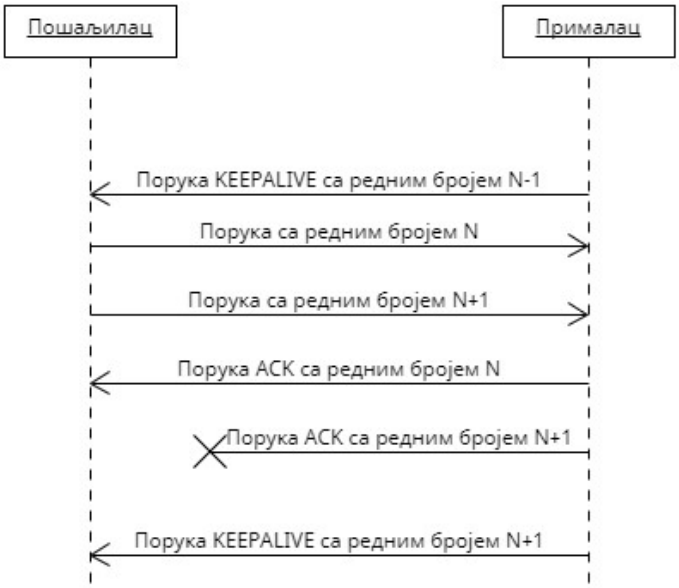
слој протокола *YAP3* прихвата *ACK* чак и ако је редни број *ACK* поруке већи од редног броја прве поруке клизног прозора. Претпоставка на којој се заснива оваква имплементација подразумева да прималац чува информацију о редном броју последњег примљеног пакета и да у случају да *ACK* порука садржи редни број $n+1$, пошиљалац који чека одговор може закључити да је порука $n$, такође, успешно примљена. Стога се клизни прозор пошиљаоца помера на позицију редног броја последње примљене *ACK* поруке. Илустрација претходно наведене ситуације приказана је на слици 5.6.




Слика 5.6: Опоравак од губитка поруке *ACK*.

Транспортни слој протокола *YAP3* имплементира додатни механизам опоравка од грешке помоћу поруке *KEEPALIVE*. Порука *KEEPALIVE* шаље се у случају одсуства размене порука у унапред задатаком временском интервалу и садржи редни број последње успешно примљене поруке. Чињеница да се *KEEPALIVE* порука шаље у одсуству размене порука може се такође искористити за опоравака од губитка *ACK* поруке током транспорта. У случају пријема *KEEPALIVE* поруке са редним бројем који је једнак или већи од редног броја поруке за коју се очекује *ACK* порука, пошиљалац може закључити да је прималац успешно примио поруку за коју се тренутно чека потврда,

те се клизни прозор пошиљаоца помера на позицију редног броја последње примљене *KEEPALIVE* поруке. Илустрација претходно наведене ситуације приказана је на слици 5.7.

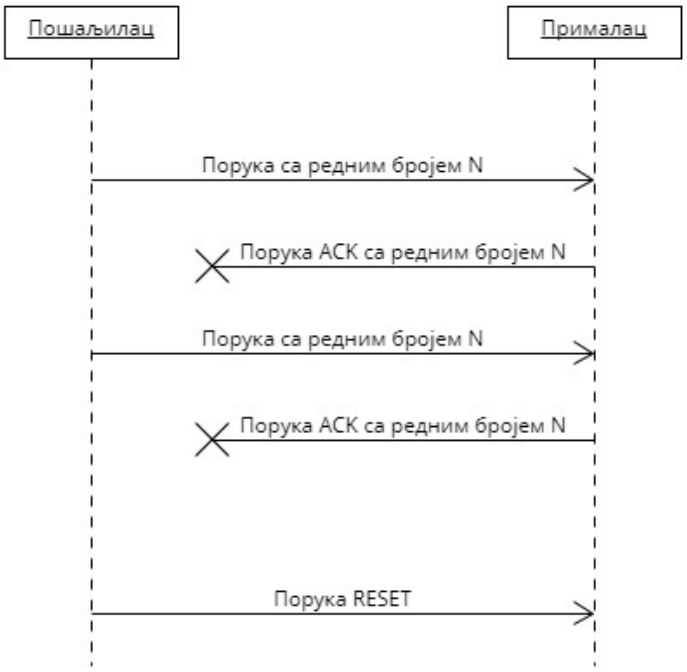



Слика 5.7: Опоравак од губитка поруке *ACK* помоћу поруке *KEEPALIVE*.

У случају одсуства корисничких порука, порука *KEEPALIVE* се шаље периодично и користи као механизам одржавања комуникације између пошиљаоца и примаоца, односно два уређаја.

