|  |  |  |
| --- | --- | --- |
|  | УНИВЕРЗИТЕТ У НОВОМ САДУ  **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**  Трг Доситеја Обрадовића бр. 6, Нови Сад |  |

**Јован Славујевић**

**Реализација трослојног система за видео пренос по *RTSP* протоколу**

МАСТЕР РАД

- Мастер академске студије -

**Ментор:**

**проф. др Предраг Теодоровић**

Нови Сад, октобар 2021.

|  |  |  |
| --- | --- | --- |
|  | УНИВЕРЗИТЕТ У НОВОМ САДУ l **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**  21000 НОВИ САД, Трг Доситеја Обрадовића 6 | Датум: |
|  |
| **ЗАДАТАК ЗА ИЗРАДУ  МАСТЕР РАДА** | Лист/Листова: |
| 2/41 |

*(Податке уноси предметни наставник - ментор)*

| Врста студија: | Основне академске студије |
| --- | --- |
| Студијски програм: | **Енергетика, електроника и телекомуникације** |
| Руководилац студијског програма: | **Др Милан Сечујски** |

|  |  |  |  |
| --- | --- | --- | --- |
| Студент: | **Јован Славујевић** | Број индекса: | **Е1-12/2020** |
| Област: | **Електроника** | | |
| Ментор: | **др Предраг Теодоровић** | | |
| НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА  ИЗДАЈЕ СЕ ЗАДАТАК ЗА МАСТЕР РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА:   * проблем – тема рада; * начин решавања проблема и начин практичне провере резултата рада, ако је таква провера неопходна; * литература | | | |

**НАСЛОВ МАСТЕР РАДА:**

|  |
| --- |
| **Реализација трослојног система за видео пренос по *RTSP* протоколу** |

**ТЕКСТ ЗАДАТКА:**

|  |
| --- |
| У оквиру овог мастер рада потребно је урадити следеће:   * Адаптирати библиотеку утичнице у објектно оријентисаној форми за *TCP/UDP IP* протоколе * Имплементирати *RTSP* серверску конзолну апликацију са подршком видео преноса фрејмова уживо са камере за програм *VLC Media Player* * Имплементирати *RTSP* клијентску конзолну апликацију са читањем видео преноса по узору на програм *VLC Media Player* * Унапредити клијентску апликацију у апликацију са графичким корисничким интерфејсом * Проширити услуге видео преноса за једнозначно приказивање по захтеву (енг. *On Demand*) раније снимљених видео записа у временским слотовима од 15 минута * Имплементирати базу података за евиденцију клијената и приступ видео преносима са креденцијалима * Омогућити заштиту корисничких података путем енкрипције размењених порука * Тестирати пројектовани систем и написати мастер рад |

|  |  |
| --- | --- |
| Руководилац студијског програма: | Ментор рада: |
| др Милан Сечујски | др Предраг Теодоровић |

|  |
| --- |
| Примерак за:  - Студента;  - Ментора |

Образац **Q2.НА.15-04** - Издање 2

|  |  |  |  |
| --- | --- | --- | --- |
| Редни број, **РБР**: | |  | |
| Идентификациони број, **ИБР**: | |  | |
| Тип документације, **ТД**: | | Монографска публикација | |
| Тип записа, **ТЗ**: | | Текстуални штампани материјал | |
| Врста рада, **ВР**: | | Мастер рад | |
| Аутор, **АУ**: | | Јован Славујевић | |
| Ментор, **МН**: | | Проф. др Предраг Теодоровић, доцент | |
| Наслов рада, **НР**: | | Реализација трослојног система за видео пренос по *RTSP* протоколу | |
| Језик публикације, **ЈП**: | | Српски / ћирилица | |
| Језик извода, **ЈИ**: | | Српски | |
| Земља публиковања, **ЗП**: | | Република Србија | |
| Уже географско подручје, **УГП**: | | Аутономна Покрајина Војводина | |
| Година, **ГО**: | | 2021. | |
| 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) | | (8/61/41/3/46/0/0) | |
| Научна област, **НО**: | | Електроника и телекомуникације | |
| Научна дисциплина, **НД**: | | Видео обрада и рачунарске комуникације | |
| Предметна одредница/Кqучне речи, **ПО**: | | *RTSP*, *Qt, GUI, SQL,* криптографија | |
| **УДК** | |  | |
| Чува се, **ЧУ**: | | Библиотека Факултета техничких наука, Нови Сад | |
| Важна напомена, **ВН**: | |  | |
| Извод, **ИЗ**: | | У овом раду је описана реализација и функционисање трослојног система за видео пренос по протколу за стримовање у реалном времену. Омогућено је вишезначно видео преношење уживо са једне или више камера повезаних на уређај на којем се серверска апликација извршава, као и једнозначно приказивање по захтеву раније снимљених видео записа у временским размацима од 15 минута. Систем је имплементиран на персоналној машини - *i7@3GHz* са 16 *GB RAM* и *Windows* *10* *OS*. Систем је подржан и на *Linux* *OS*, како би се омогућила имплементација на популарним ембедед платформама као што је нпр. *Raspberry Pi*. Развој је текао у комбинованом *Microsoft Visual Studio* и *Qt* окружењу уз помоћ *CMake* алата за генерисање *build* рецепта. | |
| Датум прихватања теме, **ДП**: | |  | |
| Датум одбране, **ДО**: | |  | |
| Чланови комисије, **КО**: | Председник: | др Вук Врањковић, ванредни професор |
|  | Члан: | др Жељко Лукач, доцент | Потпис ментора |
|  | Члан, ментор: | др Предраг Теодоровић, доцент |  |

Образац **Q2.НА.04-05** - Издање 1

|  |  |  |  |
| --- | --- | --- | --- |
| 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**: | | Jovan Slavujević | |
| Mentor, **MN**: | | PhD Predrag Teodorović, docent | |
| Title, **TI**: | | Realization of three-layer *RTSP* video streaming system | |
| Language of text, **LT**: | | Serbian / Cyrillic | |
| Language of abstract, **LA**: | | English | |
| Country of publication, **CP**: | | Republic of Serbia | |
| Locality of publication, **LP**: | | Autonomous Province of Vojvodina | |
| Publication year, **PY**: | | 2021. | |
| Publisher, **PB**: | | Author’s reprint | |
| Publication place, **PP**: | | Faculty of Technical Sciences, Dositeja Obradovica sq. 6, 21000, Novi Sad | |
| Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes) | | (8/61/41/3/46/0/0) | |
| Scientific field, **SF**: | | Electronics and telecommunications | |
| Scientific discipline, **SD**: | | Video processing and computer communications | |
| Subject/Key words, **S**/**KW**: | | *RTSP*, *Qt, GUI, SQL,* cryptography | |
| **UC** | |  | |
| Holding data, **HD**: | | The Library of Faculty of Technical Sciences, Novi Sad, Serbia | |
| Note, **N**: | |  | |
| Abstract, **AB**: | | This article describes realization and functioning of three-layer video streaming system according to real-time streaming protocol. Also, here is provided live multicast video streaming from one or more cameras attached on running device, also there is provided unicast “*On Demand*” video streaming of earlier recorded video files in 15 minutes time slots. System was implemented on personal computer – *i7@3GHz* with 16 *GB RAM* and *Windows 10* *OS*. Also, System has supported *Linux* OS, in order to provide implementation on popular embedded platforms such as *Raspberry Pi*. Development was realized by combining *Microsoft Visual Studio* and *Qt* environment via *CMake* tool for build recipes generation. | |
| Accepted by the Scientific Board on, **ASB**: | |  | |
| Defended on, **DE**: | |  | |
| Defended Board, **DB**: | President: | Vuk Vranjković, PhD, full professor |
|  | Member: | Željko Lukač, PhD, docent | Mentor’s sign |
|  | Member, Mentor: | Predrag Teodorović, PhD, docent |  |

Obrazac **Q2.НА.04-05** - Izdanje

Изјава о академској честитости

Студент/киња: Јован Славујевић

Број индекса: Е1-12/2021

Студент/киња: мастер академских студија Факултета техничких наука

Аутор/ка рада под називом:

Реализација трослојног система за видео пренос по *RTSP* протоколу

Потписивањем изјављујем:

* да је рад искључиво резултат мог сопственог истраживачког рада;
* да сам рад и мишљења других аутора које сам користио/ла у овом раду назначио/ла или цитирао/ла и да су наведени у списку литературе/референци који су саставни део овог рада;
* да сам добио/ла све дозволо за коришћење ауторског дела који се у потпуности/целости уносе у предати рад и да сам то јасно навео/ла;
* да сам свестан/на да је плагијат коришћење туђих радова у било ком облику (као цитата, парафраза, слика, табела, дијаграма, дизајна, планова, фотографија, филма, музике, формула, веб сајтова, компјутерских програма и сл.) без навођења аутора или представљање туђих ауторских дела као мојих, кажњиво по закону (Закон о ауторским и сродним правима, Службени гласник Републике Србије, бр. 104/2009, 99/2011, 119/2012), као и других закона и одговарајућих аката Универзитета у Новом Саду;
* да сам свестан/на да плагијат укључује и представљање, употребу и дистрибуирање рада предавача или других студената као сопствених;
* да сам свестан/на последица које код доказаног плагијата могу проузроковати на предати рад и мој статус;
* да је електронска верзија рада идентична штампаном примерку и пристајем на његово објављивање под условима прописаним актима Универзитета.

Нови Сад, 03.11.2021 Потпис студента/киње:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Захвалност

Посебну захвалност дугујем својим родитељима и девојци који су били велика подршка и чврст ослонац током досадашњег школовања.

Захваљујем се ментору проф. др Предрагу Теодоровићу на саветима, стрпљењу и стручној помоћи приликом израде овог рада.

Садржај

[1. Увод 1](#_Toc86310882)

[2. Теоријске основе 3](#_Toc86310883)

[2.1 Етернет 3](#_Toc86310884)

[2.2 *OSI* референтни модел 4](#_Toc86310885)

[2.3 *IP* протокол 6](#_Toc86310886)

[2.3.1 *UDP* 7](#_Toc86310887)

[2.3.2 *TCP* 8](#_Toc86310888)

[2.4 *RTSP* протокол 9](#_Toc86310889)

[2.4.1 *RTP* 10](#_Toc86310890)

[2.4.2 *RTCP* 11](#_Toc86310891)

[2.5 Кодек 13](#_Toc86310892)

[2.5.1 *H.264* 14](#_Toc86310893)

[2.6 Базa података 16](#_Toc86310894)

[2.6.1 *SQL* 17](#_Toc86310895)

[2.7 Криптографија 19](#_Toc86310896)

[2.7.1 Асиметрична енкрипција 20](#_Toc86310897)

[2.7.2 Симетрична енкрипција 21](#_Toc86310898)

[3. Алати и *3rd-party* библиотеке 22](#_Toc86310899)

[3.1 *CMake* 22](#_Toc86310900)

[3.2 *OpenCV* 23](#_Toc86310901)

[3.3 *FFmpeg* 24](#_Toc86310902)

[3.4 *Qt* 25](#_Toc86310903)

[3.5 *MySQL* 26](#_Toc86310904)

[3.6 *Crypto++* 27](#_Toc86310905)

[4. Концепт решења 28](#_Toc86310906)

[4.1 Библиотека утичнице у објектно-оријентисаној форми 28](#_Toc86310907)

[4.2 Архитектура и функционалност серверске апликације 31](#_Toc86310908)

[4.3 Архитектура и функционалност клијентске апликације 33](#_Toc86310909)

[5. Програмско решење 35](#_Toc86310910)

[5.1 Програмско решење серверске апликације 35](#_Toc86310911)

[5.1.1 Сервер за видео пренос 35](#_Toc86310912)

[5.1.1.1 Читање са камере и уписивање у видео запис 36](#_Toc86310913)

[5.1.1.2 *RTSP* интеракција са клијентима 38](#_Toc86310914)

[5.1.1.3 *RTP* вишезначни видео пренос 41](#_Toc86310915)

[5.1.1.4 *RTP* једнозначни “*On Demand*” видео пренос 43](#_Toc86310916)

[5.1.2 Мрежни менаџер 46](#_Toc86310917)

[5.1.2.1 Интеракција са мрежним корисницима 46](#_Toc86310918)

[5.1.2.2 Читање и писање у бази података 47](#_Toc86310919)

[5.1.2.3 Криптовање порука 50](#_Toc86310920)

[5.2 Програмско решење клијентске апликације 51](#_Toc86310921)

[5.2.1 *RTSP* клијент 52](#_Toc86310922)

[5.2.2 *RTP* клијент 54](#_Toc86310923)

[5.2.3 Мрежни корисник 56](#_Toc86310924)

[6. Верификација и профилисање 58](#_Toc86310925)

[6.1 Лимити и потрошње серверске апликације 58](#_Toc86310926)

[6.2 Перформансе клијентске апликације у односу на *VLC* 59](#_Toc86310927)

[7. Закључак 60](#_Toc86310928)

[8. Литература 61](#_Toc86310929)

Списак слика

[*Слика 1. Различите платформе за видео пренос* [3] 1](#_Toc86310786)

[*Слика 2. Етернет каблови повезани на комутатору* [5] 3](#_Toc86310787)

[*Слика 3. Илустрација OSI нивоа* [8] 4](#_Toc86310788)

[*Слика 4. Пример IPv4 адресе* [11] 6](#_Toc86310789)

[*Слика 5. Илустрација UDP комуникације* [13] 7](#_Toc86310790)

[*Слика 6. Илустрација различитих начина UDP преноса* [15] 8](#_Toc86310791)

[*Слика 7.* *Илустрација TCP комуникације* [13] 8](#_Toc86310792)

[*Слика 8. Блок шема RTSP протокола* [19] 9](#_Toc86310793)

[*Слика 9. Архитектура RTP протокола* [19] 11](#_Toc86310794)

[*Слика 10. Илустрација RTCP* [19] 11](#_Toc86310795)

[*Слика 11. Разлика компресије без и са губицима* [21] 13](#_Toc86310796)

[*Слика 12. Процес кодирања и декодирања H.264 кодека* [23] 14](#_Toc86310797)

[*Слика 13. H.264 Предвиђања* [23] 14](#_Toc86310798)

[*Слика 14. Архитектура базе података у слојевима* [27] 17](#_Toc86310799)

[*Слика 15. Блок шема криптографије* [30] 19](#_Toc86310800)

[*Слика 16. Илустрација асиметричне енкрипције* [31] 21](#_Toc86310801)

[*Слика 17. Илустрација симетричне енкрипције* [31] 21](#_Toc86310802)

[*Слика 18. CMake лого* [32] 22](#_Toc86310803)

[*Слика 19. OpenCV лого* [33] 23](#_Toc86310804)

[*Слика 20. FFmpeg лого* [34] 24](#_Toc86310805)

[*Слика 21. Qt лого* [36] 25](#_Toc86310806)

[*Слика 22. MySQL лого* [38] 26](#_Toc86310807)

[*Слика 23. Crypto++ лого* [40] 27](#_Toc86310808)

[*Слика 24. Уобичајени сценарио коришћења утичнице у TCP* [41] 28](#_Toc86310809)

[*Слика 25. UML класни дијаграм Wrapper библиотеке* 29](#_Toc86310810)

[*Слика 26. Идејна блок шема серверске апликације* 32](#_Toc86310811)

[*Слика 27. Идејна блок шема клијентске апликације* 34](#_Toc86310812)

[*Слика 28. Блок дијаграм сервера за видео пренос* 35](#_Toc86310813)

[*Слика 29. UML класни дијаграм за H.264 Streaming Wrapper* 36](#_Toc86310814)

[*Слика 30. Алгоритам рада читања са камере и видео записивања* 38](#_Toc86310815)

[*Слика 31. UML класни дијаграм RTSP сервера* 39](#_Toc86310816)

[*Слика 32. Алгоритам рада RTSP сервера* 40](#_Toc86310817)

[*Слика 33. UML класни дијаграм RTP сервера* 42](#_Toc86310818)

[*Слика 34. Алгоритам рада RTP сервера* 43](#_Toc86310819)

[*Слика 35. UML класни дијаграм “On Demand” RTP сервера* 44](#_Toc86310820)

[*Слика 36. Алгоритам рада “On Demand” RTP сервера* 45](#_Toc86310821)

[*Слика 37. UML класни дијаграм мрежног менаџера* 46](#_Toc86310822)

[*Слика 38. Алгоритам рада NetworkClientHandler* *инстанце* 49](#_Toc86310823)

[*Слика 39. UML класни дијаграм клијентске апликације* 52](#_Toc86310824)

[*Слика 40. Изглед листе активних видео преноса* 53](#_Toc86310825)

[*Слика 41. Успешно приказивање живог видео преноса* 54](#_Toc86310826)

[*Слика 42. Алгоритам рада RTP клијента* 55](#_Toc86310827)

[*Слика 43. Тексутално поље и дугме за конекцију са мрежним менаџером* 56](#_Toc86310828)

[*Слика 44. Dialog прозор за пријаву корисника* 56](#_Toc86310829)

[*Слика 45. Dialog прозор за регистрацију корисника* 57](#_Toc86310830)

[*Слика 46. Резултати перформанси између два решења RTP клијента* 59](#_Toc86310831)

**Списак табела**

[*Табела 1. Потрошња серверске апликације у режиму мировања* 58](#_Toc86276245)

[*Табела 2. Потрошња серверске апликације у режиму живог преноса* 58](#_Toc86276246)

[*Табела 3. Потрошња серверске апликације у режиму преноса по захтеву* 58](#_Toc86276247)

# 1. Увод

Видео пренос представља незаобилазну појаву данашњице, а у највећој мери предњачи интернет видео пренос. Не изненађује чињеница да је директност телевизије пронашла пут до интернета. Значај интернет видео присуства се више ни не доводи у питање за организацију свих врста и величина. *Online* видео постаје императив, па је тако све више различитих платформи које омогућавају живи видео пренос.

Видео пренос уживо (енг. *Livestreaming*) [1] представља емитовање директног видео преноса путем интернета. Овај пренос се, како му само име каже, врши уживо, у реалном времену, и корисницима преноса се пружа могућност да истовремено виде догађај који је у току чак и на супротном крају света. Видео пренос уживо разликује се од преноса по захтеву (енг. *On-Demand*), јер се код овог другог репродукује претходно снимљен садржај и емитује се у засебном циклусу за сваког корисника. Најбољи пример видео преноса по захтеву представља *YouTube* [2], премда је у последњих неколико година подржан и у експанзији видео пренос уживо и на овој платформи.

Постоје два типа живог преноса. У првом, пренос је доступан директно на интернет страници одакле се користи интегрисани софтвер. У другом типу, праћење преноса могуће је само уз помоћ самосталног (енг. *Standalone*) програма – рачунарске или мобилне апликације, где софтвер омогућава приказивање живог преноса.



*Слика 1. Различите платформе за видео пренос* [3]

У оквиру овог рада реализован је софтверски пакет који обухвата видео пренос - серверску страну, али и његово приказивање - корисничку страну. Ток видео преноса диктиран је по специјалном стандарду видео преноса какав је протокол за пренос у реалном времену (енг. *Real-Time Streaming Protocol*) – *RTSP*. Поред тога приступ видео преносима лимитиран је евидентирањем корисника у бази података и приступањем уз помоћ јединствених корисничких креденцијала.

Видео пренос представља пренос директних снимака са камере. Број камера је конфигурабилан, па их тако може бити једна или више. Камере морају бити физичке повезане на уређају на којем је покренут сервер. Систем подржава и својеврсно „враћање у назад“ по захтеву у виду 4 могућа преноса која константно пружају снимак од последњих сат времена у размацима од по 15 минута.

Корисничка апликација поседује напредни графички кориснички интерфејс (енг. *Graphical User Interface*) – *GUI*, са менијем оријентисаним ка кориснику (енг. *User-friendly*). Цео систем је имплементиран и тестиран на локалној мрежи, али би уз извесне измене могао да функционише и на глобалној мрежи.