Уколико претходно наведени механизми не допринесу опоравку од грешке, порука се шаље поново. Након три неуспешна чекања поруке *ACK* за послати пакет, пошиљалац шаље поруку *RESET*. Порука *RESET* представља сигнал за поновну иницијализацију везе између пошиљаоца и примаоца. Иницијализација везе подразумева ресетовање свих интерних информација протокола, као и клизног прозора, на почетне вредности. Илустрација претходно наведене ситуације приказана је на слици 5.8.

Транспортни слој протокола *YAP3* имплементира наведено проширење помоћу помоћне структуре података `SlidingWindow`, дефинисе у оквиру заглавља `slidingwindow.h`. Помоћна структура података `SlidingWindow` приказана је у листингу 5.1.




Слика 5.8: Поновна иницијализација.

```
struct SlidingWindow {
    /* Konstruktor */
    SlidingWindow(std::uint8_t const max_msg_number)
    : m_max_msg_number(max_msg_number) {
        last_acked = m_max_msg_number - 1;
        to_be_acked = 0;
        offset_to_send = 0;
        retry_count = 0;
    }
    /* Maksimalni redni broj poruke */
    std::uint8_t const m_max_msg_number;
    /* Redni broj poslednje uspesno poslate poruke */
    std::uint8_t last_acked;
    /* Ocekivani redni broj sledece primljene poruke */
    std::uint8_t to_be_acked;
    /* Preostali prostor klizajuceg prozora */
    std::uint8_t offset_to_send;
```

```
    /* Broj retransmisija */
    std::uint8_t retry_count;
};
```

Листинг 5.1: Клизни прозор транспортног слоја протокола *YAP3*.

Поља `to_be_acked` и `last_acked` могу имати вредности у распону од нула до петнаест, јер се радни број поруке записан у контролном заглављу транспортног слоја записује помоћу четири бита. Како би се избегло прекорачење (енг. *overflow*), након сваке промене вредности неког од ова два поља, нова вредност представља модуо при дељењу са шеснаест.

Током извршавања, транспортни слој одржава поља структуре на следећи начин:

1. Редни број поруке која се шаље израчунава се на основу поља `offset_to_send`. Након сваке послате поруке поље `offset_to_send` се инкрементира. Протокол шаље поруке све док је `offset_to_send` мањи од дужине клизног прозора.

2. Порука *KEEPALIVE* шаље се са редним бројем поруке који је једнак вредности поља `last_acked`.

3. За сваки примљену *ACK* или *KEEPALIVE* поруку, провера се да ли је редни број примљене поруке мањи од броја `((to_be_acked+offset_to_send) % 16)`. Уколико јесте, нова вредност поља `last_acked` постаје редни број примљене поруке увећан за један. У супротном се порука игнорише.

4. Уколико *ACK* или *KEEPALIVE* не стигну у унапред задатом интервалу или протокол прими поруку *NACK*, клизни прозор се шаље поново. Поље `retry_count` се увећава за свако поновно слање поруке.

5. Уколико `retry_count` достигне вредност три, протокол шаље поруку *RESET* и структура `SlidingWindow` се иницијализује на почетне вредности.

6. У случају пријема поруке *RESET* структура `SlidingWindow` се иницијализује на почетне вредности.

Медијуми за серијску комуникацију су подложи оштећењима и губицима порука услед спољашњих утицаја, стога је неопходно осигурати поуздан проток података претходно наведеним или неким другим сличним механизмима.

## 5.5 Кориснички интерфејс

Како би корисници могли да користе протокол, неопходно је пружити кориснички интерфејс. Слој апликације протокола *YAP3* имплементира интерфејс за приступ коришћењем редова порука стандарда *POSIX* (енг. *POSIX message queues*) [9].

Приступ протоколу *YAP3* апстрахован је у оквиру класе `Yap3Client` заглавља `yap3client.h`. Класа `Yap3Client` остварује везу са слојем апликације протокола *YAP3*. Помоћу остварене везе слој апликације и инстанца класе `Yap3Client` размењују поруке. Интерфејс класе `Yap3Client` приказан је у листингу 5.2.

```
class Yap3Client {
    /* Konstruktor */
    Yap3Client(const std::uint8_t id) noexcept;
    /* Interfejs za slanje poruka */
    bool timed_send(std::vector<std::uint8_t> const& data,
                    std::chrono::milli const& timeout)
                    const noexcept;
    bool send(std::vector<std::uint8_t> const& data) const noexcept;
    /* Interfejs za primanje poruka */
    ssize_t timed_receive(std::vector<std::uint8_t>& data,
                    std::chrono::milli const& timeout)
                    const noexcept;
    ssize_t receive(std::vector<std::uint8_t>& data) const noexcept;
}
```

Листинг 5.2: Интерфејс класе `Yap3Client`.