Рад се састоји од следећих поглавља:

* Првог, уводног поглавља;
* Другог поглавља, у оквиру којег су описане теоријске основе коришћених технологија у оквиру пројекта;
* Трећег поглавља, где су побројани алати и библиотеке које су интегрисане у софтверско решење пројекта;
* Четвртог поглавља, у оквиру ког је описан концепт решења;
* Петог поглавља, у којем је евалуирано програмско решење;
* Шестог поглавља, које се тиче резултата тестирања функционалности и перформанси система;
* Последњег, седмог поглавља, у којем су изнети закључак рада и могућа унапређења тренутне верзије реализованог система.

# 2. Теоријске основе

У овом поглављу су описане теоријске основе коришћених технологија у оквиру овог пројекта.

## 2.1 Етернет

Етернет (енг. *Ethernet*) [4] представља једну од највише коришћених мрежних технологија унутар локалних рачунарских мрежа (енг. *Local Area Network*) – *LAN*. У последњим годинама брзина преноса података досеже и величине реда *Gbps*, што је енорман раст у односу на њене скромније почетке.

Стандарди *Ethernet 2.0* и *IEEE 802.3* су његови најчешће коришћени стандарди. Оба користе дељиву магистралу, као посредника преноса, за коју важи да само један чвор може да преноси податке у датом тренутку. Подаци се преносе запаковани у *frame* чије заглавље мора садржати *MAC* (*Media Access Control*) адресу извора и дестинације. Сваки чвор у оквиру једне мреже врши надгледање свог локалног дељивог сегмента и преузима *frame* који је намењен њему.

Етернет користи механизам у оквиру којег сваки чвор проверава заузетост магистрале. Чвор који предаје податке чека испуњење пасивног услова (енг. *Idle condition*) на магистрали, а потом иницира слање. Ситуација у којој више чворова истовремено иницира слање након испуњења услова представља сценарио који, на жалост, изазива колизију. Услед колизије сваки од чворова прекида слање и предаје ометајући (енг. *Jamming*) сигнал, чиме долази до смањења брзине преноса података. Ометајућим сигналом информишу се остали чворови на мрежи да сачекају одређени период пре него што покушају нову трансмисију.



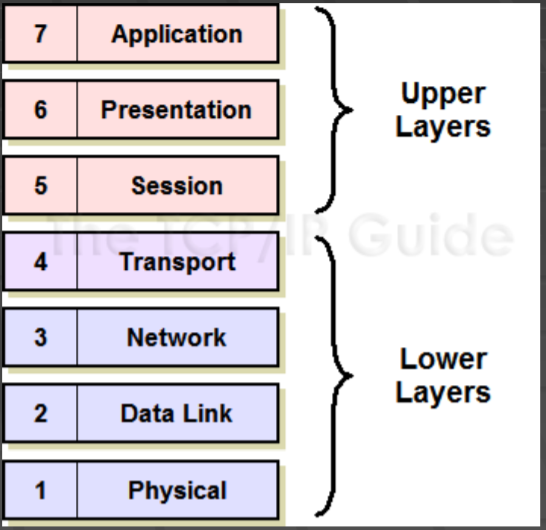
*Слика 2. Етернет каблови повезани на комутатору* [5]

## 2.2 *OSI* референтни модел

*Open System Interconnection (OSI)* [6,7] референтни модел представља апстрактни опис архитектуре мреже, у оквиру којег је описана интеракција уређаја, програма, сервиса и протокола при мрежним комуникацијама. Он представља све процесе потребне за успешну комуникацију и дели све процесе у нивое (слојеве). Овако дизајниран модел се назива слојевита архитектура.

Овај модел састоји се од седам различитих нивоа, подељених у две групе.

* Прву групу сачињавају горња три слоја (енг. *Upper Layers*) у коју спадају слојеви апликације, презентације и сесије. Ова група описује процес интеракције корисник-рачунар, рад корисника са апликацијом и процес комуникације апликација међу собом као крајњим тачкама;
* Друга група је сачињена од доња четири слоја (енг. *Lower Layers*) који дефинишу како се преносе информације са једног на други крај.



*Слика 3. Илустрација OSI нивоа* [8]

Као што је већ речено *OSI* референтни модел се састоји од седам различитих нивоа, а нивои су:

* Физички слој (енг. *Physical layer*) представља трансмисију сигнала. Овај слој дефинише електрична и физичка својства мрежних. Примери протокола су *RS-232*, *RS-449*, *OTN*, *DSL*, *Ethernet*, *USB*, *Bluetooth* и други;
* Слој везе (енг. *Data link layer*) има за улогу физичко адресирање, приступ медијуму, брине се за размену података између мрежних уређаја и за детекцију/конекцију могућих грешака у физичком слоју. Примери протокола су *PPP*, *HDLC*, *CSLIP*, *SLIP* и други;
* Мрежни слој (енг. *Network layer*) је слој који описује протоколе и сервисе који обезбеђују идентификацију крајњих корисника мреже, као и руте између њих. Поред *IP* (*IPv4*, *IPv6*) који је најзначајнији, мрежни протоколи су и *ICMP*, *ARP*, *RARP* и други;
* Транспортни слој (енг. *Transport layer*) води рачуна о пакетима који путују између два рачунара. Примери протокола на транспортном слоју су *TCP* и *UDP,* о којима ће бити више речено у неком од наредних поднаслова;
* Слој сесије (енг. *Session layer*) је слој који се бави успостављањем везе између крајњих корисника, те синхронизацијом исте. Најлакше га је објаснити код видеа преко интернета, где не желимо имати тон без слике, или слику без тона, или обоје али без синхронизације. За то се брине овај слој. Примери протокола су *PAP*, *CHAP*, *SSH*, *SAP*, *SIP* и други;
* Слој презентације (енг. *Presentation layer*) покрива енкрипцију и кодирање податка. Подаци који се користе на разним рачунарима се кодирају на разне начине (*little-endian*, *big-endian*). Све такве конверзије се изводе (уколико су имплементиране) на презентационом слоју. Примери протокола су *MIME*, *XDR*, *TLS* и *SSL*;
* Апликативни слој (енг. *Application layer*). На овом слоју програмер користи *API* са којим остварује мрежну комуникацију са одређеном сврхом, а да притом не мора водити рачуна о нижим слојевима, за које се брине оперативни систем. Примери протокола су *HTTP*, *FTP*, *telnet*, *SMTP*, *NNTP* и други.

## 2.3 *IP* протокол

Интернет протокол (енг. *Internet Protocol*) *– IP* [9,10] је протокол трећег слоја *OSI* референтног модела (слоја мреже). Садржи информације о адресирању, чиме се постиже да сваки мрежни уређај (рачунар, сервер, радна станица, интерфејс рутера) који је повезан на интернет има јединствену адресу и може се лако идентификовати у целој интернет мрежи. Такође, садржи контролне информације које омогућавају пакетима да буду прослеђени (рутирани) на основу познатих *IP* адреса.

*IP* има три примарне функције:

* Адресирање – дефинише начин доделе интернет адреса;
* Рутирање – одређивање путање за пренос података са једног рачунара на други без претходног успостављања везе (енг. *Connectionless*), по *Best-effort* модулу;
* Фрагментацију – и поновно састављање пакета када је то потребно.

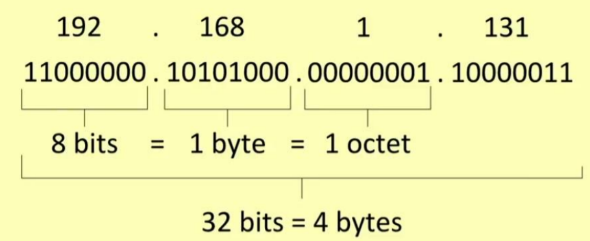
С обзиром да је сам концепт *IP* протокола ослобођен механизама који осигуравају поузданост, сам процес усмеравања (рутирања) пакета унутар мреже је релативно брз и једноставан. *IP* не захтева претходно успостављање везе у тренутку слања података, већ рачунар који шаље податке покушава све док не проследи поруку. Пренос података је релативно непоуздан, што значи да нема готово никакве гаранције да ће послати пакет заиста и доћи до одредишта, може доћи до промене садржаја, да сегменти не стижу по редоследу, могу се дуплирати или потпуно изгубити. Тако да уколико апликација захтева поузданост, користи се механизам *TCP* протокола у слоју изнад самог *IP* протокола.

Постоји више различитих типова *IP* адреса:

* Приватна *IP* адреса – адреса уређаја у оквиру локалне мреже;
* Јавна *IP* адреса – глобална адреса рутера (мреже) коју деле сви уређаји у оквиру једне локалне мреже;
* Статичка *IP* адреса – адреса коју ручно додељује корисник. Не мења се;
* Динамичка *IP* адреса – адреса коју уређају додељује мрежа и може се променити.

Такође, постоје и две верзије *IP*:

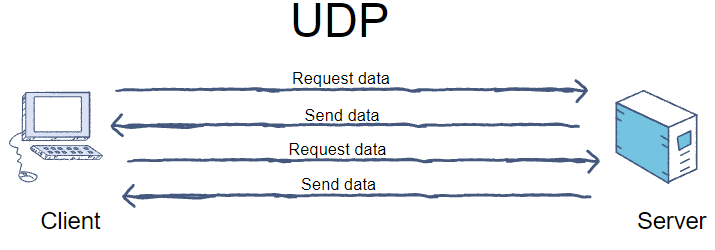
* Интернет протокол верзија 4 (*IPv4*) – састоји се од 4 броја која могу имати вредност од 0 до 255, где су они раздвојени тачкама. Сваки број представљен је једним бајтом. *IP* адресу рачунар види као 32-битни број;
* Интернет протокол верзија 6 (*IPv6*) – наследник *IPv4*, где су адресе 128 бита широке, што је тачно 2128 уникатних адреса домаћинских интерфејса.



*Слика 4.* *Пример IPv4 адресе* [11]

### 2.3.1 *UDP*

*UDP* (*User Datagram Protocol*) [9,12] је једноставан протокол који обезбеђује основне функције транспортног слоја *OSI* модела. Користи се за размену пакета порука (*datagram*) између рачунара. За разлику од *TCP*, овај протокол не подразумева сталну везу него се пакети “бацају” одредишном рачунару, без одржавања везе и провере грешака. На овај начин протокол не гарантује испоруку пакета нити исти редослед испоруке пакета као при слању. Због ових особина *UDP* протокол је брз и користи се за апликације којима је важна брзина, док се приспеће пакета и одржавање редоследа препушта предајној страни. Користи га велики број апликација, нарочито мултимедијалне попут интернет телефоније и видео конференције, као и протоколи *RTP*, *VoIP*, *DNS*, сервери за рачунарске игре и други.



*Слика 5. Илустрација UDP комуникације* [13]

*UDP* подржава сва три основна типа адреса: једнозначна (енг. *Unicast*), адреса за емитовање (енг. *Broadcast*) и вишезначна (енг. *Multicast*).

*Unicast*

Једнозначна (енг. *Unicast*) [14] врста преноса података користи се за нормални пренос података ка једном кориснику (*host-to-host*). Једнозначна метода ради на клијент-серверу и *peer-to-peer* мрежама. У једнозначним пакетима као одредишна *IP* адреса користи се одређена *IP* адреса уређаја којем је овај пакет намењен. *IP* адреса одређеног уређаја састоји се од дела мрежне адресе (у којој се налази овај уређај) и дела адресе *host*-a (део који стално одређује ову специфичност у својој мрежи). Све то доводи до могућности усмеравања једнозначног пакета по целој мрежи.

*Broadcast*

Емитовање (енг. *Broadcast*) [14] се користи за слање пакета свим *host*-овима на мрежи, па из тог разлога пакети користе посебну *IP* адресу за емитовање. Када *host* прими пакет сa адресом за емитовање, тај пакет третира као пакет за једну поруку.

Примери када се користи пренос података путем емитовања:

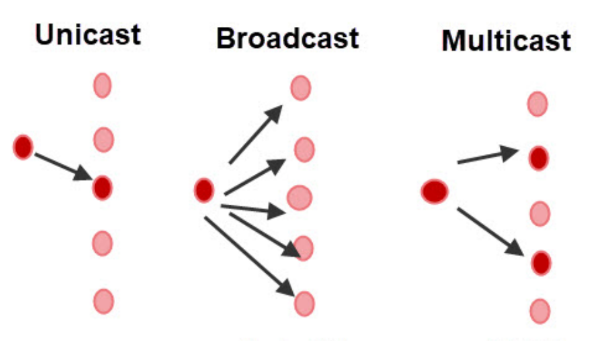
* Стварање мапе припадности адреса вишег нивоа нижим;
* Захтев за адресу;
* Протоколи усмеравања размењују информације о усмеравању.

*Multicast*

Вишезначни (енг. *Multicast*) [14] пренос је дизајниран за очување пропусности у *IP* мрежама. Овај тип смањује промет допуштајући *host*-овима да пошаљу један пакет одабраној групи *host*-овa. Код вишезначног преноса, изворни *host* може послати само један пакет који може досећи хиљаде примаоца.

Примери вишезначног преноса података:

* Видео и аудио извештај;
* Размена информација о рутама које се користе у усмереним протоколима;
* Дистрибуција софтвера;
* Сажеци вести.

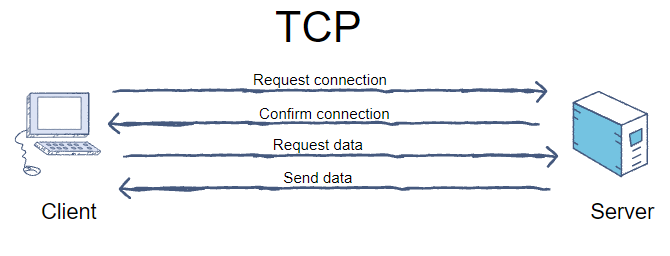


*Слика 6. Илустрација различитих начина UDP преноса* [15]

### 2.3.2 *TCP*

Трансмисиони контролни протокол (енг. *Transmission control protocol*) – *TCP* [9,16]је протокол који припада четвртом слоју *OSI* референтног модела, има за улогу да обезбеди поуздан трансфер података у *IP* окружењу. Трансмисиони контролни протокол нуди поузданост, ефикасну контролу тока података, истовремено слање и примање података и мултиплексирање које омогућава истовремен рад низа процеса са виших слојева једне конекције. Врши трансфер података као неструктуиран низ бајтова који се идентификују секвенцом. Овај протокол групише бајтове у сегменте, додели им број секвенце, апликацијама додели број порта и проследи их *IP* протоколу.

Омогућава комуникационе услуге на средњем нивоу између апликационог програма и интернет протокола. Комуникација између апликација уз помоћ *TCP* протокола се одвија тако што се прво између клијента и сервера успостави веза, за разлику од комуникације *UDP* протоколом у којем нема успостављања везе.



*Слика 7.* *Илустрација TCP комуникације* [13]

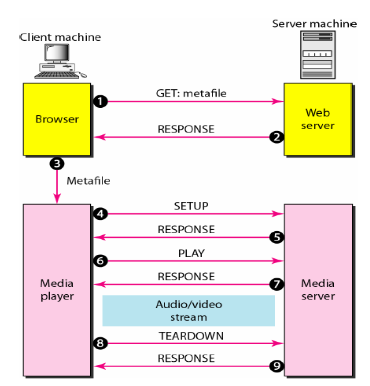
## 2.4 *RTSP* протокол

*Real-Time Streaming Protocol* – *RTSP* [17,18] је протокол апликативног нивоа намењен контроли доставе података са *real-time* својствима као што су аудио и видео. *RTSP* остварује и контролише један или више временски синхронизованих токова континуираних података, не захтевајући конекцију. Сервер одржава сесију која није никако зависна од веза преносних слојева као што је на пример *TCP* веза. Клијент зато може да користи и протокол без везе (енг. *Connectionless*) као што је *UDP*. Токови података контролисаних од стране *RTSP*-а могу користити *RTP* али функције које обавља *RTSP* не зависе од транспортног механизма који се користи за пренос података.

Користећи *RTSP* омогућује се потпуна контрола над репродукцијом аудио и видео записа. *RTSP* је управљачки протокол сличан секундарној конекцији у случају *FTP* протокола. Овај протокол се користи и код *IPTV* и служи за контролу и пренос података.

Протокол је *web* оријентисан и функционише на следећи начин:

* *HTTP* клијент приступа *web* серверу наредбом *GET*;
* *Web* сервер шаље *metafile* клијенту, који се прослеђује *media player-u*;
* *Media player* успоставља везу са *media* сервером слањем команде *SETUP*;
* *Media* сервер потврђује сесију;
* *Media player* шаље команду *PLAY* да би започео репродукцију (енг. *Streaming*);
* Аудио и видео запис се преноси користећи протокол на бази *UDP* протокола;
* Конекција се прекида командом *TEARDOWN*;
* *Media* сервер потврђује раскид сесије.



*Слика 8.* *Блок шема RTSP протокола* [19]

### 2.4.1 *RTP*

*Real-time Transport Protocol* – *RTP* [17,18] осигурава подршку за пренос података у стварном времену (енг. *Real-time*). Као и *RTSP*, у питању је протокол апликативног нивоа. Услуге које пружа:

* Временска реконструкција;
* Откривање изгубљених пакета;
* Сигурност и идентификација садржаја.

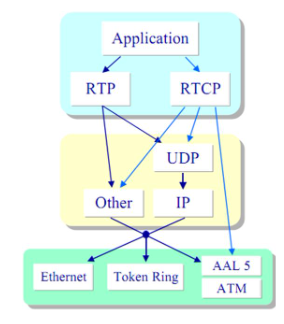
*RTP* је примарно створен за вишезначни пренос *real time* података, али се може користити и за једнозначни пренос. *RTP* се допуњује са *RTCP* контролним протоколом како би добио податке о квалитету преноса и о учесницима у преносу. Пакети послати преко интернета имају непредвидиво кашњење због несинхронизованости двеју страна, стране која шаље и стране која прима пакете. *Real time* апликације захтевају временски синхронизовано слање и репродукцију података. Главне особине *RTP* протокола су:

* Временско означавање (енг. *Timestamping*);
* Нумерација пакета (енг. *Sequence numbers*).

Временско означавање је најважнији податак апликација које захтевају истовремену комуникацију. Пошиљалац у то поље уписује тренутак слања првог узорка. Временске ознаке расту са количином времена коју покрива пакет. Након пријема пакета, пријемник користи временске ознаке како би правилно реконструисао примљени податак.

*UDP* не испоручује пакете редоследом којим су послати па се користи нумерација пакета како би се пристигли пакети правилно распоредили. Из тог разлога већина данашњих уређаја поседује један пријемни одељак у који смешта тек пристигле пакете и покушава да их прерасподели. Помоћу нумерације пакета такође се може открити и губитак пакета. У једном тренутку преноса, пошиљалац *RTP* пакета може слати само једну врсту или тип података и у току преноса могуће је да се та врста промени услед утицаја неких фактора (загушења мреже). Такође, *RTP* садржи и информацију о извору сигнала, што омогућује пријемној страни да зна одакле долазе подаци. *RTP* ради преко *UDP* протокола како би искористио његове особине погодне за мултимедијске садржаје. *UDP* је изабран као одредишни протокол за *RTP* из два разлога:

* *RTP* је дизајниран примарно за вишезначно слање па му самим тим директна веза *TCP* не одговара;
* За апликације које раде у реалном времену, поузданост испоруке пакета није једнако важна као и правовременост доласка истих.



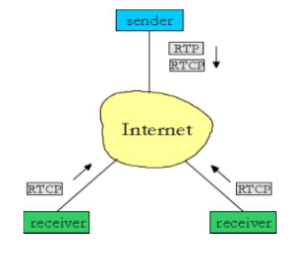
*Слика 9. Архитектура RTP протокола* [19]

### 2.4.2 *RTCP*

*Real-Time Control Protocol - RTCP* [19] представља контролни протокол који ради заједно са *RTP*-ом на апликативном слоју. Протоколи испод апликативног слоја морају осигурати мултиплексирање контролних пакета и пакета података (на пример, коришћењем различитог броја порта код *UDP*-а).Сваки учесник у *RTP* сесији периодично шаље *RTCP* контролне пакете свим другим учесницима. Сваки *RTCP* пакет садржи извештај пошиљаоца и/или примаоца.

*RTCP* сесија:

* Углавном једна вишезначна адреса - сви *RTP/RTCP* пакети који припадају сесији користе вишезначну адресу;
* *RTP* и *RTCP* пакети се разликују по броју порта;
* Да би се ограничио саобраћај - сваки учесник смањује свој саобраћај са повећањем броја учесника сесије.



*Слика 10. Илустрација RTCP* [19]

Постоје четири *RTCP* функције:

* Примарна функција – слање повратних информација о квалитету сервиса за податке који су послати корисницима. Ова функција се обавља слањем *SR* и *RR*  типова података;
* Друга функција – слање сталног идентификатора извора који називамо *Canonical name* (*CNAME*). *CNAME* се користи како би се извор могао јединствено идентификовати од стране учесника комуникације. Такође, неопходан је када један извор генерише више типова података (аудио и видео подаци) да би била могућа међусобна синхронизација;
* Трећа функција – на основу информација које дају примарна и друга функција (ове две функције су обавезне за све учеснике у комуникацији) а примају сви учесници, могу се израчунати потребни интезитети слања ових пакета;
* Четврта функција – опциона је. Омогућава да се у току сесије корисницима приказују потребне информације о учесницима.

*RTCP* може синхронизовати различите медијске токове унутар једне *RTP* сесије, на пример апликација за видео конференцију где сваки учесник прави један *RTP* ток за слику, а други за звук. Временске ознаке у *RTP* пакетима везане су за *clock* узорковање слике и звука, а не за стварно време. Сваки пакет извештава пошиљаоца и садржи (за последњи пакет из одговарајућег *RTP* тока):

* Временску ознаку *RTP*;
* Стварно време кад је пакет направљен.

Пријемници повезују ове ознаке ради синхронизације слике и звука.

*RTCP* формати пакета:

* Извештај примаоца (енг. *Receiver report*) – служи за рачунање укупног кашњења између пошиљаоца и примаоца, шаље резултат примаоцу који није активни судионик трансфера;
* Извештај пошиљаоца (енг. *Sender report*) – ова порука се периодично шаље од стране активних извора и даје извештај о статистици свих посланих и примљених *RTP* пакета. Извештај садржи апсолутни временски код (*timestamp*) који представља број секунди који је протекао од поноћи;
* Опис извора (енг. *Source description message*) – описује изворе саобраћаја, што укључује и слање *CNAME* податке;
* Поздравна порука (енг. *Bye message*) – извор шаље ову поруку када затвара ток података (*stream*);
* Порука карактеристична за апликацију (енг. *Application specific message*) – ово је пакет података који дефинише употребу нове апликације (која још није дефинисана стандардом). Омогућује дефиницију новог типа поруке.

## 2.5 Кодек

Кодек [20] је алат за кодирање који обрађује видео и складишти га у низу бајтова. Овај термин води порекло од скраћенице за кодер – декодер, тј. компресор – декомпресор. Добро је познато да се приликом чувања било ког аудио или видео садржаја захтева значајна количина меморије рачунара, па се јавља потреба да такви типови садржаја буду на неки начин смањени, како би заузели што мању количину простора. Управо у те сврхе користи се кодек.

Кодеци користе алгоритме за ефикасно смањење величине аудио или видео датотеке, а затим је декомпресује по потреби. У зависности од кодека, кодирање се може јавити на један од два начина:

* са губитком (енг. *Lossy Compression*);
* без губитака (енг. *Lossless Compression*).

Када се траже величине датотека којима се може управљати, компресија са губитком [21] је најпоузданија доступна метода. Иако се сигурно губи на квалитету звука, видеа или обоје, компресија је „нужно зло“ у свету у којем доминира потреба за дељењем и складиштењем онога што би у противном значило непрактичне величине датотека.

Компресија без губитака [21] функционише слично као *ZIP* или *RAR* датотека, по томе што је након компресије и декомпресије датотека у суштини иста. Уколико се користи компресија без губитака, датотека не губи много на квалитету, али није ефикасна за сладиштење великих датотека јер се заправо не врши много компресије. Осим у случају рада у филмској индустрији или у уређивању видео записа, мала је вероватноћа дељења видео датотека у формату без губитака.



*Слика 11. Разлика компресије без и са губицима* [21]

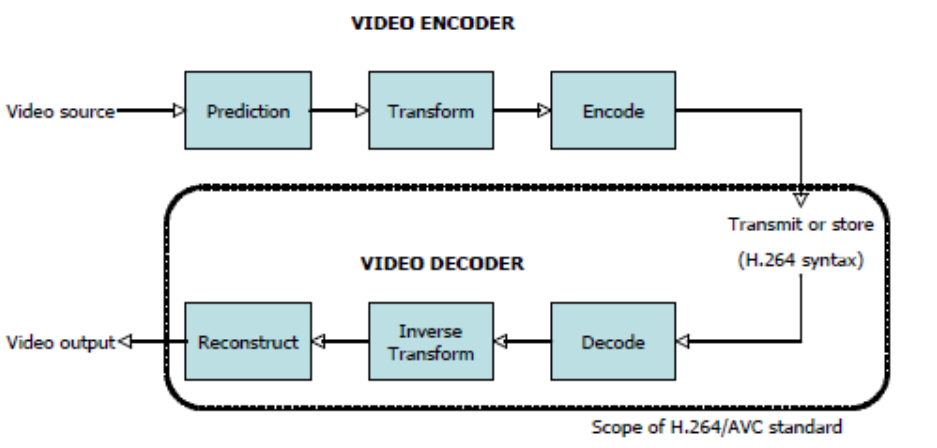
Након што је датотека кодирана помоћу одређеног кодека, исти кодек се мора користити за декодирање, како би се могла репродуковати на уређају.

Постоји на десетине различитих врста кодека који користе различите технологије како би кодирали и смањили видео датотеку. Најпознатији кодек који се користи за компресовање различитих аудио записа је *MP3, Windows Media Studio,* *AC3*, *AAC* и други, док су за сажимање различитих видео садржаја најпознатији *DivX*, *MPEG-4*, *H.264*, али постоје и *XviD*, *Windows Media Video* и многи други.

### 2.5.1 *H.264*

*H.264* [22,23] је индустријски стандард за видео компресију, процес претварања дигиталног видеа у формат који заузима мањи капацитет када се складишти или преноси. Представља најпопуларнији избор кодека, јер може користити компресију и са губитком и без губитака у зависности од поставки које су одабране приликом кодирања, као што су брзина кадрова, квалитет и циљна величина датотеке.

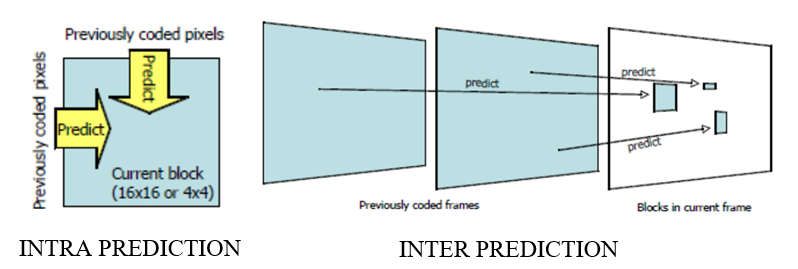
Стандард *H.264* се надовезује на концепте ранијих стандарда као што су *MPEG-2* и *MPEG-4* и нуди потенцијал за бољу ефикасност компресије и већу флексибилност у компресовању, преносу и складиштењу видео записа и беспрекорне репродукције на више уређаја.



*Слика 12. Процес кодирања и декодирања H.264 кодека* [23]

Као што је приказано на претходној слици, *H.264* видео кодер врши процесе предвиђања, трансформације и кодирања за компресију.

* Предвиђање - кодер формира предвиђање макроблока (16x16 приказаних пиксела) на основу претходно кодираних података, било из тренутног фрејма (унутар предвиђања – енг. *Intra prediction*) или из других фрејмова који су већ кодирани и пренети (међу-предвиђање – енг. *Inter prediction*);



*Слика 13. H.264 Предвиђања* [23]

* Трансформација - блок заосталих узорака се трансформише помоћу 4x4 или 8x8 целобројне трансформације. Трансформација даје скуп коефицијената, од којих је сваки тежинска вредност (енг. *Weighting value*) за стандардни основни образац. Излаз трансформације се квантизује. Постављањем параметра квантизације на ниску вредност, остаје више коефицијената који нису нула, што резултира бољим декодираним квалитетом слике, али нижом компресијом и обрнуто;
* Енкодовање (енг. *Encoding*) – низ вредности које се морају кодирати формирају компресовани ток битова (енг. *Bitstream*). Ове вредности укључују:   
  1. квантизовани коефицијенти трансформације;
  2. информације које омогућавају декодеру да поново створи предвиђање;
  3. информације о структури компресованих података и алатима за компресовање који се користе током кодирања;
  4. информације о комплетној видео секвенци;

Ове вредности и параметри се конвертују у бинарне кодове коришћењем кодирања променљиве дужине и/или аритметичког кодирања. Свака од ових метода кодирања производи ефикасну, компактну бинарну репрезентацију информација. Кодирани ток битова се тада може сачувати и/или пренети.

Видео декодер изводи комплементарне процесе декодирања, инверзне трансформације и реконструкције како би произвео декодирани видео низ.

* Декодовање (енг. *Decoding*) – декодер прима компресовани ток битова, декодира сваки од елемената синтаксе и издваја квантизоване коефицијенте трансформације, информације предвиђања и друге параметре енкодовања. Ове информације се затим користе за обрнути процес кодирања и поновно стварање низа видео слика;
* Инверзна трансформација – Квантизовани коефицијенти трансформације се поново скалирају. Сваки коефицијент се множи са целобројном вредношћу да би се вратила његова оригинална скала. Инверзна трансформација комбинује стандардне базичне обрасце, заостале у прерачунатим коефицијентима, за поновно стварање сваког блока заосталих података. Ови блокови се комбинују заједно да формирају заостали макроблок;
* Реконструкција - За сваки макроблок декодер формира идентично предвиђање оном које је направио кодер. Декодер додаје предвиђање декодираном остатку ради реконструкције декодираног макроблока који се затим може приказати као део видео кадра.

## 2.6 Базa података

База података [24,25] је скуп међусобно повезаних података, похрањених у спољњој меморији рачунара. Подаци су истовремено доступни разним корисницима и апликационим програмима. Убацивање, промена, брисање и читање података обавља се посредством заједничког софтвера.

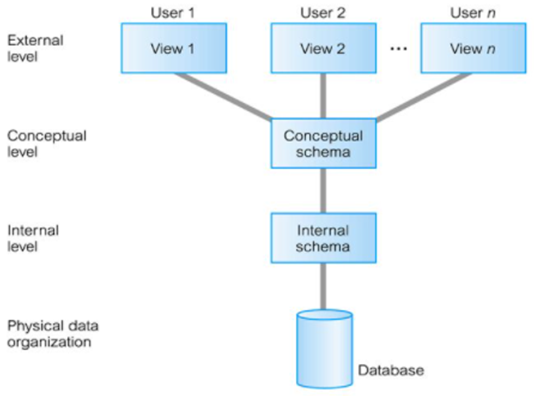
Систем за управљање базом података (енг. *Data Base Management System*) – *DBMS* [26] је сервер базе података. То је софтверски систем који омогућава креирање, дефинисање, коришћење, одржавање и контролу приступа бази података. *DBMS* омогућава крајњим корисницима или програмерима да деле податке, односно пружа могућност да се подаци истовремено користе од стране више апликација и лишава нас потребе да свака апликација има своју копију података.

Подаци у бази су логички организовани у складу са неким моделом података. Модел података је скуп правила која одређују како може изгледати логичка структура базе. Чини основу за конципирање, пројектовање и имплементацију базе. Досадашњи *DBMS*-ови су обично подржавали неке од следећих модела:

* Релациони модел – заснован на математичком појму релације, где се и подаци и везе међу подацима приказују се “правоугаоним” табелама. Овај модел представља најчешће примењени модел;
* Мрежни модел – база је предочена усмереним графом. Чворови су типови записа, а лукови дефинишу везе међу типовима записа;
* Хијерархијски модел – специјални случај мрежног модела. База је предочена једним стаблом или скупом стабала. Чворови су типови записа, а хијерархијски однос “надређени-подређени” изражава везу међу типовима записа;
* Објектни модел – инспирисан је објектно оријентисаним програмским језицима. База је скуп трајно похрањених објеката који се састоје од својих интерних података и “метода” за руковање са тим подацима. Сваки објекат припада некој класи. Између класа се успостављају везе наслеђивања, агрегације, односно међусобног коришћења операција.

Архитектура базе података састоји се од три “слоја” и интерфејса међу слојевима, као што је приказано на следећој слици. Реч је о три нивоа апстракције:

* Физичка организација података (енг. *Physical data organization*)– односи се на физички приказ и распоред података на јединицама спољне меморије. То је аспект којег виде само системски програмери (који развијају *DBMS*);
* Унутрашњи ниво (енг. *Internal level*) – у оквиру овог нивоа налази се интерна шема (енг. *Internal schema*), која описује како се елементи логичке дефиниције базе пресликавају на физичке уређаје;
* Концептуални ниво (енг. *Conceptual level*) – односи се на логичку структуру целе базе. То је аспект којег види пројектант базе, односно администратор. Запис логичке дефиниције назива се концептуална шема (енг. *Conceptual schema*). Шема је текст или дијаграм који дефинише логичку структуру базе, и у складу је са задатим моделом;
* Спољашњи ниво (енг. *External level*) – представља локални логички ниво, односи се на логичку слику о делу базе којег користи поједина апликација. То је аспект којег види корисник или апликациони програмер. Запис једне локалне логичке дефиниције зове се поглед (енг. *View*). То је текст или дијаграм којим се именују и дефинишу сви локални типови података и везе међу тим типовима, опет у складу са правилима коришћеног модела.



*Слика 14. Архитектура базе података у слојевима* [27]

Комуникација корисника односно апликационог програма и *DBMS*-а одвија се помоћу посебних језика. Ти језици традиционално се деле на следеће категорије:

* Језик за опис података (енг. *Data Description Language - DDL*);
* Језик за манипулисање подацима (енг. *Data Manipulation Language - DML*);
* Језик за постављање упита (енг. *Query Language - QL*).

Оваква подела на три језика данас је већ прилично застарела. Наиме, код релационих база постоји тенденција да се сва три језика обједине у један свеобухватни. Пример таквог интегрисаног језика за релационе базе је *SQL*, који служи за дефинисање података, манипулисање и претраживање.

### 2.6.1 *SQL*

*Structure Query Language (SQL)* [25]је релациони упитни језик. Релације се креирају једном наредбом и одмах су доступне, што га чини једноставним за коришћење. Униформан је, јер се сви подаци и резултати операције приказују у виду табела и омогућавају интерактивно и класично програмирање. У основи, то је језик који омогућава комуникацију са базама података и управљањем свих података које оне садрже.

SQL стандард разврстава наредбе [28] у следећих 7 категорија:

* Наредбе за шему базе података (енг. *SQL - schema statements*) – за креирање, измену и избацивање шема и објеката шема (*CREATE*, *ALTER*, *DROP*);
* Наредбе за податке (енг. *SQL - data statements*) – за приказ и ажурирање података базе података (*SELECT*, *INSERT*, *UPDATE*, *DELETE*);
* Наредбе за трансакције (енг. *SQL - transaction statements*) – за стартовање, завршавање и постављање параметара за трансакције (*COMMIT*, *ROLLBACK*);
* Наредбе за контролу (енг. *SQL - control statements*) –користе се за контролу извршавања секвенце *SQL* наредби (*CALL*, *RETURN*);
* Наредбе за конекције (енг. *SQL - connection statements*) – за успостављање и прекидање *SQL* конекције (*CONNECT*, *DISCONNECT*);
* Наредбе за сесију (енг. *SQL - session statements*) – за постављање иницијалне вредности и других параметара *SQL* сесије (*SET*);
* Наредбе за дијагностику (енг. *SQL - diagnostic statements*) – сигнализују изузетке у *SQL* рутинама (*GET DIAGNOSTIC*).

Једна од коришћених наредби у оквиру овог пројекта јесте наредба за постављање упита. Упит се поставља флексибилном наредбом *SELECT* где се резултат упита схвата као нова привремена релација, изведена из сталних. Принцип рада ће се приказати следећим примерима:

Пример 1: Пронаћи бројеве и имена свих студената на нивоу 1

**SELECT** *SNO*, *SNAME*

**FROM** *STUDENT*

**WHERE** *LEVEL* = 1;

Пример 2: Пронаћи бројеве и имена студената који су положили испит (предмет) са шифром 121

**SELECT** *STUDENT.SNO*, *STUDENT.SNAME*

**FROM** *STUDENT*, *REPORT*

**WHERE** *STUDENT.SNO* = *REPORT.SNO*

**AND** *REPORT.CNO* = 121;

Овде су имена атрибута проширена, тако што је у префиксу наведена база (табела) из које се извлачи податак, да би се избегла двозначност. Може се приметити да *SQL* **SELECT**наредба замењује природни спој из релационе алгебре.

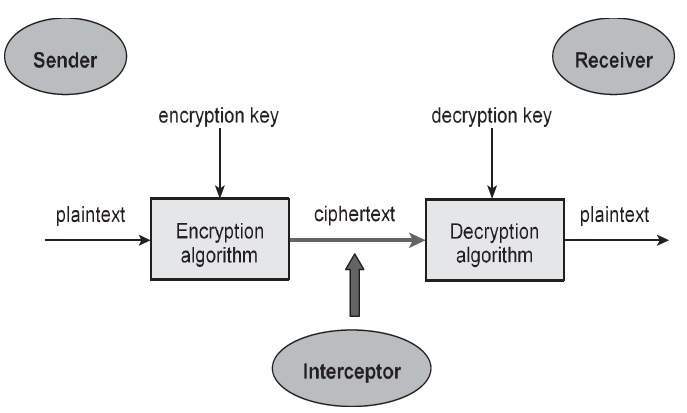
## 2.7 Криптографија

Криптографија [29] је наука која се бави методама очувања тајности информација. Када се личне, финансијске, војне или информације државне безбедности преносе са места на место, оне постају рањиве на прислушкивачке тактике. Овакви проблеми се могу избећи криптовањем (шифровањем) информација које их чине недоступним нежељеној страни. Шифра и дигитални потпис су криптографске технике које се користе да би се имплементирали безбедносни сервиси. Основни елемент који се користи назива се шифарски систем или алгоритам шифровања. Сваки шифарски систем обухвата пар трансформација података, које се називају шифровање и дешифровање.

* Шифровање (енкрипција, енг. *Encryption*) – процедура која трансформише оригиналну информацију (отворени текст, енг. *Plain text*) у шифроване податке (шифрат, енг. *Cipher text*);
* Дешифровање (декрипција, енг. *Decryption*) – реконструише отворени текст на основу шифрата.

Приликом шифровања, поред отвореног текста, користи се једна независна вредност која се назива кључ шифровања (енкрипциони кључ, енг. *Encryption key*), док трансформација за дешифровање користи кључ дешифровања (декрипциони кључ, енг. *Decryption key*). Број симбола који представљају кључ (дужина кључа) зависи од шифарског система и представља један од параметара сигурности тог система.

Криптоанализа је наука која се бави разбијањем шифри, односно откривањем садржаја отвореног текста на основу шифрата, а без познавања кључа. Различите технике криптоанализе називају се напади.



*Слика 15. Блок шема криптографије* [30]

Криптографија мора да обезбеди следеће:

* Интегритет података (енг. *Data integrity*) – брине се о томе да не дође до неовлашћене промене информација као што су мењање, брисање и замена информација. Да би се осигурала веродостојност, мора постојати начин провере да ли је информација промењена од стране неовлашћене особе;
* Тајност (енг. *Confidentiality*) информација – осигурава да је садржај информације доступан само овлашћеним особама односно само онима који поседују кључ. Постоје бројни начини заштите тајности, почев од физичке заштите до математичких алгоритама који скривају податке;
* Провера идентитета (енг.*Authentication*) – пре почетка комуникације стране које комуницирају морају да се представе једна другој и буду сигурне са киме комуницирају;
* Немогућност избегавања одговорности (енг. *Non-repudiation*) – постаје изузетно значајна у последње време када се електронским путем врше трансакције новца.