Конструктор класе `Yap3Client` захтева идентификациони број пошиљаоца као аргумент. Идентификациони број пошиљаоца омогућава имплементацију повратне спреге протокола *YAP3* и корисничког програма. Имплементација повратне спреге протокола *YAP3* и корисничког програма омогућава слање и примање података у оба смера спреге. Један од случајева употребе (енг. *use case*) овакве спреге омогућава протоколу *YAP3* да обавести пошиљаоца да ли је прималац примио поруку. Уколико би корисник директно користио протокол *UART* или неки други протокол слоја везе, кориснички програм не би имао могућност да добије потврду о пријему поруке. Пример креирања везе између корисничког програма и протокола *YAP3* помоћу класе `Yap3Client` приказан је у листингу 5.3.

```
#include "yap3client.h"

/* Identifikacioni broj posaljioca */
constexpr auto sender_id{1U};
/* Upostavljanje veze */
const yap3::Yap3Client yap3client{sender_id};
```

Листинг 5.3: Приступ протоколу *YAP3* помоћу инстанце класе `Yap3Client`.

Слој апликације креира интерни ред порука (енг. *message queue*) у који инстанце класе `Yap3Client` уписују, односно шаљу поруке. Додатно, слој апликације креира ред порука за сваког корисника. Корисник помоћу инстанце класе `Yap3Client` може приступити свом реду порука и прочитати поруке које шаље слој апликације.

```
/* Identifikacioni broj posaljioca */
constexpr auto sender_id{1U};
/* Uspostavljanje veze */
const yap3::Yap3Client yap3client{sender_id};
/* Identifikacioni broj primaoca */
constexpr auto receiver_id{2U};
/* Kreiranje identifikacionog zaglavlja */
constexpr auto id_header_1_2{(sender_id << 4) | (receiver_id & 0xf)};

/* Slanje poruke */
if(yap3client.send({id_header_1_2, 0x01, 0x02, 0x03})) {
    std::cout << "Uspesno poslata poruka\n";
} else {
    /* Rukovanje greskom */
}

/* Kreiranje bafera za poruku */
std::vector<std::uint8_t> buffer;
/* Prijem poruke */
if(yap3client.receive(buffer)) {
    std::cout << "Uspesno primljena poruka\n";
} else {
    /* Rukovanje greskom */
}
```

Листинг 5.4: Размена порука помоћу инстанце класе `Yap3Client`.

Приликом слања порука, неопходно је навести идентификациони број примаоца поруке у оквиру идентификационог заглавља слоја апликације, како би инстанца протокола *YAP3* на страни примаоца знала коме је порука намењена. Такође, по пријему поруке, на основу информација из идентификационог заглавља слоја апликације, прималац може закључити ко је пошиљалац поруке. Наведени механизам пружа наивни вид аутентификације између пошиљаоца и примаоца. Пример слања и примања поруке помоћу инстанце класе `Yap3Client` приказан је у листингу 5.4.

Тренутно, протокол *YAP3* све корисничке поруке третира на исти начин. Једно од унапређења слоја везе протокола *YAP3* може бити увођење категорија порука, при чему неке категорије могу имати већи приоритет у односу на остале.

## 5.6 Употреба протокола *YAP3*

Домени употребе протокола *YAP3* могу бити разнолики. Превасходно је намењен уређајима са уграђеним рачунаром, али се може користити и за потребе комуникације између корисничких рачунара и разних микро-контролера.

Модерне архитектуре хардвера немењеног уређајима са уграђеним рачунаром предлажу употребу више уграђених рачунара у оквиру једног уређаја. У оквиру такве архитектуре, поред Етернета, на који су најчешће сви уграђени рачунари повезани, *UART* представља један од често коришћених механизама повезивања уграђених рачунара. Протокол *YAP3* најчешће налази примену управо у оквиру уређаја са таквом архитектуром хардвера. Протокол се може искористити за контролу животног циклуса (енг. *life cycle*) између уграђених рачунара, размену информација о тренутној температури или размену неких других информација.