Постоје две врсте енкрипције:

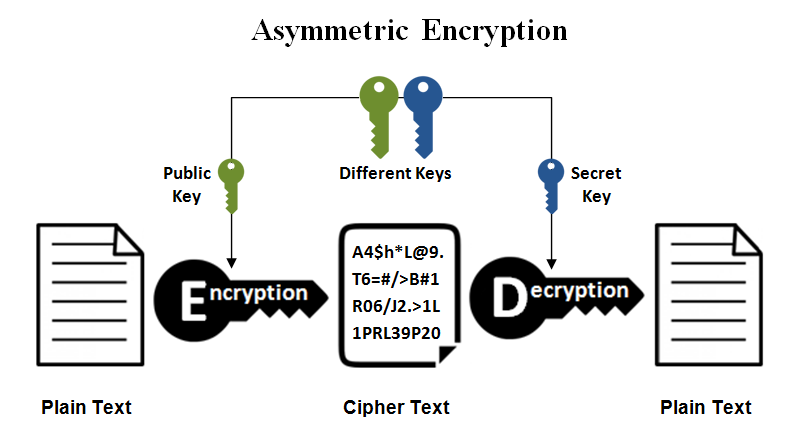
* Асиметрична енкрипција (енг. *Asymmetric encryption*);
* Симетрична енкрипција (енг. *Symmetric encryption*).

### 2.7.1 Асиметрична енкрипција

Асиметрична енкрипција [29] користи два кључа – јавни и приватни. У исто време се праве приватни и одговарајући јавни кључ. Јавни кључ се даје апликацијама које шаљу шифроване податке, помоћу кога шифрују поруку коју желе да пошаљу. Када прималац добије шифрат, дешифрује га помоћу свог приватног кључа. На тај начин сваки прималац има свој приватни кључ а јавни се може дати било коме, пошто се он користи за шифровање, а не и дешифровање.

Заштита јавног кључа није потребна јер је јавно доступан и може се проследити путем интернета. Асиметрични кључ има далеко бољу моћ у обезбеђивању сигурности информација које се преносе током комуникација. Асиметрично шифровање се углавном користи у свакодневним комуникационим каналима, нарочито преко интернета, али за слање мање количине података због брзине која је доста спорија од симетричне енкрипције.

Популарни алгоритам за шифровање асиметричног кључа укључује *ElGamal*, *RSA*, *DSA*, *Elliptic curve techniques*, *PKCS*.



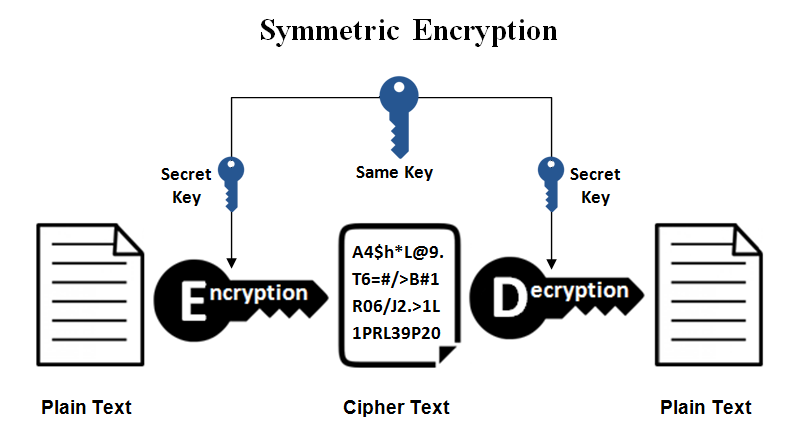
*Слика 16. Илустрација асиметричне енкрипције* [31]

### 2.7.2 Симетрична енкрипција

За разлику од асиметричне, симетрична енкрипција [29] користи исти кључ и за шифровање и за дешифровање. Баш због тога је разноврсност, а самим тим и сигурност алгоритама овакве енкрипције велика. Битан фактор је и брзина – брзина симетричне енкрипције је и до 1000 пута већа у поређењу са асиметричном енкрипцијом. Користи тајни кључ који може бити број, реч или низ случајних слова.

Поред свих предности које има на пољу сигурности и брзине алгоритма, симетрична енкрипција поседује велики недостатак у виду дилеме како пренети тајни кључ. Проблем је у томе што уколико се тајни кључ пресретне порука се може прочитати. Због тога се овај тип енкрипције најчешће користи приликом заштите података који се не деле са другим странама, већ је шифра позната на једној страни те шифру није потребно слати ка другој.

Примери симетричног шифровања су *Blowfish*, *AES*, *RC4*, *DES*, *RC5* и *RC6*, а најчешће коришћени симетрични алгоритам је *AES-128*, *AES-192* и *AES-256*.



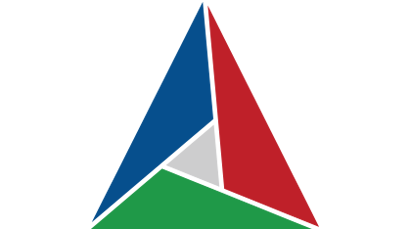
*Слика 17. Илустрација симетричне енкрипције* [31]

# 3. Алати и *3rd-party* библиотеке

У оквиру овог поглавља описани су алати и библиотеке треће стране које су коришћене приликом реализације система.

## 3.1 *CMake*

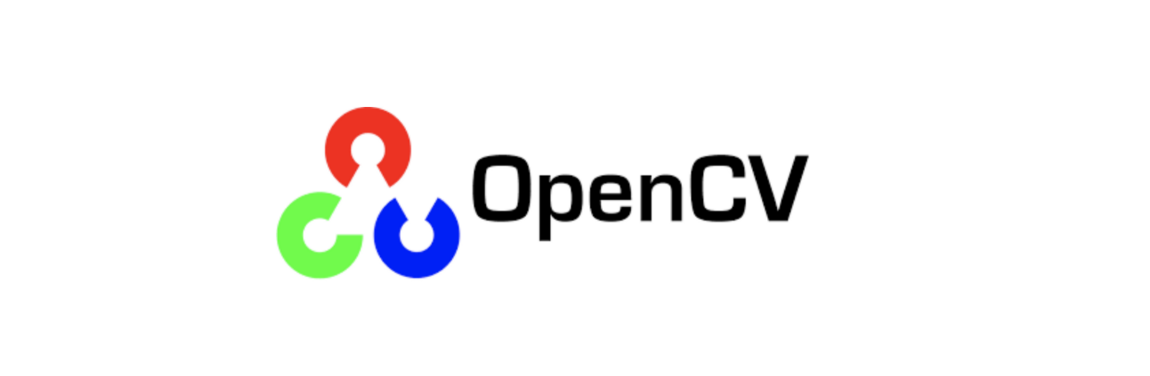
*CMake* [32] је породица алата дизајнирана за *build*, тестирање и паковање софтвера. У питању је *Cross-Platform* алат отвореног кода (енг. *Open Source*). Користи се да контролише процес компајлирања софтвера коришћењем једноставних конфигурационих фајлова који су независни од платформе или компајлера и да генерише изворне *Makefile* рецепте и радне просторе који могу бити кориштени у компајлерском окружењу по избору корисника. Пакет *CMake* алата креиран је од стране *Kitware* услед потребе за моћним, универзалним *build* окружењем за пројекте отвореног кода као што су *ITK* и *VTK*.



*Слика 18. CMake лого* [32]

## 3.2 *OpenCV*

*OpenCV* [33] представља колекцију библиотека писаних у C/C++, Python и Java програмском језику које се првенствено користе за компјутерску визију у реалном времену. Првобитно ју је развила компанија Intel, а данас је доступна као софтвер отвореног кода. Неке од примена ових библиотека укључују: сегментацију, праћење покрета, препознавање лица, препознавање гестова, померање камера унутар 3D простора (енг. *Egomotion*), али и препознавање објеката помоћу тренираних неуронских мрежа.



*Слика 19. OpenCV лого* [33]

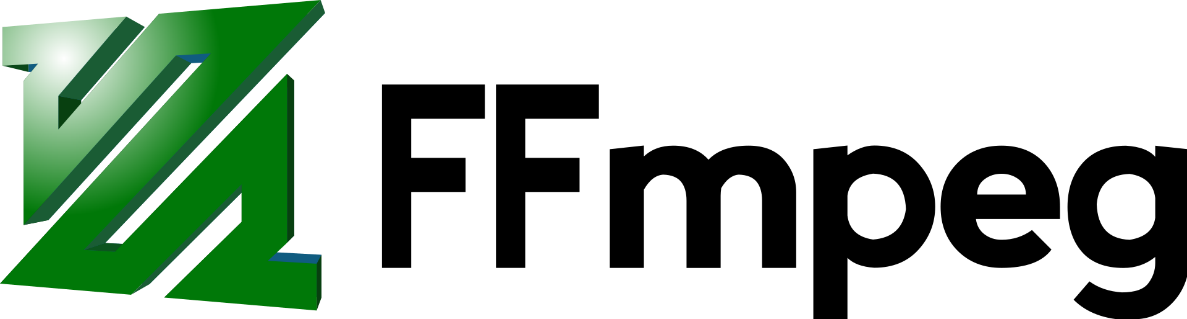
## 3.3 *FFmpeg*

*FFmpeg* [34]је бесплатни софтвер пакет отвореног кода који се састоји од скупа библиотека и програма за руковање видео, аудио и другиx мултимедијалниx датотека и токова. *FFmpeg* је алат дизајниран за обраду видео и аудио датотека. Широко се користи за транскодирање формата, основно уређивање (скраћивање и спајање), скалирање видео записа, видео постпродукцијске ефекте и усклађеност са стандардима (*SMPTE*, *ITU*).

*FFmpeg* такође укључује и друге алате командне линије:

* *Ffplay* – једноставан *Media Player* који користи *SDL* и *FFmpeg* библиотеке;
* *Ffprobe* – алатка за приказ информација о медијима (текст, *CSV*, *XML*, *JSON*).

*FFmpeg* библиотеке су основни део софтверских медијских плејера као што је *VLC*, и укључене су у основну обраду за *YouTube.* Обухвата кодере и декодере за многе аудио и видео формате, што га чини изузетно корисним.



*Слика 20. FFmpeg лого* [34]

## 3.4 *Qt*

*Qt* [35] је *Cross-platform* радно окружење (*Framework*) за развијање *Desktop*, ембедед или мобилних апликација. Платформе које подржава су: *Linux*, *OS X*, *Windows*, *VxWorks*, *QNX*, *Android*, *iOS*, *BlackBerry*, *Sailfish OS* и други.

*Qt* није програмски језик сам по себи, него *framework* написан у програмском језику *C++*. Поседује препроцесор – *MOC* (*Meta-Object Compiler*), који проширује *C++* програмски језик са функционалностима као што су сигнали и слотови (резервисане речи *signals* и *slots*). Пре компајлирања, *MOC* парсира *source* фајлове написане у *Qt*-овом продуженом *C++* „програмском језику“ и генерише стандардне компајлирајуће *C++ source* фајлове из њих. *Qt* *framework*, као и апликације или библиотеке које га користе могу се компајлирати на било ком стандардном *C++* компајлеру, као што су: *GCC*, *Clang*, *ICC*, *MinGW* и *MSVC*.



*Слика 21. Qt лого* [36]

## 3.5 *MySQL*

*MySQL* [37] је релациони систем за управљање базама података - *RDBMS*. Име је добио комбинацијом префикса *My*, што представља име ћерке Мајка Видениуса, једног од оснивача компаније и добро познатог акронима - *SQL*.

*MySQL* је бесплатан и софтвер отвореног кода под условима *GNU* лиценце, али је доступан и у варијацијама власничких лиценци. *MySQL* je основан и спонзорисан од стране шведске компаније *MySQL AB*, која је купљена од стране *Sun Microsystems* (данашњи *Oracle*).

*MySQL* поседује самосталне (*standalone*) клијенте који корисницима омогућавају директну интеракцију са *MySQL* базом података користећи *SQL* језик, али чешће се, овај релациони систем, користи у оквиру других програма да би се имплементирале апликације којима су потребне релационе базе. Између осталог, *MySQL* поседује *API* написан на *C* програмском језику.



*Слика 22. MySQL лого* [38]

## 3.6 *Crypto++*

*Crypto++* [39] (такође познат и као *CryptoPP*) је бесплатна библиотека отвореног кода која садржи класе које описују криптографске алгоритме и шеме написана од стране *Wei Dai*. *Crypto++* се нашироко користи у академским и студентским пројектима, некомерцијалним и пројектима отвореног кода, али и за комерцијалне пројекте. Публикован је 1995. године.

Библиотека потпуно подржава 32-битне и 64-битне архитектуре за многе веће оперативне системе и платформе, укључујући *Android*, *Apple*, *BSD*, *Cygwin*, *IBM AIX*, *S/390*, *Linux*, *MinGW*, *Solaris*, *Windows*, *Windows Phone* и *Windows RT*. Библиотека подржава *C++* стандарде користећи *C++03*, *C++11*, *C++14* и *C++17* *runtime* библиотеке и различите компајлере, укључујући *GCC*, *ICC*, *Clang*, *MSVC*, *Borland Turbo C++* итд.



*Слика 23. Crypto++ лого* [40]

# 4. Концепт решења

У овом поглављу описан је и илустрован концепт решења, идеја пројекта, као и архитектура и функционалност појединачних модула система.

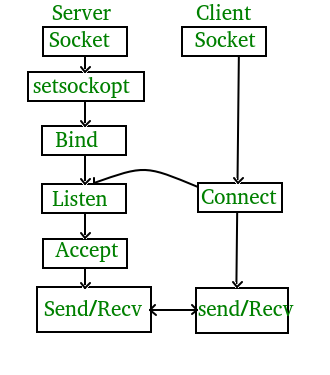
## 4.1 Библиотека утичнице у објектно-оријентисаној форми

Интерфејс за програмирање апликације (енг. *Application Programming Interface*) - *API* за утичницу (енг. *Socket*) представља срце комуникације међу процесима или уређајима, старајући се о ниским нивоима (енг. *Layer*) комуникације као што су интернет протокол или ниже од тога, остављајући кориснику могућност да се стара и размишља само о комуникацији на апликативном нивоу.

Не постоји заједнички интерфејс за оперативне системе, али су ове библиотеке за *Windows* и *Linux* поприлично сродне, те се може приметити доста заједничких структура и функција на обе стране, иако постоје минорне различитости између њих.

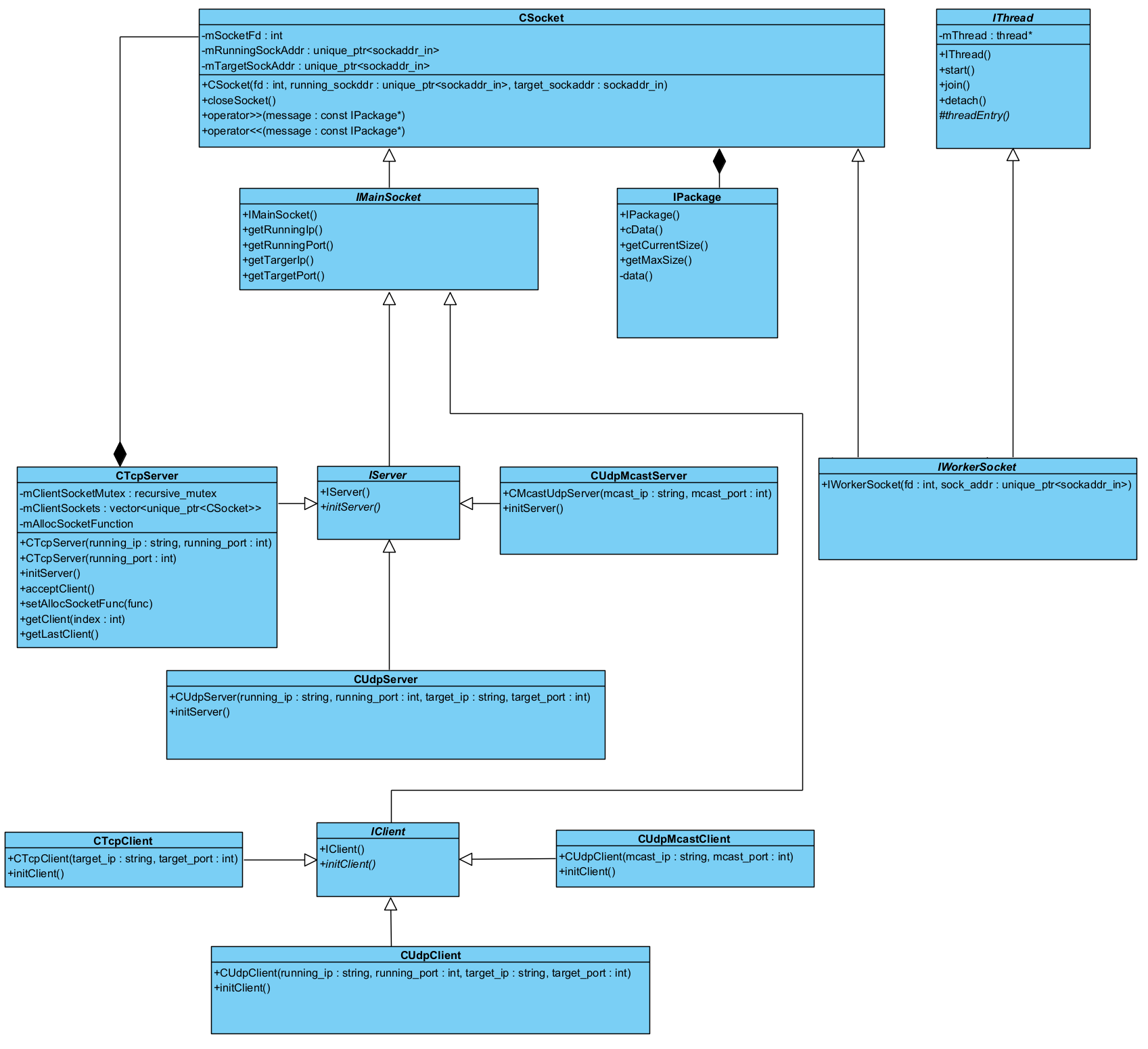
Иако веома неопходан и готово свеприсутан, овај *API* написан је у програмском језику *C*, те, као такав, није комфоран за коришћење. Идеја је била прилагодити библиотеку објектно-оријентисаној парадигми паковањем њених функција и структура у класе познатије као *Wrapper* класе. На тај начин, поред сажетог и читког кода, добија се заједнички интерфејс за оба оперативна система, који се лако може користити у оквиру било које апликације.

На овај начин је ограничено додатно конфигурисање утичнице које *API* пружа. Међутим, познавајући чињеницу да се у већини сценарија интерфејс користи на идентичан начин добија се стандардизовани *C++* *API* за коришћење утичнице.



*Слика 24. Уобичајени сценарио коришћења утичнице у TCP* [41]

Интерфејс се састоји од 3 пара класа које је могуће користити – инстанцирати. У питању су парови – клијент и сервер, а 3 пара покривају *TCP*, затим једнозначни и вишезначни *UDP IP*.



*Слика 25. UML класни дијаграм Wrapper библиотеке*

Свака од наведених класа поседује искључиво конструктор са параметрима, те је приликом њиховог инстанцирања неопходно проследити одговарајуће аргументе – то су заправо *IP* адреса и порт који ће бити додељени утичници објекта и/или адреса и порт које очекује са друге стране. Поред тога, пре почетка рада, неопходно је урадити иницијализацију објекта позивом неопходне методе – за клијентске објекте то је метода *InitClient*, док је за серверске *InitServer*. Методу за иницијализацију потребно је позвати у оквиру *try/catch* блока, јер метода баца изузетке (енг. *Throw exception*) приликом неуспеха иницијализације. Уколико се то не учини, али ипак дође до грешке – долази до терминације апликације.

Све споменуте класе наслеђују основну класу – *CSocket*, у којој су написане методе за пријем/слање поруке. Методе су написанe у виду оператора >> (за пријем података) и << (за слање података). Објектима класе *CTcpServer* је забрањено коришћење тих метода, јер његова утичница за улогу има прихватање нових клијената, а не непосредно комуницирање са другом страном. Прихватање клијената извршава се помоћу методе *acceptClient*. У питању је блокирајућа функција, која блокира извршавајућу нит све док се не успостави веза са новим клијентом. Приликом успостављања везе са клијентом, према иницијалним подешавањима, алоцира се нови објекат типа *CSocket*, који се даље може користи за интеракцију са повезаним клијентом. Уколико корисник жели другачије, могуће је алоцирати неку другу класу изведену из базне класе, мењањем показивачке функције за алоцирање.

Поред наведених класа постоје и апстрактне класе које је могуће извести, додатно дефинисати и у оквиру њих укључити неопходне функционалности. Класа *IPackage* је интерфејс који се користи за слање/пријем порука. Свака изведена класа ове класе аутоматски може да се користи као порука за слање или објекат у којем ће бити смештена примљена порука. У суштини, у оквиру изведене класе потребно је дефинисати (*override*) чисте виртуалне (енг. *Pure virtual*) методе које заправо представљају тзв. *Getter* методе за максималну величину, искоришћену величину, као и стартну адресу низа поруке, које су потребне за акције пријема и слања порука.

Од апстрактних класа, такође, интересантна је и класа *IWorkerSocket*, који наслеђује *CSocket* и *IThread* класе и на тај начин кориснику пружа алоцирање утичнице која ће моћи да извршава дефинисану петљу приликом алоцирања/стартовања његове нити. Важно је напоменути да се класе изведене из овог интерфејса могу алоцирати приликом *acceptClient* методе *CTcpServer* објекта, променом показивачке функије за алоцирање клијентске утичнице.

Поред тога, овај *API* пружа помоћне функције у виду генерисања неких битних података. Такав је генератор слободних портова, који попуњава у свој стек све искоришћене портове и генерише слободни насумични (енг. *Random*) порт. По сличном принципу обезбеђује се и вишезначна *IPv4* адреса. Такође, имплементирана је и функција која пружа *IPv4* адресу машине на којој се функција извршава у локалној мрежи.

## 4.2 Архитектура и функционалност серверске апликације

У оквиру овог пројекта серверска апликација има за улогу да обавља велики број функционалности те се може поделити у 2 велике целине - модула:

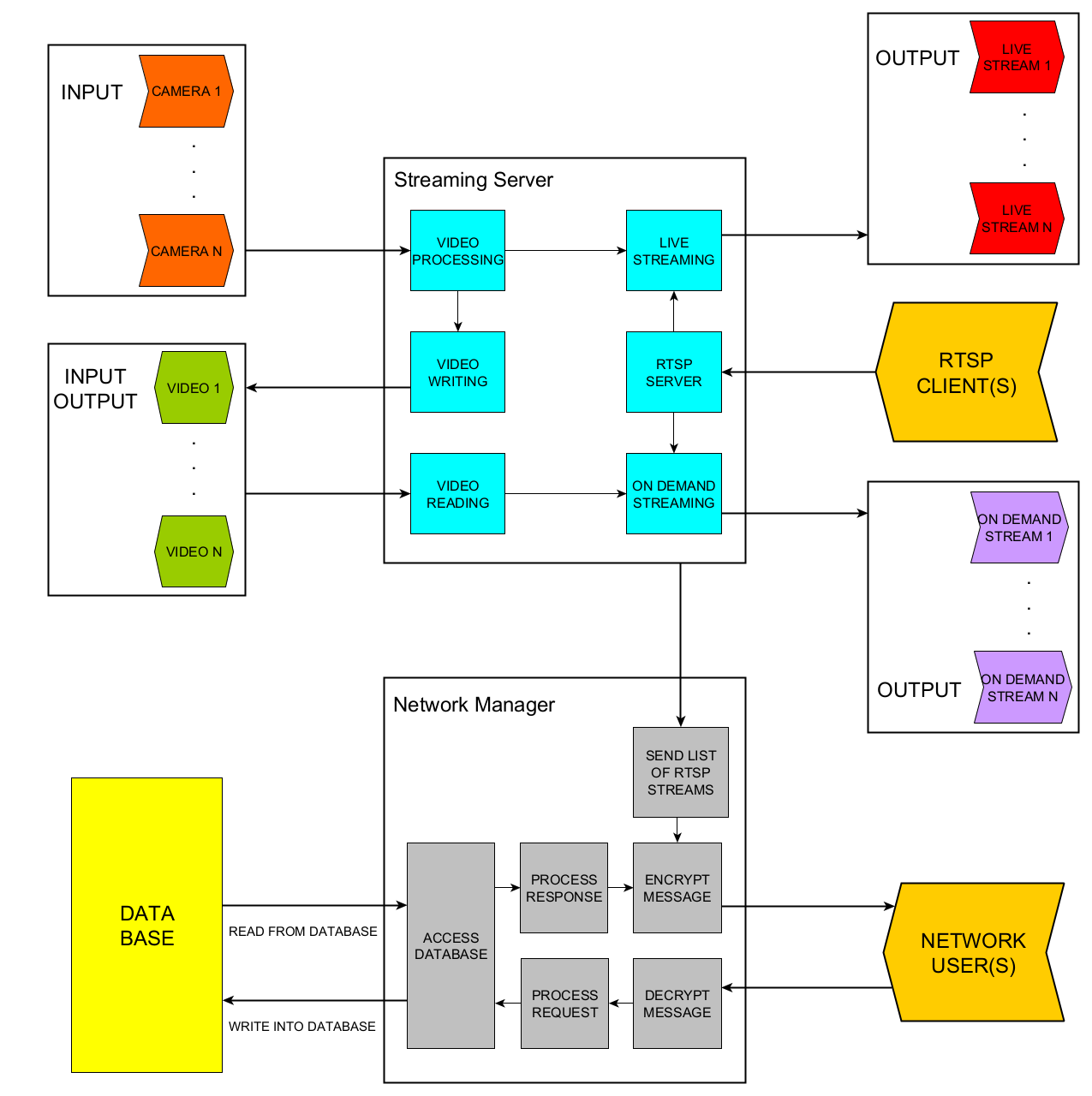
* Сервер за видео пренос – модул који је задужен за целокупно видео процесуирање, интеракцију са клијентима видео преноса и услуживање видео преноса;

1. Видео процесуирање подразумева преузимање слике са камере и додатну обраду за живи пренос, те уписивање снимка у видео запис како би се касније могао користити за видео преносе по захтеву;
2. Интеракција са клијентима видео преноса подразумева комуникацију по *RTSP* протоколу;
3. Услуживање видео преноса обухвата омогућавање константног живог видео преноса уколико је барем један клијент присутан, као и отварање засебних сесија са клијентима за видео пренос по захтеву;

* Мрежни менаџер – модул који је задужен за комуникацију са мрежним корисницима, одржавање и приступање бази података, као и заштиту личних података корисника;

1. У оквиру комуникације са мрежним корисницима размењују се захтеви за регистрацију или пријаву корисника и одговори менаџера, и коначно прослеђује се листа постојећих *RTSP* преноса, коју је потребно благовремено ажурирати;
2. Одржавање и приступање бази података обухвата приступање табели постојећих корисника, те уписивање нових, уз проверу да ли је неки од података већ искоришћен код постојећих корисника;
3. Овај део комуникације је изузетно осетљив, јер се приступа приватним подацима корисника – корисничко име, *e-mail* адреса и шифра, с тога је потребно те податке заштитити.

На следећој слици приказана је идејна блок шема серверске апликације.



*Слика 26. Идејна блок шема серверске апликације*

## 4.3 Архитектура и функционалност клијентске апликације

У оквиру овог пројекта, у функционалном смислу, клијентска апликације може се поделити у 3 веће целине - модула:

* *RTSP* клијент – модул који је задужен за:

1. повезивање са *RTSP* сервером и комуницирање по *RTSP* протоколу;
2. потраживање живог или преноса по захтеву;
3. прекидање везе са *RTSP* сервером или пребацивање на неки други живи или пренос по захтеву;

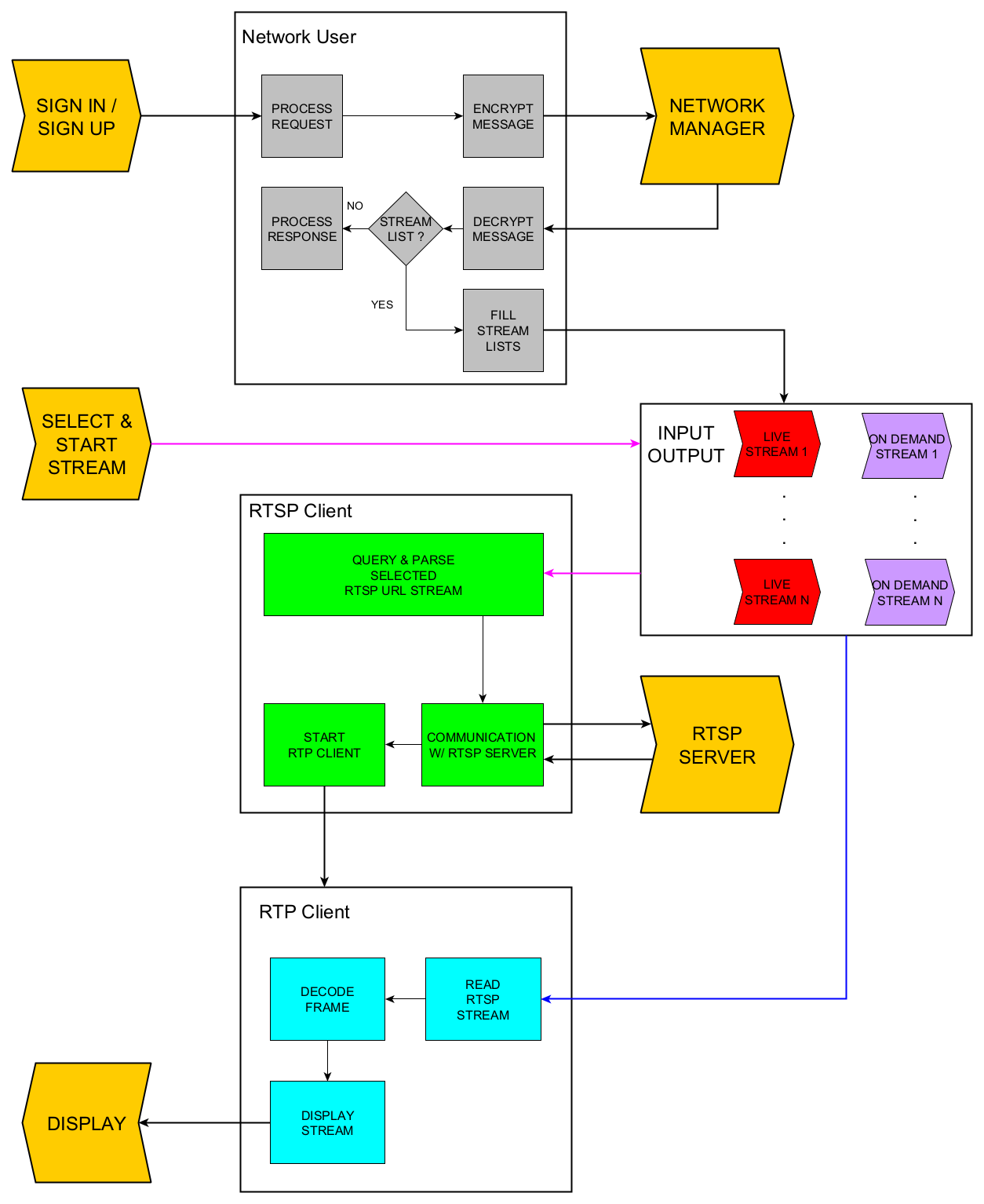
* *RTP* клијент – модул који је задужен за:

1. повезивање на пренос на основу инструкција које добије од *RTSP* клијента – протокол, *IP* адреса и порт преноса на који се повезује;
2. пријем и колекцију пристиглих пакета у резултујући *frame*;
3. декодовање и приказивање видео преноса;

* Мрежни корисник – модул који је задужен за:

1. повезивање са мрежним менаџером и евидентирање у мрежи система;
2. енкрипцију послатих и декрипцију примљених порука;
3. благовремено ажурирање активних видео преноса.

На следећој слици приказана је идејна блок шема клијентске апликације.



*Слика 27. Идејна блок шема клијентске апликације*

# 5. Програмско решење

У овом поглављу описано је програмско решење пројекта - функционалност и рад појединачних модула система. Програмско решење описано је у оквиру два поднаслова, чиме се раздваја програмско решење серверске и клијентске апликације.

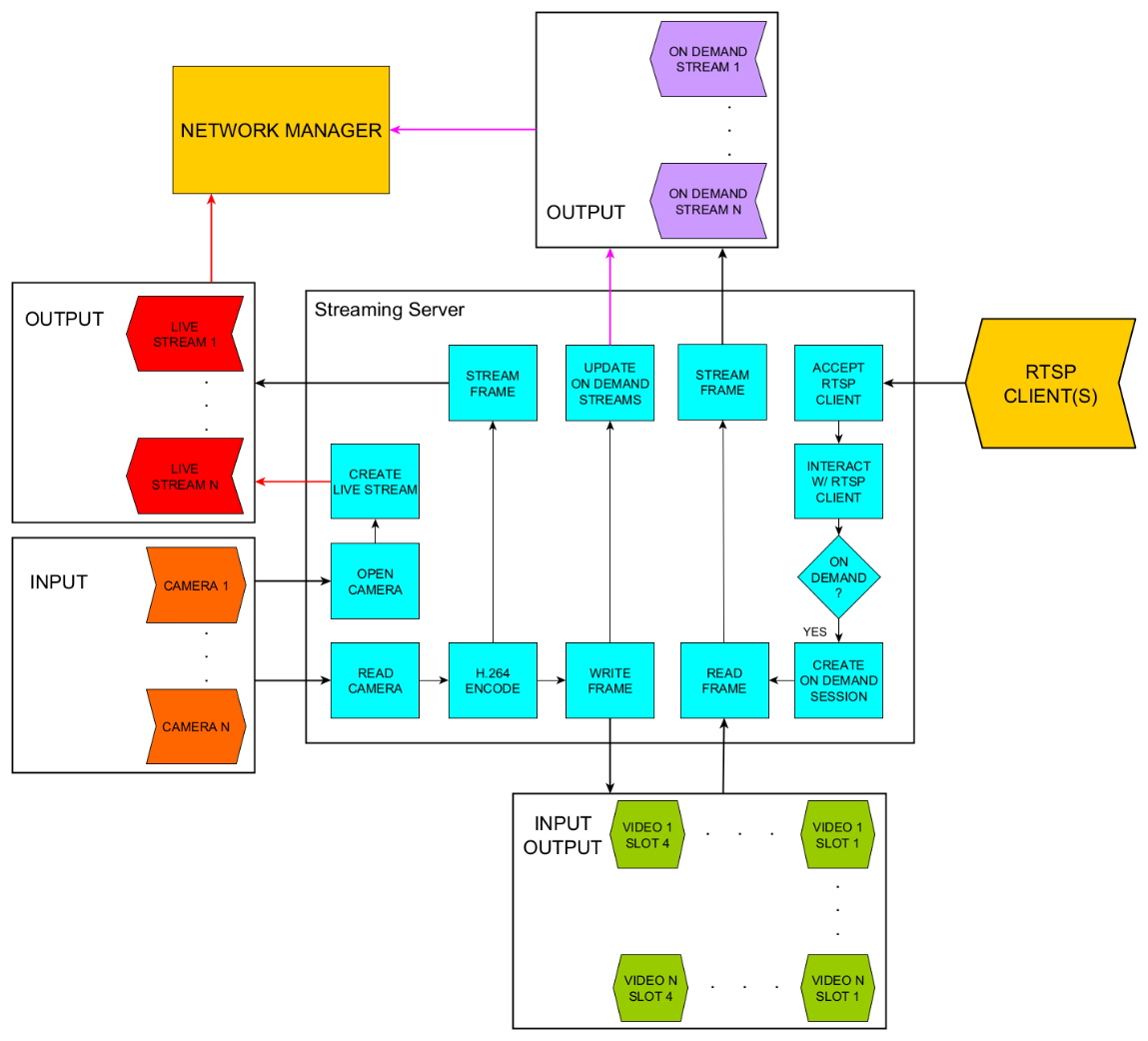
## 5.1 Програмско решење серверске апликације

На основу концепта решења, може се закључити да је серверска апликација у оквиру овог трослојног система подељена у два главна модула:

* Сервер за видео пренос;
* Мрежни менаџер.

### 5**.1.1 Сервер за видео пренос**

Овај модул је, како му само име каже, одговоран за видео пренос. Међутим његове активности превазилазе те оквире, па је тако задужен за целокупно видео процесуирање, интеракцију са клијентима видео преноса и услуживање видео преноса – вишезначно и једнозначно. На следећој слици приказана је детаљнија блок шема сервера за видео пренос.



*Слика 28. Блок дијаграм сервера за видео пренос*

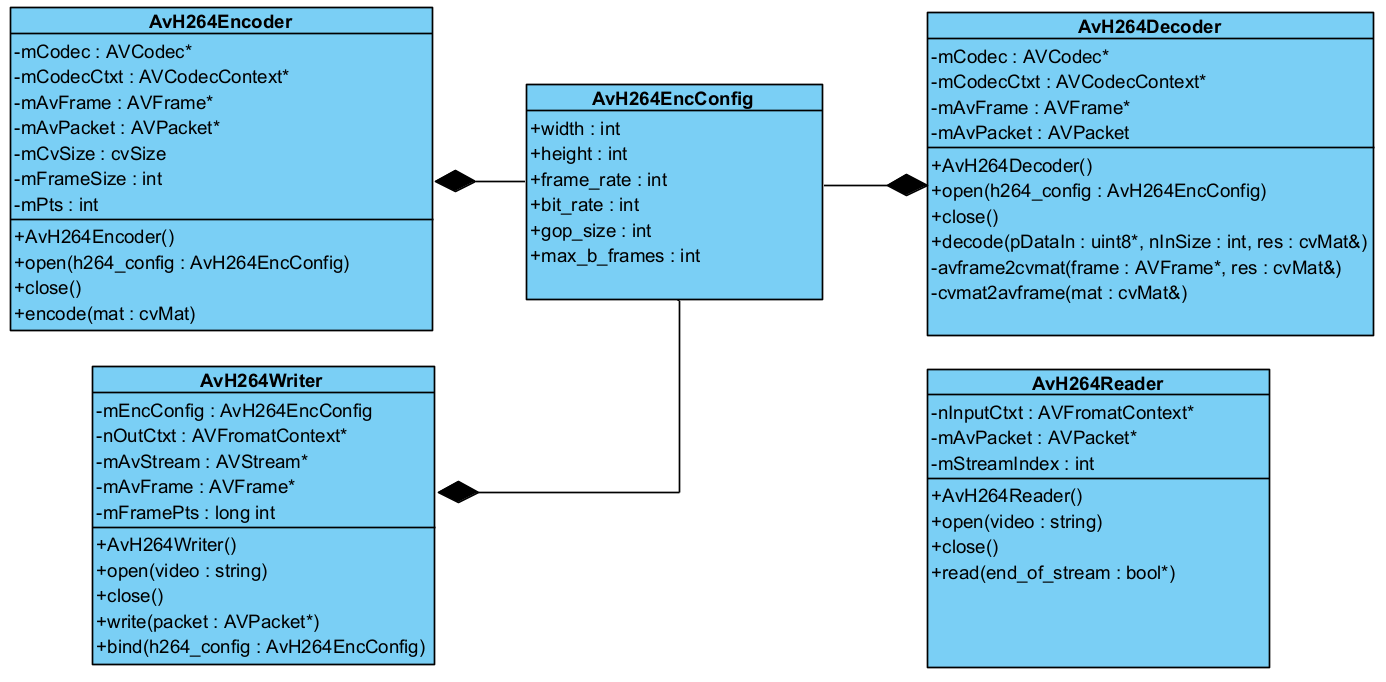
#### 5.1.1.1 Читање са камере и уписивање у видео запис

Серверска апликација при покретању зна којим бројем камера ће управљати, односно читати са њих. Да би управљала камером, апликација користи објекте из *OpenCV* библиотеке. Приликом иницијализације, неопходно је инстанцирати објекат класе *cv::VideoCapture*, којем се прослеђује редни индекс камере која је физички повезана и евидентирана у машини на којој се серверска апликација извршава. Индексирање уређаја креће од нуле.