Како би уграђени рачунар користио протокол, неопходно је изворни кôд протокола превести за архитектуру циљаног уграђеног рачунара. За превођење протокола *YAP3* коришћен је систем за превођење кôда (енг. *build system*) *Bazel*[2]. Помоћу команде приказане у листингу 5.5 изворни кôд протокола *YAP3* преводи се за архитектуру `x86_64`. Подршка превођења за друге

[2]Систем за превођење кôда *Bazel* се може инсталирати помоћу команде `apt install bazel` у оквиру оперативног система *Linux*.

архитектуре представља једно од унапређења протокола и може се остварити помоћу система за превођење кôда без измена изворног кôда протокола *YAP3*.

```
bazel build //protocol:yap3
```

Листинг 5.5: Превођење протокола за архитектуру `x86_64`.

Добијени извршни кôд, може се директно покренути помоћу команде приказане у листингу 5.6.

```
yap3 /dev/ser1 115200
```

Листинг 5.6: Пример покретања протокола. Аргументи командне линије `/dev/ser1` и `115200`, представљају путању до терминал и жељену брзину преноса података.

## 5.7 Тестирање и евалуација резултата

Тестови јединица кôда не могу покрити све случајеве употребе протокола *YAP3*, те не могу пружити довољно информација о коректности извршавања протокола у оквиру уграђених рачунара. Стога, како би евалуирали протокол *YAP3*, неопходно је тестирати протокол помоћу тестова интеграције у окружењу што сличнијем окружењу уграђених рачунара.

### Симулација окружења уграђених рачунара

Како је развој софтвера за уграђене рачунаре непрактичан уколико се изводи директно на уграђеном рачунару, за потребе развоја и тестирања протокола *YAP3* користи се окружење *Docker*. Окружење *Docker* омогућава симулирање окружења уграђеног рачунара.

```
FROM ubuntu:20.04
RUN apt update && \
    apt install -y --no-install-recommends \
    sudo openssh-server g++
RUN mkdir /var/run/sshd
RUN echo 'root:root' | chpasswd
RUN sed -i 's/#PermitRootLogin prohibit-password/ \
    PermitRootLogin yes/' /etc/ssh/sshd_config
ENTRYPOINT service ssh start && bash
```

Листинг 5.7: Минимално *Docker* окружење за тестирање протокола *YAP3*.

Пример милималног окружења *Docker* у оквиру ког је могуће тестирати протокол *YAP3* приказан је у листингу 5.7. Окружење се може превести помоћу команде приказане у листингу 5.8.

```
DOCKER_BUILDKIT=1 docker build -t yap3-test .
```

Листинг 5.8: Превођење *Docker* окружења.

За потребе симулације серијске комуникације неопходно је повезати две инстанце окружења *Docker*. Окружења се могу повезати помоћу виртуелне цеви (енг. *pipe*) за серијску комуникацију креиране уз помоћ команде `socat`. Пример позива команде `socat` приказан је у листингу 5.9.

```
socat -d -d pty,raw,echo=0 pty,raw,echo=0
2022/08/07 23:55:20 socat[1203849] N PTY is /dev/pts/N
2022/08/07 23:55:20 socat[1203849] N PTY is /dev/pts/M
```

Листинг 5.9: Креирање виртуелне цеви за серијску комуникацију.

Команда `socat` креира два псеудо терминала`/dev/pts/N` и `/dev/pts/M`, при чему су `N` и `M` цели бројеви, који представљају крајеве виртуелне цеви за комуникацију. Како би повезали две инстанце окружења *Docker* потребно је свакој инстанци проследити путању до једног псеудо терминала. Пример покретања два повезана окружења *Docker*, који представљају два уграђена рачунара, заједно са извршним кôдом протокола *YAP3* приказан је на у листинзима 5.10 и 5.11.

```
docker run --rm -v /putanja/do/izvrsnog/koda/yap3:/usr/sbin/yap3 \
  -v /dev/pts/N:/dev/ser1 --name peer1 \
  -it yap3-test:latest
```

Листинг 5.10: Покретање првог виртуалног уграђеног рачунара.

```
docker run --rm -v /putanja/do/izvrsnog/koda/yap3:/usr/sbin/yap3 \
  -v /dev/pts/M:/dev/ser1 --name peer2 \
  -it yap3-test:latest
```

Листинг 5.11: Покретање другог виртуалног уграђеног рачунара.

У оквиру покренутих инстанци виртуелних уграђених рачунара, извршни кôд протокола биће доступан на путањи `/usr/sbin/yap3`. Протокол се надаље може покретати као што је приказано у листингу 5.6. Протокол је неопходно

покренути као позадински процес (енг. *daemon*) пре покретања процеса који ће користити протокол. На тај начин се обезбеђује да ни једна порука није пропуштена од стране протокола *YAP3*.