Након што је уређај успешно отворен, креира се и покреће посебна нит у оквиру које се извршава петља читања све до краја рада серверске апликације или евентуалног физичког прекида везе са камером. У оквиру петље за складиштење података користи се објекат класе *cv::Mat*. Преузета слика у матричном објекту прослеђује се на додатну обраду у виду додавања исписа тачног времена у којем је слика преузета. Време је написано у формату: дан у недељи, датум, сати, минуте и секунде.

Објекат *cv::Mat* складишти слику у сировом формату. Сирова слика може да досеже огромне величине. За слику резолуције 640 x 480 добија се 307.200 пиксела, што множењем са бројем основих боја из *RGB* система резултује у 921.600 бајта, односно 900 *kB*. Очигледно је да слика у оваквом формату није практична за манипулисање, складиштење или видео пренос путем мреже.

Због тога се након преузимања слике са камере примењује енкодовање садржаја према *H.264* кодеку. Овај механизам реализује се уз помоћ библиотеке *FFmpeg*. Попут *Wrapper*-а за библиотеку утичнице, примењен је идентичан приступ за ову библиотеку написану на *C* програмском језику. *Wrapper* се састоји из 5 класа и свака има јединствену намену. Класе покривају *H.264* енкодовање и декодовање, као и уписивање и читање *H.264* видео записа, док је пета класа заправо структура која служи за *H.264* поставке. На следећој слици приказане су релације између класа и њихова *UML* репрезентација.



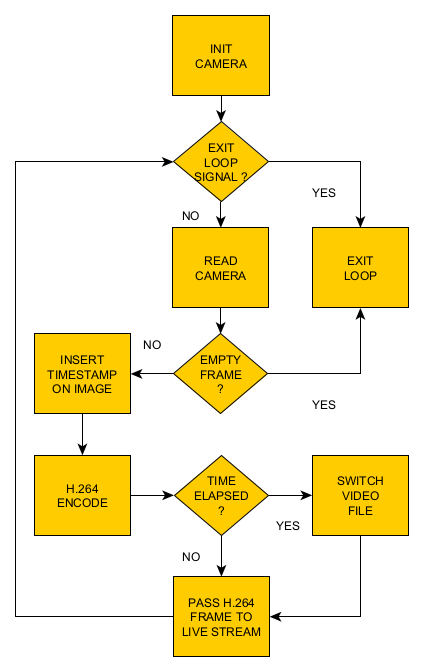
*Слика 29. UML класни дијаграм за H.264 Streaming Wrapper*

Након што је сирови *frame* камере прошао кроз *H.264* енкодер добија се *frame* који је мањи, а његова величина варира у зависности од промена које је ухватила камера. Највећа величина до које досеже је око 70 *kB*. Слика уистину, услед кодирања губи на квалитету, који је у овом случају занемарљив јер компресовани *frame* је минимално 13 пута мањи у односу на сирови.

Новонастали *frame* се прослеђује објекту класе *AvH264Writer* који има улогу видео записивача. Генерисање видео записа функционише тако да сваки фајл не треба бити дужи од 15 минута. У моменту када то време истекне, обавештава се мрежни менаџер о настанку новог видео записа и ажурирају се листе доступних видео преноса по захтеву. Евиденција о протеклом времену од 15 минута извршава се уз помоћ специјалне нити која има за улогу да мери протекло време и обавести другу нит о томе да је време истекло, те да је потребно креирати нови видео фајл и надаље у њему извршавати видео записивање.

Креирање новог видео записа функционише по принципу кружног попуњавања постојећих фајлова када постану застарели, како би се избегло бесконачно попуњавање ограничене меморије. С обзиром да постоје 4 видео записа од по 15 минута који заједно представљају протеклих сат времена до следеће итерације, постоји могућност да корисник користи застарели фајл преко видео преноса по захтеву. Услед тога видео записивач неће бити у могућности да пресними коришћени фајл. Овај проблем решен је тиме што за сваки постојећи видео пренос по захтеву постоје два фајла која се наизменично користе за уписивање. Од тренутка када је видео записивач завршио записивање у једном фајлу тада фајл постаје доступан корисницима за видео пренос по захтеву. Његов видео запис је „свеж“ све до тренутка када поново треба да се уписује у њега, али је доступан за преглед једну итерацију више (15 минута) све док видео записивач извршава записивање у његовом пару. Од тренутка када је записивање у његовом пару завршено, парни фајл постаје доступан корисницима за преглед, а застарели фајл постаје резервисан за преписивање 3 итерације касније. До проласка 3 итерације сигурно долази до прекида репродукције застарелог фајла, јер је у етру нови (парни) фајл, те нема бојазни да ће бити коришћен у некој сесији за видео пренос по захтеву. На тај начин, постоји 8 видео записа од по 15 минута за 4 видео преноса по захтеву који се циклично попуњавају, што и даље резултује количином утрошене меморије која се толерише.

Уписани видео *frame* се даље прослеђује на пренос уживо, што ће бити описано у оквиру поднаслова за *RTP* вишезначни видео пренос.



*Слика 30. Алгоритам рада читања са камере и видео записивања*

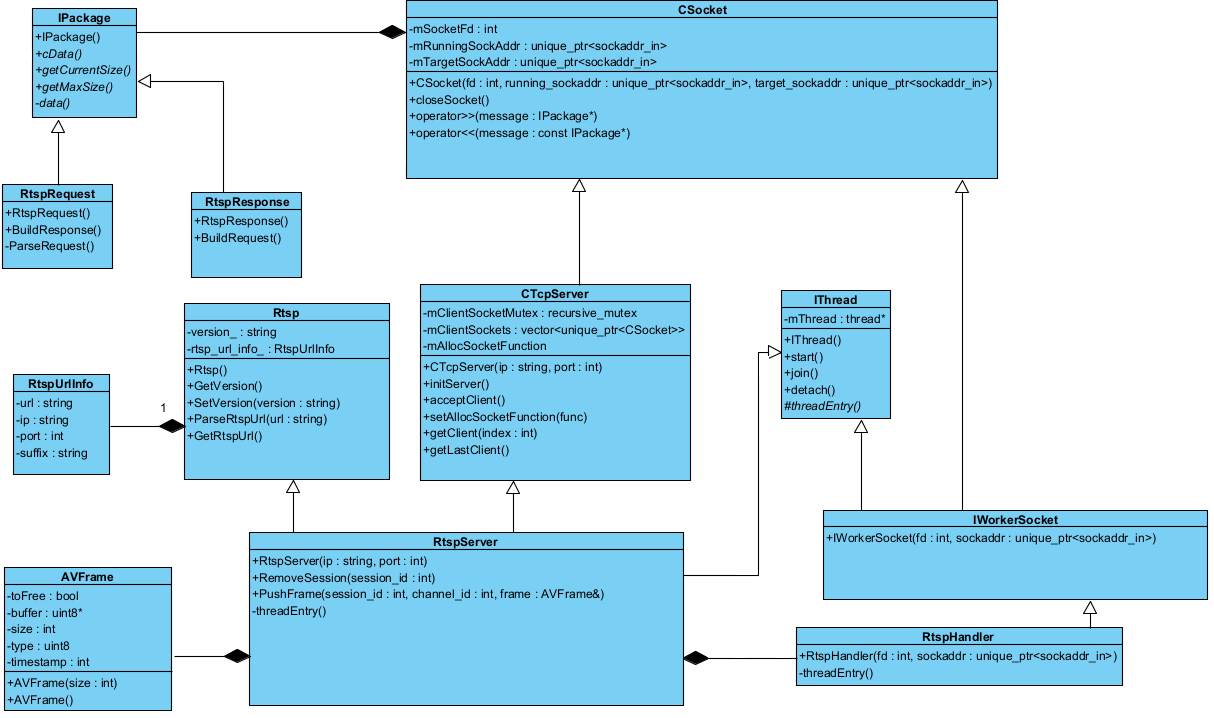
#### 5.1.1.2 *RTSP* интеракција са клијентима

Сервер за видео пренос пружа пренос искључиво по *RTSP* протоколу. Овај протокол захтева од корисника видео преноса да, поред утичнице за добављање видео садржаја, поседује и утичницу за интеракцију са сервером. Одатле се може закључити да серверска страна, попут клијентске, мора поседовати минимално две утичнице за пружање једног видео преноса, с тим што више преноса могу да деле једну утичницу за интеракцију. Прва утичница је заправо *RTSP* сервер, док је друга *RTP* сервер, о којем ће више бити писано у наредним поднасловима.

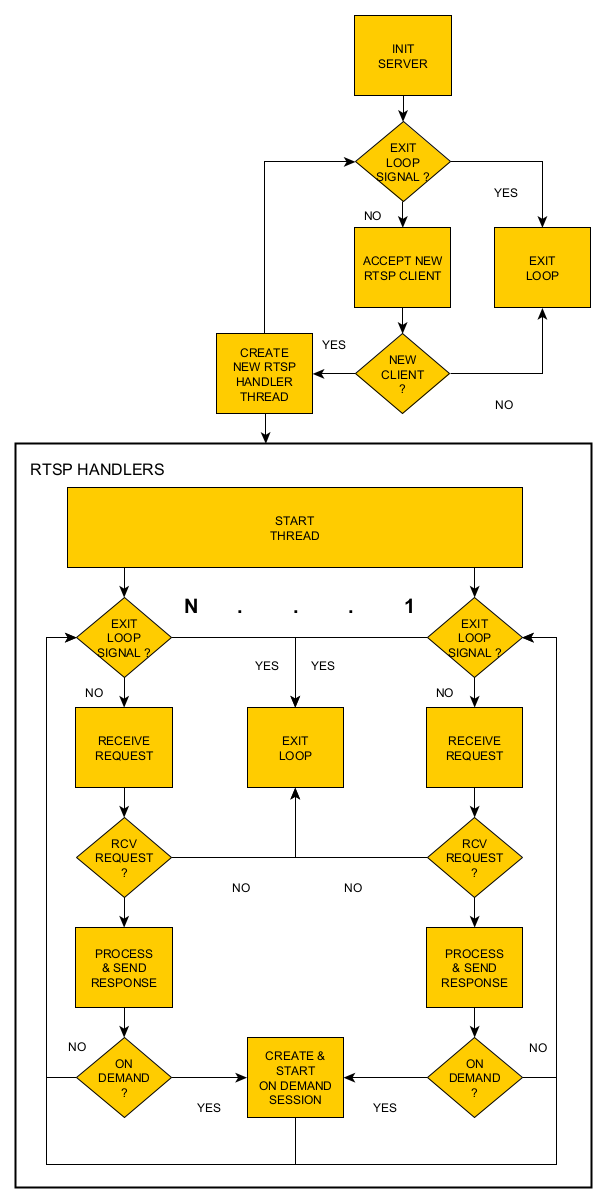
*RTSP* сервер базиран је на *TCP* протоколу, где са сваким повезаним клијентом отвара посебну нит за комуникацију. У оквиру ове утичнице, клијент шаље захтев на који видео пренос жели да се закачи, прослеђујући његов јединствени *URL* линк. Линк се састоји од *IP* адресе и порта *RTSP* сервера, као и назива канала (видео преноса) на крају линка. *RTSP* сервер се може илустровати као телевизор, а клијент као даљински управљач, где се притиском бројева на управљачу добија комбинација на основу које телевизор зна на који канал треба усмерити гледаоца. Комбинација бројева, у *RTSP* протоколу представља *URL* линк, док је усмеравање гледаоца прослеђивање *IP* адресе, порта и протокола видео преноса, односно *RTP* сервера, али и друге важне информације као што је кодек који се користи. Овај протокол омогућава додатне функционалности као што су искључивање (прекид рада), укључивање (поновно успостављање рада) и друге могућности.

Веза између клијента и сервера постоји докле год корисник гледа било који видео пренос. Ова веза обично није динамична већ је у пасивном режиму, где је нит у већини времена блокирана и очекује нови захтев од клијента или његову евентуалну дисконекцију. У том тренутку, нит се затвара, док се ресурси ослобађају за новог клијента.

Дакле, целокупни подмодул – *RTSP* сервер функционише по *Multi-threading* принципу, где постоји нит која прихвата нове клијенте и отвара нову нит за сваког од њих, што је укупно (n+1) нити за n повезаних *RTSP* клијената. Овај број нити не представља проблем, јер нити имају спорадично извршавање и у већини времена су у пасивном - блокирајућем стању услед *receive* функције.



*Слика 31. UML класни дијаграм RTSP сервера*



*Слика 32. Алгоритам рада RTSP сервера*

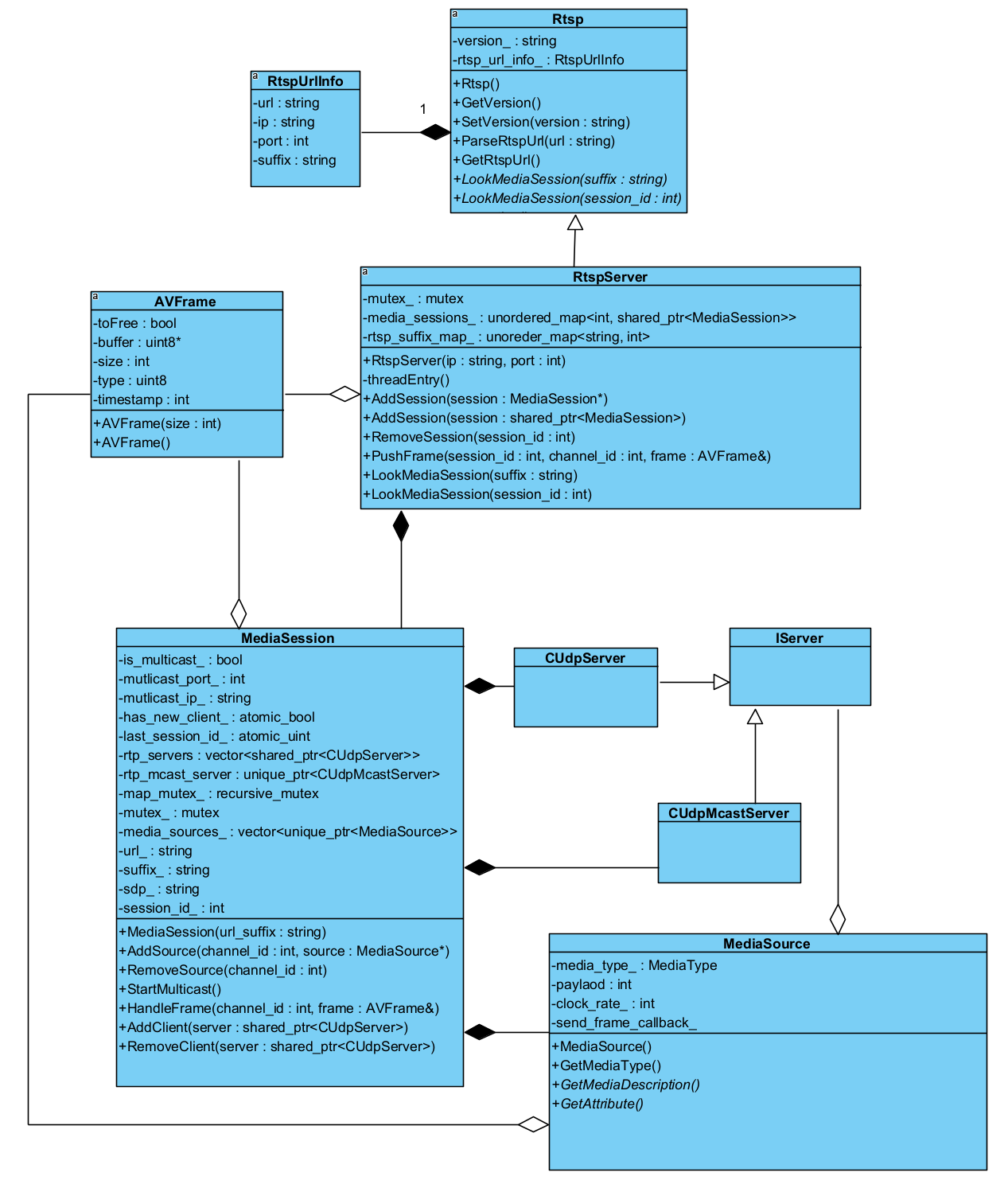
#### 5.1.1.3 *RTP* вишезначни видео пренос

Раније је утврђено да *RTSP* и *RTP* представљају уређени пар серверских утичница за видео пренос, где један без другог не могу. *RTSP* сервер мора да има пар који ће да пласира видео садржај, док *RTP* сервер у већини случаја не постоји или није започео пренос све док овај други није сигнализирао креирање или почетак преноса. Код вишезначног видео преноса то није обавезан случај, јер вишезначни сервер може самостално постојати и преносити видео садржај без познавања чињенице да ли је неки клијент повезан.

Вишезначни сервер поседује јединствену *Multicast* *IP* адресу и порт на којој публикује садржај, па је тако сваки клијент у могућности да без проблема преузме садржај одатле. Међутим, у оквиру овог система то није случај, јер је зарад оптимизованијег рада серверске апликације вишезначни *RTP* сервер упасивном режиму уколико нема нити једног евидентираног корисника, тако да клијент, чак иако би унапред знао све неопходне информације канала, не би могао да гледа видео пренос без претходне интеракције са *RTSP* сервером, под условом да сервер већ није евидентирао неког клијента који користи живи пренос.

Вишезначни видео пренос је, у оквиру овог система, намењен искључиво за пренос уживо, што има смисла, јер живи пренос пласира идентичан садржај ка свим корисницима истовремено. Постојала је могућност пласирања оваквог садржаја путем једнозначног видео преноса, где би се сваком клијенту понаособ слао идентичан садржај, што би дефинитивно имало утицај на перформансе, а не постоји потреба за тим.

Дакле, у случају вишезначног видео преноса довољна је једна нит која преузима енкодовани видео *frame*, те га шаље у *RTP* формату. *RTP* пакет поседује максималну величину од 1460 бајта за *payload*, из тог разлога велики *frame*-ови се шаљу у сегментима у опсегу те величине. Поред енкодованог видео садржаја на почетку пакета умеће се заглавље - *RTP Header*, које има 12 бајта у оквиру којих су садржани подаци неопходни за њихову композицију и коришћење на клијентској страни, као што су временска ознака (енг. *Timestamp*), величина пакета, идентификација последњег сегмента и други.



*Слика 33. UML класни дијаграм RTP сервера*

Као што се види на претходном *UML* дијаграму, срце овог подмодула представља *MediaSession* класа, која има могућност функционисања како вишезначног, тако и једнозначног видео преноса. Међутим, у оба случаја користи се пласирање идентичног садржаја ка свим корисницима, па овај други није од интереса. Алоцирани објекат ове класе даље се везује (енг. *Bind*) за *RTSP* сервер класу, која се заправо користи за слање *frame*-a. Сервер на основу прослеђеног *ID*-a канала, тражи сесију у мапи својих сесија (видео преноса) и прослеђује *frame* ка правој. Уколико је барем један клијент повезан, сервер одобрава слање садржаја. Такође, у случају једнозначног видео преноса обавештава *MediaSession* објекат да креира засебну инстанцу *RTP* сервера за сваког клијента и пласира му исти *frame*. Такође, обавештава га о прекиду везе са *RTSP* клијентом па му омогућава уништавање засебне инстанце или прекид живог видео преноса уколико нема активних корисника.