## Симулација оштећења и губитка порука

Како би у оквиру симулираног окружења уграђених рачунара што веродостојније тестирали протокол *YAP3*, неопходно је симулирати оштећења и губитак порука током преноса. У те сврхе, имплементирано је проширење протокола *YAP3* које насумично прави намерну грешку током комуникације. Намерна грешка може бити уметања додатног или промена вредности неког бајта у примљеној поруци како би проузроковали *CRC16* грешку или намерно пропуштање слања поруке *ACK* за успешно примљену поруку. Избор грешке врши се помоћу униформне расподеле, тако да је вероватноћа за сваку грешку подједнака. Помоћу увођења наведених намерних грешки може се тестирати стопа ретрансмисије као и поузданост протокола. Наведено проширење протокола *YAP3* није подразумевано активно. Да би се проширење активирало неопходно је превести изворни код протокола као што је приказано у листингу 5.12.

```
bazel build --copt "-DINJECT_MALFORMED_PACKETS_ENABLED" \
     //protocol:yap3
```

Листинг 5.12: Активација проширења за симулацију грешке током превођења протокола *YAP3*.

Извршни кôд са активираним проширењем за симулацију грешке могуће је додатно подесити приликом покретања. Приликом покретања, проширење проверава да ли је дефинисана променљива окружења (енг. *environment variable*) `YAP3_ERROR_SIMULATION_PERCENTAGE` и уколико јесте узима њену вредност као цео број. Променљива окружења `YAP3_ERROR_SIMULATION_PERCENTAGE` представља који проценат порука је потребно оштетити неком од претходно описаних намерних грешака. По пријему поруке, проширење генерише цео број из целобројне униформне расподеле опсега $[1, 100]$. Уколико је генерисани цео број у опсегу $[1, YAP3\_ERROR\_SIMULATION\_PERCENTAGE]$ умеће се грешка. Примери покретања протокола *YAP3* преведеног са активном симулацијом грешке приказани су у листинзима 5.13 и 5.14. Уколико симулација грешке није активна, променљива окружења се игноришe.

```
YAP3_ERROR_SIMULATION_PERCENTAGE=10 yap3 /dev/ser1 115200
```

Листинг 5.13: Покретање протокола *YAP3* са активном симулацијом грешке. Стопа уметнутих грешака је 10 процената.

```
YAP3_ERROR_SIMULATION_PERCENTAGE=25 yap3 /dev/ser1 115200
```

Листинг 5.14: Покретање протокола *YAP3* са активном симулацијом грешке. Стопа уметнутих грешака је 25 процената.

## Тестови за проверу поузданости и пропусности протокола

За потребе тестирања протокола *YAP3*, поред тестова јединица кôда, написана су и два тест програма који представљају корисничке програме. Један од тест програма имплементира вишеструко слање поруке, док други прима послате поруке. Изворни кôд тест програма налази се у фајловима `yap3sender.cc` и `yap3receiver.cc`, у оквиру фолдера `tests`. Помоћу наведених програма може се тестирати поузданост протокола.

Извршни кôд тест програма, преведен помоћу команди приказаних у листингу 5.15, може се покренути у оквиру окружења `Docker` након покретања протокола *YAP3*.

```
bazel build //protocol/tests:yap3sender
bazel build //protocol/tests:yap3receiver
```

Листинг 5.15: Превођење тест програма.

Покретање оба тест програма захтева позитиван цео број као аргумент командне линије. Наведени аргумент представља број порука које треба послати, односно примити. Тест програм који шаље поруке интерно генерише случајни низ бајтова који представља поруку. Генерисаном низу се, поред идентификационог заглавља, додаје још један бајт у који се уписује редни број поруке. Додатни бајт који садржи редни број поруке користи се као механизам за детекцију грешки током извршавања протокола. Уколико се протокол извршава коректно, тест програм који прима поруке ће примати поруке према редоследу слања. У супротном тест програм који прима поруке исписаће поруку о грешци. Тест програм који прима поруке неопходно је покренути пре покретања програма који шаље поруке.

Како би се тестирала пропусност протокола, тест програму који шаље поруке може се проследити додатни позитиван цео број као аргумент. Додатни аргумент представља временски период, при чему је јединица времена милисекунда, између слања две поруке. Подразумевани (енг. *default*) временски период између слања две поруке је педесет милисекунди. Пример покретања тест програма за слање и примање порука приказан је у листинзима 5.16 и 5.17.

```
yap3sender 100 20
```

Листинг 5.16: Покретање тест програма за слање порука. Порука се шаље сваких 20 милисекунди.

```
yap3receiver 100
```

Листинг 5.17: Покретање тест програма за примање порука.

Као додатан вид провере исправности извршавања протокола могу се проверити логови протокола. Протокол исписује све логове на стандардни излаз, те се за проверу исправности могу парсирати логови који исписују примљене и послате поруке преко медијума. У сваком од парсираних логова мора да постоји порука која задовољава формат поруке слоја везе. У оквиру такве поруке могу се проверити вредности контролних заглавља и утврдити да ли поруке одговарају претходно описаним алгоритмима слања и примања порука. Додатно, протокол исписује статистику извршавања протокола. Статистика садржи информације о укупном броју примљених и послатих порука, броју детектованих грешака, броју порука корисника, броју ресетовања протокола и слично.

## Евалуација и анализа резултата рада протокола

Током развоја, протокол *YAP3* је континуирано тестиран помоћу претходно наведених тест програма са различитим улазним параметрима, као и активираном симулацијом грешке. Тривијални тест примери подразумевају слање неколико десетина порука при чему се појединачна порука шаље сваких педесет милисекунди, док комплекснији тест примери подразумевају слање

Табела 5.1: Статистика извршавања протокола са различитим подешавањима.

| Временски интервал | Проценат грешке | Укупно уметнутих грешака | Број уметнутих *CRC* грешака | Број загубљених порука *ACK* | Број ретрансмисија |
|---|---|---|---|---|---|
| 5 | 0 | 0 | 0 | 0 | 0 |
| 10 | 5 | 412 | 191 | 219 | 212 |
| 15 | 10 | 975 | 496 | 479 | 588 |
| 20 | 15 | 1533 | 756 | 777 | 956 |
| 25 | 20 | 2090 | 1033 | 1057 | 1372 |
| 30 | 25 | 2762 | 1345 | 1417 | 1519 |
| 35 | 30 | 3518 | 1832 | 1686 | 2007 |

више хиљада порука са различитим временским интервалом слања појединачних порука. Време извршавања програма који шаљу и примају поруке директно је пропорционално производу броја порука и временског интервала између порука.

У табели 5.1, приказана је статистика извршавања протокола након слања десет хиљада порука при различитим подешавањима, при чему су за слање и примање порука искоришћени претходно наведени тест програми. Статистика је добијена на основу анализе логова извршавања протокола. Колона *Временски интервал* представља интервал, при чему је јединица времена милисекунда, између слања два поруке, док колона *Проценат грешке* представља проценат порука које је потребно оштетити током извршавања. Наведене колоне представљају параметре подешавања извршавања тестова интеграције протокола. Колоне *Укупно уметнутих грешака*, *Број уметнутих CRC грешака* и *Број загубљених порука ACK* представљају статистику уметнутих грешака. Колона *Број ретрансмисија* представља укупан број ретрансмисија протокола неопходних за успешно испоручивање порука.

Анализа вредности у табели 5.1 приказује да протокол у случају одсуства грешке не врши ретрансмисију порука и пружа највећу пропусност. У случају присуства грешака неопходни временски интервал између слања две поруке расте пропорционално проценту грешке услед ретрансмисија које протокол обавља. Број ретрансмисија пропорционалан је броју уметнутих *CRC* грешака. За свако октривено оштећење поруке као одговор шаље се *NACK*, што за последицу има ретрансмисију порука које се тренутно налазе у кли-

зном прозору. Број ретрансмисија зависи од брзине слања порука, уколико је временски интервал између порука превише кратак клизни прозор ће бити попуњен све време, тако да ће се у случају *CRC* грешке вршити више ретрансмисија. Однос броја ретрансмисија и број загубљених порука *ACK* показује да се протокол успешно опоравља од губитка поруке *ACK* помоћу претходно наведеног проширења протокола *Врати-Се-N*, што се такође може потврдити анализом логова извршавња.

У табели су приказани постигнути резултати у оквиру окружења *Docker* где су све поруке достављене према редоследу слања. У оквиру уграђених рачунара, уз одговарајућу подршку језгра оперативног система, могу се очекивати бољи резултати, односно мањи временски интервал између слања порука.

Пропусност протокола *YAP3* зависи од пропусности медијума за комуникацију, случајева употребе и перформанси уграђеног рачунара, тако да није могуће гарантовати општу пропусност протокола.