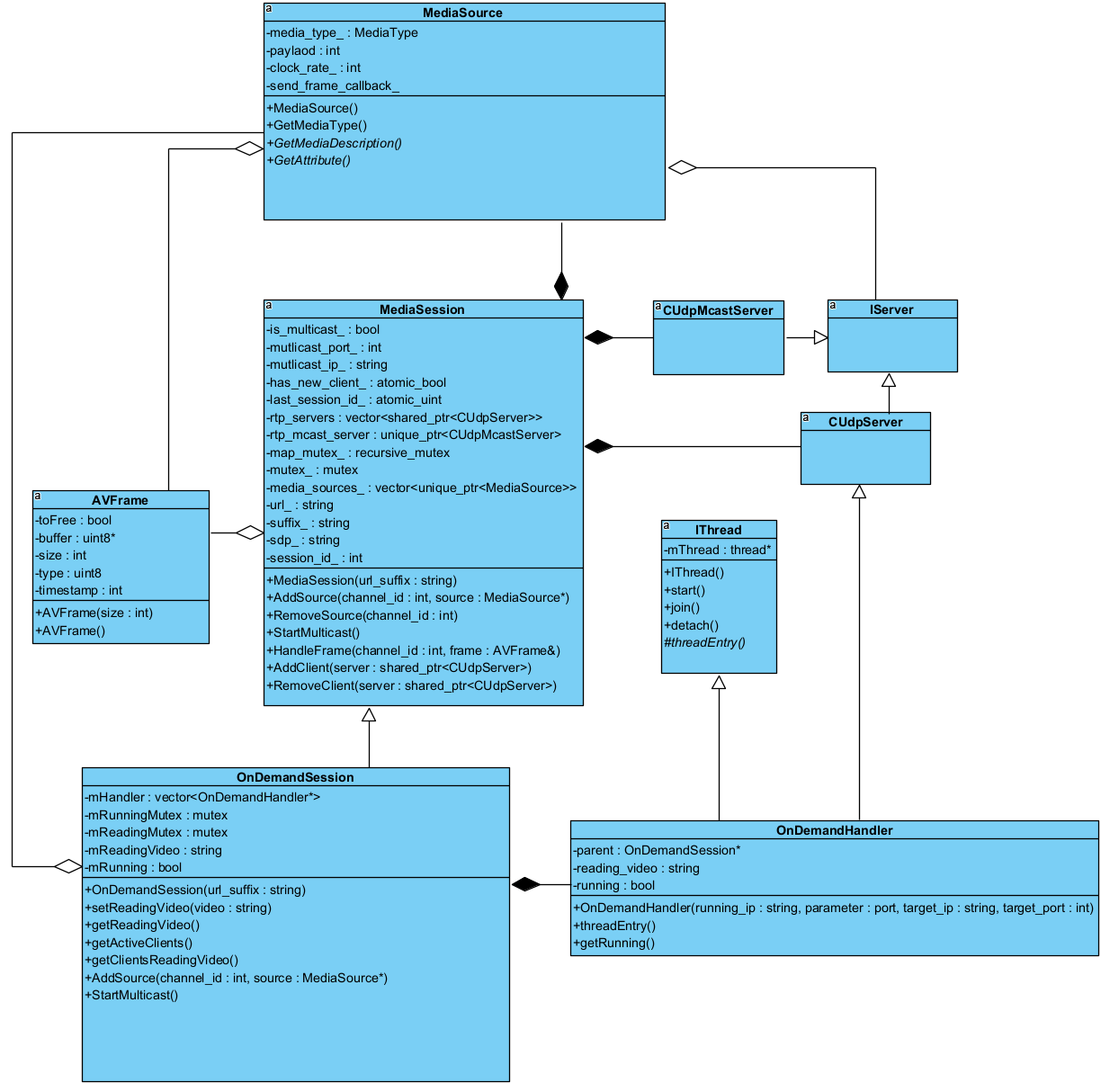


*Слика 34. Алгоритам рада RTP сервера*

#### 5.1.1.4 *RTP* једнозначни “*On Demand*” видео пренос

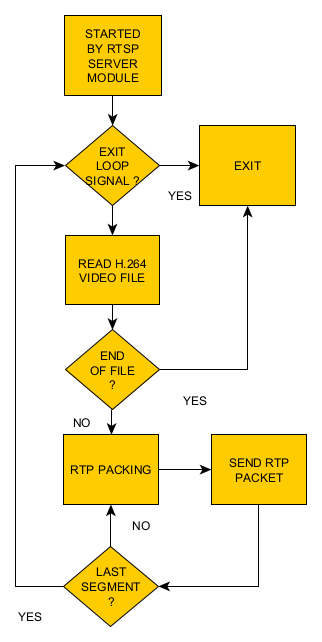
Видео пренос по захтеву је тип преноса који, за разлику од живог преноса, функционише у „1 на 1“ односу сервера и клијента. За сваког пристиглог клијента репродукује се раније снимљени видео садржај од почетка. На пример, један корисник може гледати пети минут видео снимка, док други гледа почетак видеа.

Да би тако нешто било изводљиво, поред чињенице да снимљени видео садржај мора постојати, сервер за видео пренос, након *RTSP* интеракције са клијентом, мора инстанцирати посебан једнозначни *RTP* сервер, као и читач видео записа који ће видео запис пласирати ка клијенту у оквиру засебне сесије (нити). Важно је напоменути да сваки новонастали *RTP* сервер мора поседовати јединствени порт на којем извршава видео пренос по захтеву. Такође, *RTSP* сервер током интеракције са клијентом добија и *IP* адресу и порт *RTP* клијента којем ће *RTP* сервер пласирати садржај. Овај део је изузетно осетљив, јер овакав подухват приликом повезивања више клијената резултује великом потрошњом *CPU*-a и *RAM* меморије уређаја на којем се извршава серверска апликација. С тога је, у оквиру овог пројекта, уведено ограничење активних корисника видео преноса по захтеву и сведено је на максимални број од 5 корисника по каналу (видео преносу). Овај механизам је имплементиран уз помоћ мрежног менаџера, који ће детаљније бити описан у наредном поглављу.



*Слика 35. UML класни дијаграм “On Demand” RTP сервера*

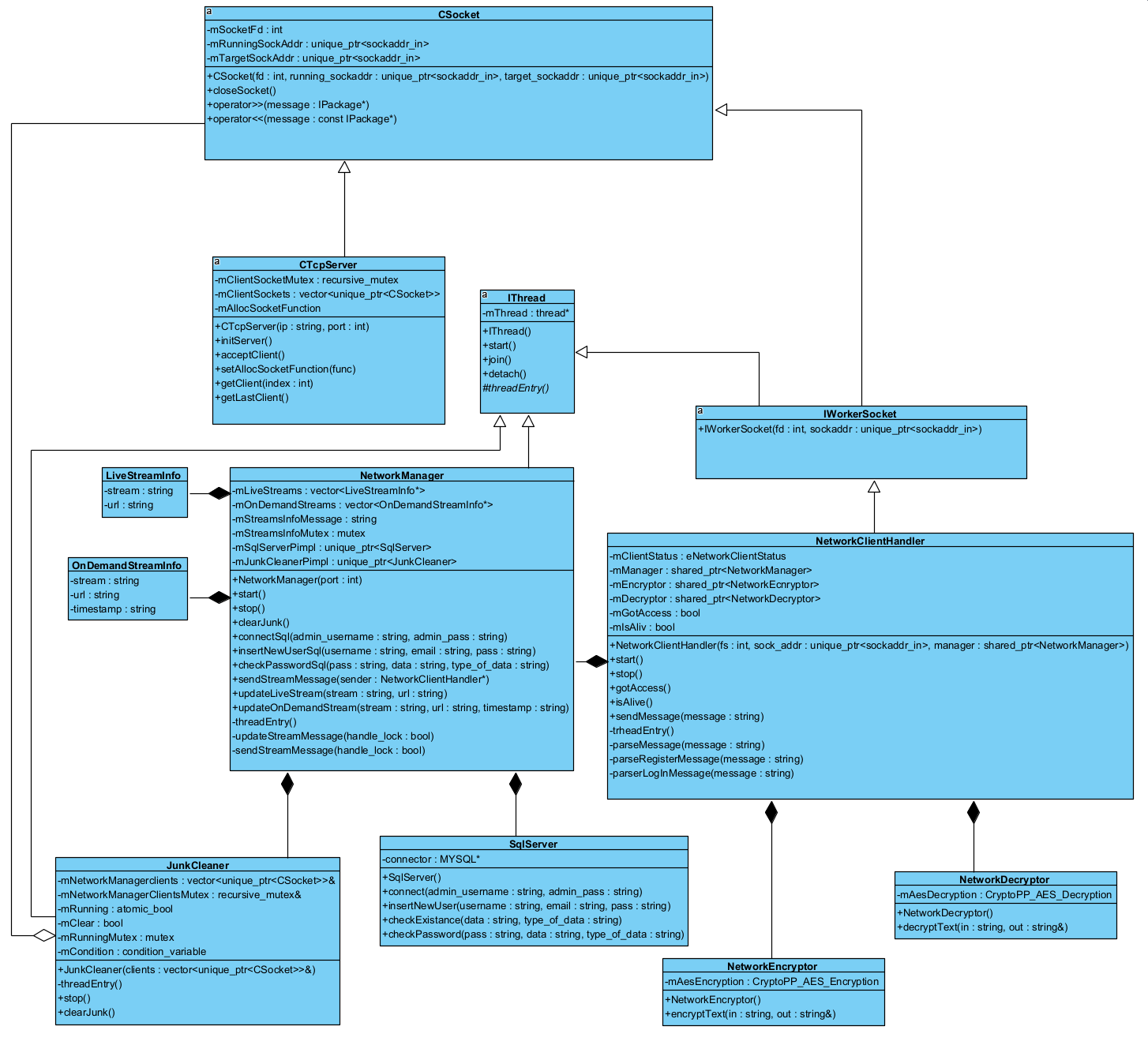
Као што се види на претходном *UML* дијаграму, главну класу овог подмодула представља *OnDemandSession* класа. У питању је изведена класа *MediaSession* класе, која у потпуности наслеђује њене функционалности, али их у неким ставкама проширује, док их у неким ограничава или у потпуности забрањује. Изведена класа нема више могућност вишезначног видео преноса, с обзиром да је утврђено да тако нешто нема смисла за видео пренос по захтеву. Такође, забрањује *RTSP* серверу могућност слања садржаја путем мапирања сесија. *RTSP* сервер обавештава *OnDemandSession* објекат да креира засебну инстанцу *RTP* сервера за сваког клијента, а уз то се инстанцира видео читач (*AvH264Reader*) и нит у којој се извршава пласирање видео садржаја по захтеву.



*Слика 36. Алгоритам рада “On Demand” RTP сервера*

### 5.1.2 Мрежни менаџер

Мрежни менаџер је модул који је, у оквиру овог система, задужен за комуникацију са мрежним корисницима, одржавање и приступање бази података, као и заштиту личних података корисника. На следећој слици приказан је *UML* класни дијаграм мрежног менаџера.



*Слика 37. UML класни дијаграм мрежног менаџера*

#### 5.1.2.1 Интеракција са мрежним корисницима

Мрежни корисници, попут *RTSP* клијената у интеракцији са сервером, морају испоштовати стриктан протокол односно начин на који комуницирају са мрежним менаџером. Протокол комуникације не представља неки од познатих стандарда из области мрежних комуникација, већ специфичан вид паковања, као и слања захтева и одговора који је посебно осмишљен за потребе овог пројекта.

Комуникација је базирана на *TCP* протоколу, где мрежни менаџер има улогу *TCP* сервера. Попут *RTSP* сервера, овај модул такође креира нити за комуникацију са пристиглим клијентима, уколико дође до прекида везе са клијентом обавештава се специјална нит да ослободи динамички алоциране ресурсе и избаци клијента из низа активних клијената.

Комуникација између мрежног менаџера и корисника може се поделити у два режима:

* Иницијални – одвија се приликом корисничког приступања мрежи;
* Устаљени – одвија се од тренутка када је иницијални део комуникације завршен све до изласка мрежног корисника.

Током иницијалног режима комуникације мрежни корисник може послати захтев за регистрацију и/или захтев за пријаву. Уколико је корисник већ регистрован, прескаче се регистрација и шаље се само захтев за пријаву. Мрежни менаџер комуникацију са корисником држи у иницијалном режиму све док захтев за пријаву није успешно обрађен, а потом се прелази у устаљени режим.

Пример валидног захтева за регистрацију корисника:

"CMD:REGISTER\_USER\r\n"  
"USERNAME=user\r\n"  
"EMAIL=mail@generic.com\r\n"  
"PASSWORD=secret98765\r\n\r\n"

Пример одговора мрежног менаџера:

"RESP:ERR\r\n"  
"REASON=E-mail address is already in use\r\n\r\n"

Може се приметити да је у сваком реду додат пар специјалних карактера “\r\n” (*Carriage return* и *Newline*), осим на крају поруке када је тај пар дуплиран. Ово је смерница механизму за парсирање поруке да ли је порука готовa или постоји нови ред. Ова логика је преузета са *RTSP* механизма за размену порука.

У устаљеном режиму комуникације, мрежни менаџер благовремено ажурира корисника о листи постојећих *RTSP* видео преноса. Поред тога, у оквиру устаљеног режима постоји интеракција са мрежним менаџером уколико корисник жели да користи неки од понуђених видео преноса по захтеву. У том случају, корисник прво шаље захтев мрежном менаџеру, који проверава број активних клијената на жељеном видео преносу, те уколико видео пренос није дошао до лимита (споменутог у поглављу 5.1.1.4), корисник даље шаље захтев ка *RTSP* серверу, у супротном добија негативну поруку од менаџера.

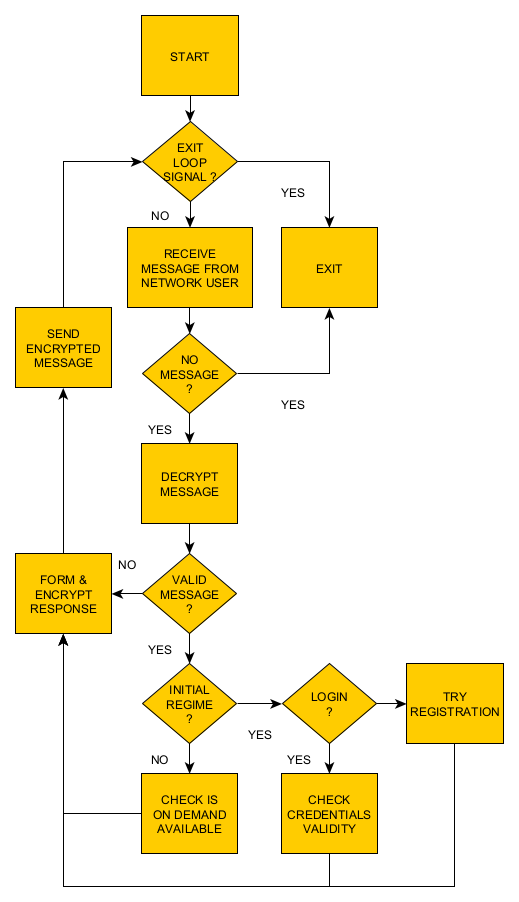
#### 5.1.2.2 Читање и писање у бази података

За сервер базе података користи се *MySQL*, а за приступање бази користи се *MySQL API* написан у програмском језику *C*, те су његове структуре и функције запаковане у оквиру мрежног менаџера по принципу *PIMPL* (*Pointer to Implementation*) идиома. За управљање користи се помоћна *Wrapper* класа, где је целокупна декларација и дефиниција те класе смештена у *source* фајлу *NetworkManager* класе, док је у *header* фајлу декларисана само класа по имену и додато је поље – показивач на декларисану класу.

На почетку рада серверске апликације неопходно је унети администраторске креденцијале да би био омогућен приступ бази података. Након тога програм из перспективе базе података се тумачи као администратор.

Док мрежни корисник подноси захтеве за регистрацију/пријаву, утичница задужена за комуникацију са мрежним корисником шаље захтев ка бази на основу података које је добио од корисника. Захтев који се шаље преко интерфејса је заправо у форми која је идентична команди коју администратор може унети путем терминала (*Command Prompt* или други).

За акцију пријаве корисника потребно је извршити упит да ли се за прослеђено корисничко име или имејл поклапа послата лозинка у табели корисника унутар базе података. За акцију регистрације је потребно уписати нове податке у базу података, али је пре тога потребно проверити да ли већ постоји идентично корисничко име и/или *e-mail* адреса у бази.



*Слика 38. Алгоритам рада NetworkClientHandler* *инстанце*

#### 5.1.2.3 Криптовање порука

Током акција за регистрацију и пријаву корисника кроз мрежу се у форми поруке преносе приватни подаци корисника – корисничко име, *e-mail* адреса и лозинка. У интернет мрежи врло је лако могуће те поруке пресрести и видети њихов садржај, чиме се нарушава приватност. С тога је веома важно обезбедити сигурност корисничких података.

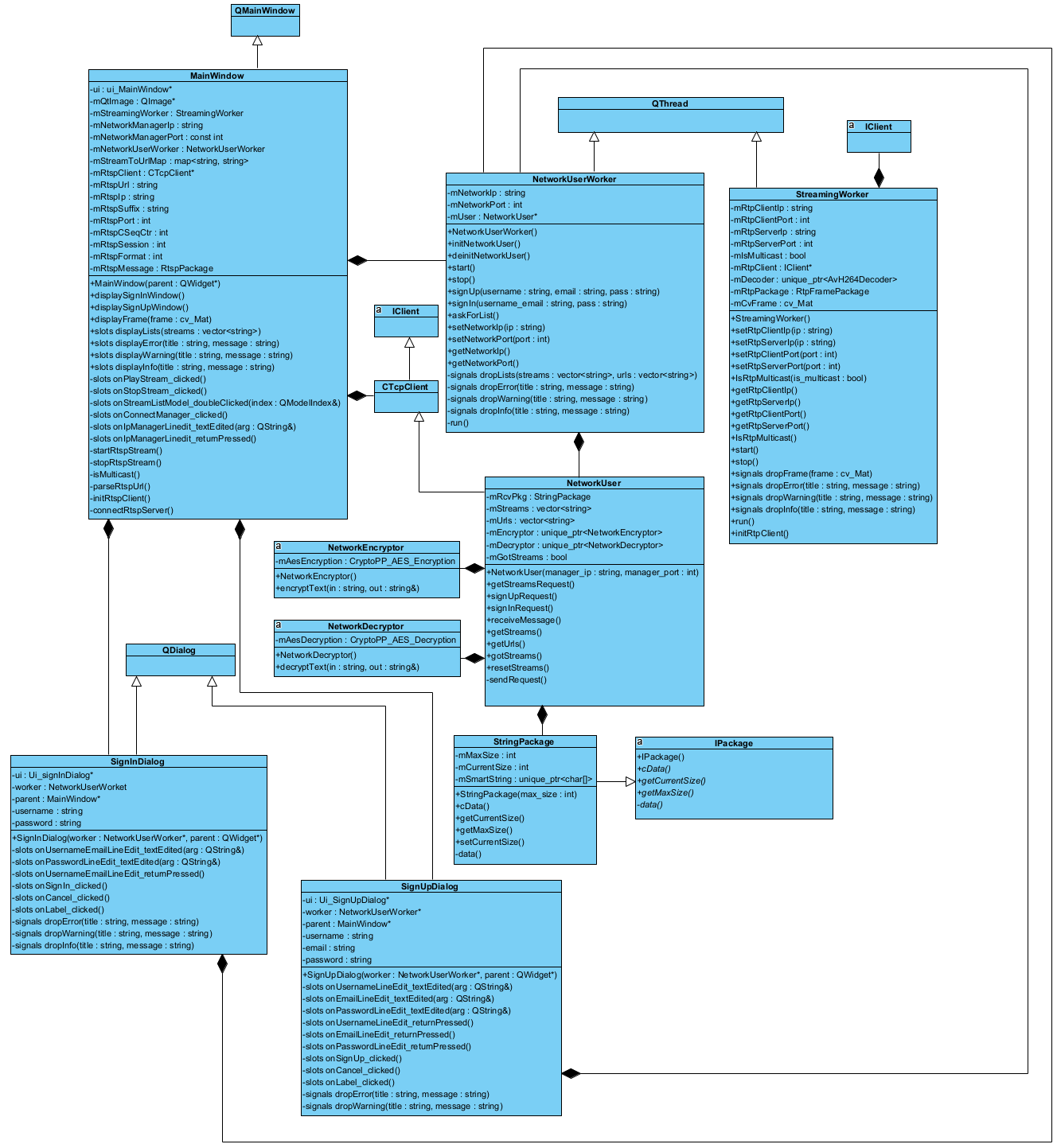
У оквиру овог пројекта то је реализовано помоћу криптографије – енкрипције података. У питању је *AES* симетрична енкрипција, у оквиру које мрежни менаџер и корисник унапред знају јавни кључ и вектор иницијализације (енг. *Initialization Vector*) – *IV*. За реализацију енкрипције података користи се библиотека *Crypto++*. Енкрипција података примењује се само у оквиру комуникације између мрежног менаџера и корисника, и то у оба смера. Принцип рада овог подмодула је веома једноставан, пре слања поруке, корисник или менаџер ураде енкрипцију података, а након пријема исте ураде декрипцију примљеног садржаја.