## 5.8 Теоријско поређење са концептуалним моделима

*HDLC* и *PPP*, иако историјски, представљају два најчешћа концептуална модела протокола слоја везе за директну комуникацију. На основу концептуалних модела написани су многи протоколи за директну комуникацију, али су најчешће ти протоколи прилагођавани за специфичне потребе корисника. Специфичне потребе корисника често представљају интелектуалну својину корисника или компанија које финансирају развој протокола, тако да имплементирани протоколи ретко постају пројекти отвореног кôда. Због претходно наведених разлога не постоји референтна имплементација отвореног кôда која се може користити и проширити према потребама корисника. Један од разлога настанка протокола *YAP3* јесте управо и недостатак референтне имплементације. Протокол *YAP3* имплементира неке механизме концептуалних модела, али постоје и битне разлике.

Модел *HDLC* предвиђа различите улоге за уређаје који учествују у комуникацији. Улоге могу бити: примарни уређај, секундарни уређај или комбинација прва два. Уколико је уређај примарни онда уређај само шаље поруке, док уколико је уређај секундарни онда уређај само прима поруке. Уколико

је тип уређаја комбинација примарног и секундарног онда уређај може и да шаље и да прима поруке. Оваква расподела улога омогућава два начина повезивања уређаја за комуникацију. Први начин подразумева један примарни и један или више секундарних уређаја, док други начин подразумева комуникацију између два уређаја у којој оба уређаја могу да шаљу и примају поруке. Проблем модела *HDLC* представља чињеница да комуникација између уређаја мора бити синхрона, односно уређај не може истовремено да шаље и прима податке. Модел *PPP*, за разлику од модела *HDLC*, предвиђа директну комуникацију између два уређаја без додељивања посебних улога и подржава асинхрону комуникацију.

Протокол *YAP3* омогућава слање и примање порука на сваком уређају на којем се извршава, тако уређај који користи протокол *YAP3* одговара комбинацији примарног и секундарног уређаја модела *HDLC*, али протокол *YAP3* омогућава асинхрону комуникацију. Протокол *YAP3* подржава директну асинхрону комуникацију између два уређаја, тако да је по томе сличан моделу *PPP*.

Модел *HDLC* трерита поруку која се шаље као низ битова, док модел *PPP* третира поруку као низ бајтова. Протокол *YAP3* третира поруку као низ бајтова, те је по томе сличнији протоколу *PPP*. Како би означили почетак и крај поруке, оба модела предвиђају уметање специјалних битова, у случају *HDLC*, или бајтова, у случају *PPP*. Технику означавања специјалних бајтова у оквиру поруке примљене са транспортног слоја, описану у секцији 5.3, протокол *YAP3* имплементира на начин предложен у моделима *HDLC* и *PPP*.

Модели *HDLC* и *PPP* предлажу употребу технике клизних прозора као механизам ретрансмисије. Модели не предлажу коју варијанту технике клизних прозора користити, избор варијанте ја на развијаоцу протокола, али је избор сужен у случају модела *HDLC*, јер модел *HDLC* предвиђа синхрону комуникацију. Протокол *YAP3* имплементира технику клизних прозора у оквиру проширења протокола *Врати-Се-N*.

Оба модела предвиђају коришћење цикличне провере редундансе као механизам детекције оштећења порука током преноса преко медијума. Протокол *YAP3* користи шеснестобитну цикличну проверу редундансе, односно *CRC16*.

Модели *HDLC* и *PPP* предлажу имплементацију механизама ретрансмисије, детекције и опоравка од грешке у оквиру слоја везе. Протокол *YAP3* имплементира наведене механизме у оквиру транпортног слоја, како би у

оквиру слоја везе енкапсулирао само приступ и подешавање медијума за комуникацију.

Генерално поређење перформанси између протокола *YAP3* и протокола који у потпуности имплементира предлоге концептуалних модела недостаје управо услед претходно наведеног проблема недостатка референтне имплементације отвореног кôда.

# Глава 6

# Закључак

У оквиру мастер рада, разматрана је употреба хардверског протокола *UART* као медијума за комуникацију између уређаја са уграђеним рачунаром. Поред описа протокола *UART*, у раду је описан референтни модел *OSI* као општи модел слојевите архитектуре протокола, као и техника клизних прозора као метод ретрансмисије, детекције и опоравка од грешке.

У раду је имплементиран протокол за директну комуникацију *YAP3* која користи *UART* као медијум за комуникацију. Како *UART* комуникација припада слоју везе референтног модела *OSI*, протокол *YAP3* имлементира слојеве вишех нивоа како би пружио поуздан канал за комуникацију. Транспортни слој протокола *YAP3*, имплементира проширење протокола *Врати-Се-N*, као метод детекције и опоравка од грешке, као и избегавања непотребне вишеструке ретрансмисије порука. Имплементирано проширење је независно од медијума за комуникацију који се користи, те се, уз одређене модификације, може користити и у неком другом протоколу. Помоћу претходно наведених механизама протокол представља поуздан и ефикасан канал за комуникацију између уграђених рачунара. Додатно, дискутовано је теријско поређење протокола *YAP3* са концептуалним моделима протокола за директну комуникацију *HDLC* и *PPP*. У оквиру рада описано је тестирање и евалуација резултата тестирања, као и како се протокол може користити и прилагодити потребама корисника.

Као могући правац даљег развоја протокола *YAP3*, планирано је проширење протокола и за друге медијуме за серијску комуникацију као што су *SPI* (*serial peripheral interface*) или *I2C* (*inter-integrated circuit*), имплементација шифровања и дешифровања, као и подршка за комплетно подешавање

протокола на основу текстуалне датотеке.

# Библиографија


[1] Basics of UART communication. `https://www.circuitbasics.com/basics-uart-communication/`.

[2] Douglas E. Comer. *Internetworking with TCP/IP*. Pearson, Boston, 2013.

[3] Gordon Davies. *Networking Fundamentals*. Packt Publishing, 2019.

[4] J.D. Day and H. Zimmermann. The OSI reference model. volume 71, pages 1334–1340. Proceedings of the IEEE, 1983.

[5] K.J. Guth and NAVAL POSTGRADUATE SCHOOL MONTEREY CA. *An Adaptive ARQ (Automatic Repeat ReQuest) Strategy for Packet Switching Data Communication Networks*. Defense Technical Information Center, 1989.

[6] ISO. `https://www.iso.org`.

[7] M. Kerrisk. *The Linux Programming Interface: A Linux and UNIX System Programming Handbook*. No Starch Press, 2010.

[8] Leo Louis. Working principle of arduino and using it as a tool for study and research. volume 1, 07 2018.

[9] POSIX message queues. `https://man7.org/linux/man-pages/man7/mq_overview.7.html`.

[10] Umakanta Nanda and Sushant Kumar Pattnaik. Universal Asynchronous Receiver and Transmitter (UART). volume 01, pages 1–5. 2016 3rd International Conference on Advanced Computing and Communication Systems (ICACCS), 2016.

[11] Eric Peňa and Mary Grace Legaspi. UART: A Hardware Communication Protocol Understanding Universal Asynchronous Receiver/Transmitter. volume 54. Analog Dialogue, 2020.

[12] Programming language C. `https://www.open-std.org/jtc1/sc22/wg14/www/docs/n1570.pdf`.

[13] N.M. Rao and M.M. Naidu. *Sliding Window Algorithm for Mobile Communication Networks*. Springer Singapore, 2018.

[14] Standard RS232. `https://mil.ufl.edu/4744/docs/RS232_standard_files/RS232_standard.html`.

[15] W.R. Stevens, B. Fenner, and A.M. Rudoff. *UNIX Network Programming: The sockets networking API*. Number Bd. 1 in Addison-Wesley professional computing series. Addison-Wesley, 2004.

[16] Andrew S. Tanenbaum and David Wetherall. *Computer Networks*. Prentice Hall, Boston, 2011.

[17] termios.h. `https://www.man7.org/linux/man-pages/man3/termios.3.html`.

[18] Linux tty command. `https://man7.org/linux/man-pages/man1/tty.1.html`.

[19] Jan L. A. Van de Snepscheut. The sliding window protocol revisited. Technical report, USA, 1991.

# Биографија аутора

Срђан Лазаревић *(Аранђеловац, Србија, 18. март 1995.)* је дипломирани информатичар Универзитета у Београду. Завршио је средњу Техничку школу „Милета Николић", смер Електротехничар рачунара, 2014. године у Аранђеловцу.

Исте године уписао је Математички факултет у Београду, смер Информатика, и дипломирао у јануару 2018. године са просечном оценом 7.66. По завршетку основних академских студија уписао је мастер студије, где је положио све испите предвиђене планом и програмом мастер студија, са просечном оценом 8.46.

У октобру 2017. почиње своју професионалну каријеру као софтверски инжењер у Научно-истраживачком институту *RT-RK*, где је био део тима који се бавио развојем и одржавањем пројеката отвореног кода *V8*, *WebKit* и *GNU GDB* за архитектуру *MIPS*. Од јануара 2019. ради на развоју софтверске платформе високих перформанси (енг. *high performance platform*) за нову генерацију аутономних возила. Тренутно је на позицији главног софтверског инжењера у области оперативних система и животног циклуса уграђених рачунара у компанији *BMW*.