## 5.2 Програмско решење клијентске апликације

На основу концепта решења, може се закључити да је клијентска апликација у оквиру овог трослојног система подељена у три мања модула:

* *RTSP* клијент;
* *RTP* клијент;
* Мрежни корисник.

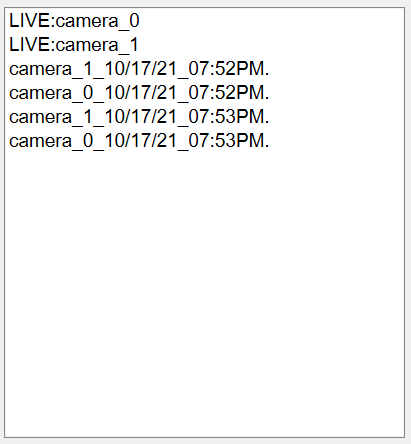
Клијентска апликација је много једноставније реализована у односу на серверску, што је и логично, јер је њен систем знатно мање комплекснији. Ради пријатнијег окружења клијентска апликација је интегрисана у *GUI* апликацију, уместо обичне конзолне. На следећој слици приказан је *UML* класни дијаграм овог модула трослојног система.



*Слика 39. UML класни дијаграм клијентске апликације*

### 5.2.1 *RTSP* клијент

*RTSP* клијент је модул који је задужен за конекцију и комуникацију са *RTSP* сервером, такође, задужен је за прекид видео преноса или за пребацивање на неки други пренос. Након што је корисник успешно евидентиран у мрежи, добија листу активних видео преноса, где приликом одабира неког од њих *RTSP* клијент може да започне свој рад.



*Слика 40. Изглед листе активних видео преноса*

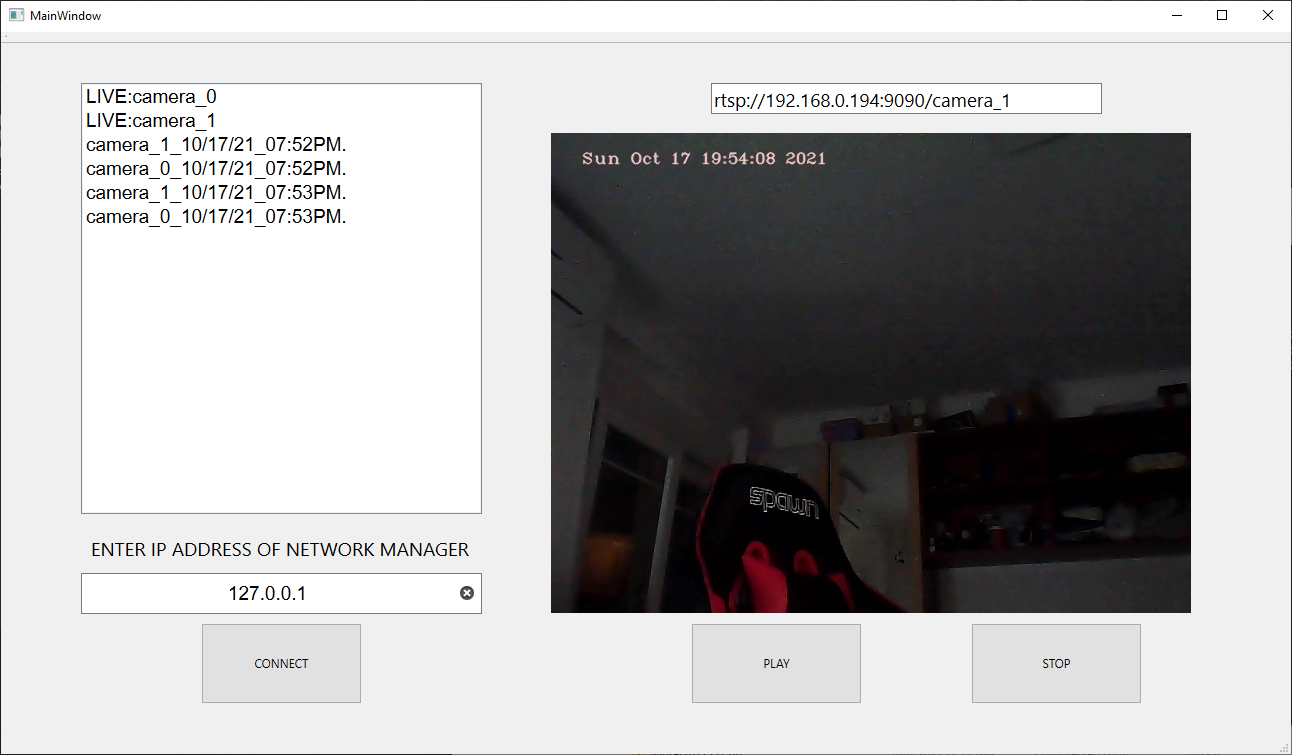
Приликом комуникације са сервером – клијент шаље следеће *RTSP* захтеве: *Options*, *Describe*, *Setup* и на крају *Play*. Уколико је сервер позитивно одговорио на сваки од датих *RTSP* захтева и проследио одговарајуће параметре при одговорима, корисник има све услове да успешно започне преглед видео преноса. У оквиру ових порука налазе се и информације као што су *IP* адреса, порт и протокол *RTP* сервера жељеног видео преноса, али и порт који сервер очекује на страни *RTP* клијента, у случају преноса по захтеву. Након успешно обрађеног *Setup* захтева, могуће је алоцирати *RTP* клијента, а након успешно обрађеног *Play* захтева, могуће је започети његов рад. Уколико корисник жели прекид репродукције видео преноса шаље се *Teardown RTSP* захтев ка серверу и шаље се нова секвенца *RTSP* захтева уколико се одабере други видео пренос.

Целокупна *RTSP* комуникација одиграва се у главној нити клијентске апликације, а због брзине комуникације између сервера и клијента не долази до блокирања корисничког интерфејса.

### 5.2.2 *RTP* клијент

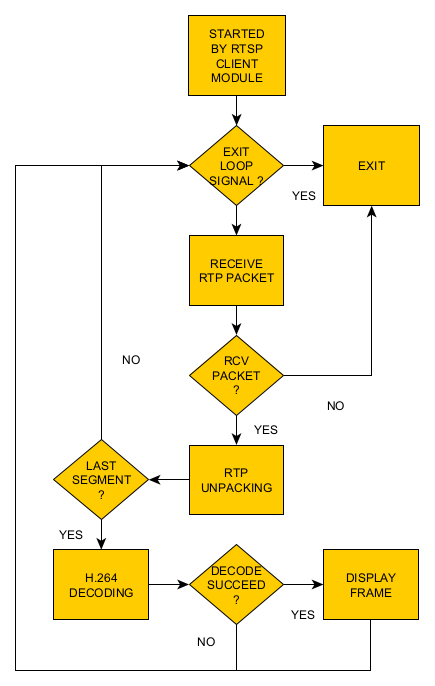
*RTP* клијент је подмодул задужен за пријем и приказивање *frame*-ова видео преноса. Његова улога на први поглед делује једноставно, међутим обухвата комплексне процесе. Рад овог подмодула почиње након што *RTSP* клијент успешно обави свој део посла - алоцира и иницијализује *RTP* клијента и покрене његову нит.

У оквиру овог подмодула за комуникацију се користи поново објекат из *Socket Wrapper* класа – у случају живог преноса то је *CUdpMcastClient*, док је у случају преноса по захтеву реч о *CUdpClient* класи. Рад се одвија тако што клијент прво прима *RTP* пакете и извршава колекцију ако је реч о сегментима, што у већини случајева јесте. Након што је одрађена колекција, односно добијен је *RTP* пакет на ком је евидентирано да је у питању последњи сегмент у његовом заглављу, састављени резултујући *frame* се прослеђује објекту класе *AvH264Decoder* који извршава процес *H.264* декомпресије. Постоји могућност да декодер у некој итерацији не успе да добије слику. То је, пре свега, честа појава код живог видео преноса јер не може успешно да одради предвиђања или неки од других наведених процеса покривених у склопу теоријских основа. За неуспелу декомпресију се примљени материјал обрише и наставља се колекција следећег. Ова појава се дешава при самом почетку репродуковања, где је потребно да се декодер „уигра“, а након првог успешног декодовања видео пренос даље тече у најбољем реду.



*Слика 41. Успешно приказивање живог видео преноса*

Након што је декомпресија успешно изведена резултујући сирови *frame* пакује се у добро познату *cv::Mat* структуру за складиштење и одатле се шаље на приказивање. Пошто приказивање преноса треба да се одигра на постојећем главном прозору то не може да се изведе из нити овог модула, јер се то коси са радом и начелима *Qt* радног окружења (енг. *Framework*). Због тога се користи посебан *Qt* механизам за интеракцију између нити познатији као *Signal-Slot*. На страни класе овог подмодула потребно је декларисати методе са *signal* атрибутом, у оквиру којих се постављају аргументи за типове података које је потребно проследити ка другој нити. Важно је напоменути да ове методе никако није потребно дефинисати, а њихово коришћење одвија се тако што се при њиховом позиву користи резервисана реч *emit.* Са стране класе главног прозора, потребно је декларисати методе са истим аргументима, али са *slot* атрибутом. У њиховом случају потребно је дефинисати шта се ради приликом емитовања података ка овој нити. Такође, неопходно је *signal* методу једног и *slot* методу другог објекта повезати помоћу статичке методе *QObject::connect*. На овај начин емитује се *cv::Mat* ка нити главног прозора и извршава приказивање његових пиксела на резервисаном месту на прозору.

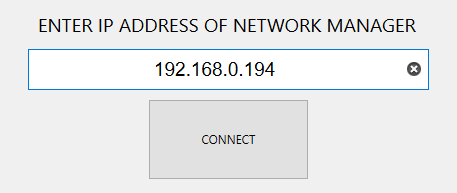


*Слика 42. Алгоритам рада RTP клијента*

### 5.2.3 Мрежни корисник

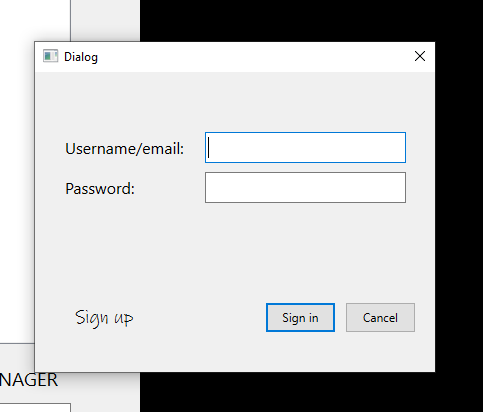
Мрежни корисник је модул који је задужен за комуникацију са мрежним менаџером која обухвата акције регистрације и/или пријаве корисника, добављања активних видео преноса, проверу заузетости захтева по преносу, али и енкрипцију и декрипцију порука које размењује.

При покретању клијентске апликације рад овог модула је први који долази на ред. Да би корисник могао да гледа било који од видео преноса у оквиру овог система неопходно је да се прво региструје, уколико не поседује налог, па да се пријави. Сваки мрежни корисник да би постао евидентиран у мрежи мора да унесе следећа три податка: *e-mail* адресу, корисничко име и лозинку. У главном прозору апликације постоји текстуално поље и дугме за повезивање са мрежним менаџером. Унутар поља потребно је унети *IP* адресу машине на којој се извршава серверска апликација, док је порт унапред дефинисан.



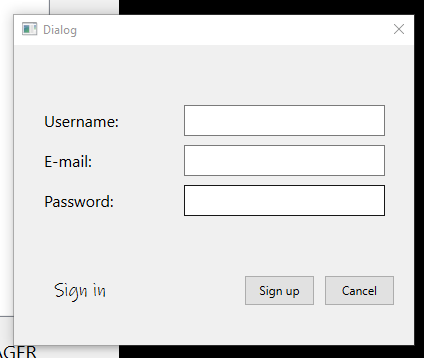
*Слика 43. Тексутално поље и дугме за конекцију са мрежним менаџером*

Притиском на дугме за повезивање покреће се процес *TCP* конекције са мрежним менаџером, уколико је процес успешно извршен отвара се додатни прозор за пријаву корисника.



*Слика 44. Dialog прозор за пријаву корисника*

Уколико корисник није регистрован притиском на текстуалну ознаку *Sign Up* се преусмерава на прозор за регистрацију, док на прозору за регистрацију постоји ознака *Sign In* чијим притиском се корисник преусмерава у прозор за пријављивање.



*Слика 45. Dialog прозор за регистрацију корисника*

Након уноса неопходних креденцијала, мрежни корисник шаље изабрани захтев са креденцијалима ка клијенту. Пре самог слања, извршава се енкрипција података по истом принципу као код мрежног менаџера серверске апликације. Након тога мрежни корисник добија одговор од сервера о успеху захтева, који је такође енкриптован, па га је потребно декриптовати.

Након што је корисник успешно пријављен затвара се прозор за пријављивање и главни прозор је поново могућ за коришћење. На менију главног прозора налази се табела коју мрежни корисник попуњава на основу поруке коју је добио од мрежног менаџера, а табела се благовремено ажурира при свакој промени листе видео преноса. Промене на табели могу бити: додавање новог видео преноса по захтеву, ажурирање временске ознаке неког видео преноса по захтеву или евентуалног прекида неког од видео преноса.

Интеракција са мрежним менаџером одиграва се у главној нити све до краја иницијалног дела комуникације, док се приликом устаљеног режима рада креира блокирајућа нит која очекује поруку од менаџера у виду ажуриране листе видео преноса.

# 6. Верификација и профилисање

У оквиру овог поглавља описан је начин на који је верификован рад трослојног система, као и измерени резултати перформанси и потрошњи модула система.

Целокупни систем је тестиран на персоналном рачунару. Тестиран је ручно, услед недостатка времена за писање аутоматских тестова или *Unit* и интеграционих тестова. Може се закључити да су крајњи тестови добили очекиване резултате и да се систем успешно покреће и ради у дужим временским интервалима.

## 6.1 Лимити и потрошње серверске апликације

Као што се може закључити, серверска апликација покрива велики број функционалности, алоцира велики број објеката и извршава велики број петљи у паралелним нитима. Апликација са таквим тенденцијама и капацитетима нема неограничене ресурсе на располагању па је потребно увидети који су њени реални лимити у складу са машином у оквиру које се извршава.

У оквиру лимитационог тестирања рађене су провере потрошње серверске апликације у рада за једну и две камере у режиму мировања – нема корисника, режиму живог преноса, режиму услуге видео преноса по захтеву – по 3 корисника по сваком видео преносу по захтеву.

|  |  |  |  |
| --- | --- | --- | --- |
| Број повезаних камера | *CPU* потрошња [%] | потрошња *RAM* меморије [MB] | потрошња  диска [MB/s] |
| 1 | 22.1 | 44.3 | 0.3 |
| 2 | 43.7 | 69.5 | 0.9 |

*Табела 1. Потрошња серверске апликације у режиму мировања*

|  |  |  |  |
| --- | --- | --- | --- |
| Број повезаних камера | *CPU* потрошња [%] | потрошња *RAM* меморије [MB] | потрошња  диска [MB/s] |
| 1 | 23.7 | 46 | 0.3 |
| 2 | 50 | 70.7 | 0.9 |

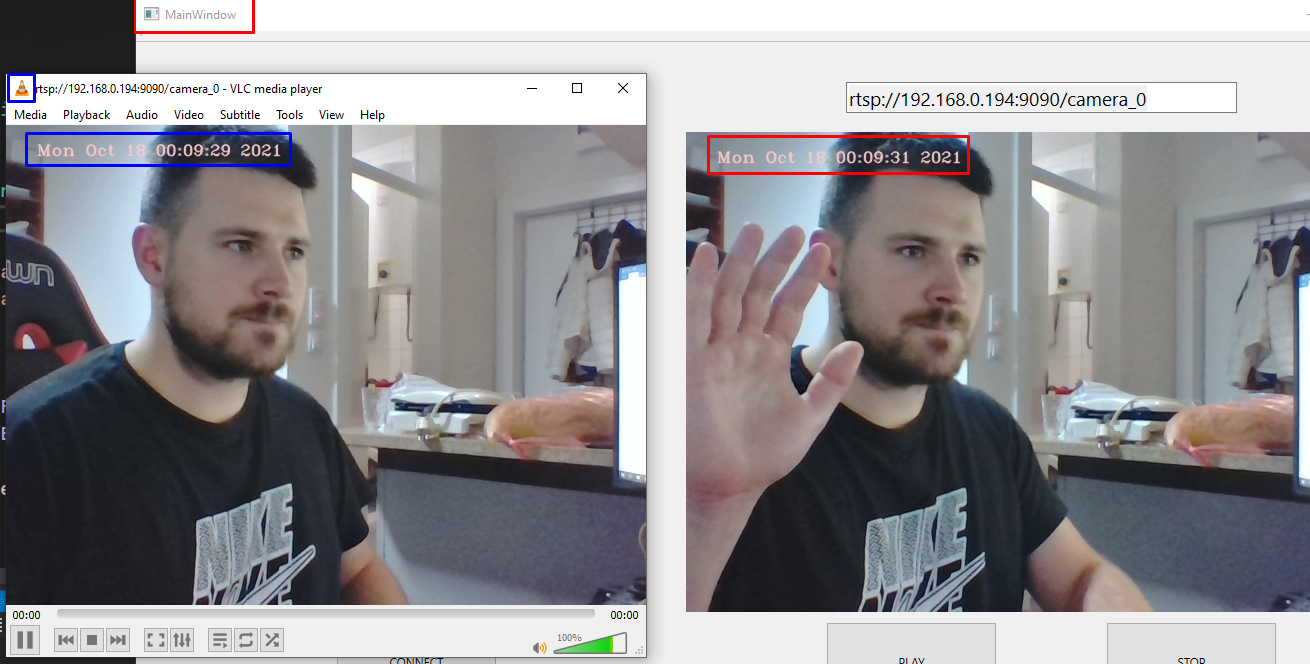
*Табела 2. Потрошња серверске апликације у режиму живог преноса*

|  |  |  |  |
| --- | --- | --- | --- |
| Број повезаних камера | *CPU* потрошња [%] | потрошња *RAM* меморије [MB] | потрошња диска [MB/s] |
| 1 | 51.5 | 107.8 | 3.4 |
| 2 | 72.8 | 160.3 | 7.9 |

*Табела 3. Потрошња серверске апликације у режиму преноса по захтеву*

## 6.2 Перформансе клијентске апликације у односу на *VLC*

Приликом развијања серверске апликације, односно модула – сервер за видео пренос као референца коришћен је програм *VLC Media Player*. Примећено је да за пренос уживо постоји константно кашњење од 2 секунде у односу на оригинални приказ са камере. Овај проблем је отклоњен приликом развијања клијентске апликације, где је кашњење сведено у *ms*.



*Слика 46. Резултати перформанси између два решења RTP клијента*

# 7. Закључак

Коначно тестирање показало је да су сви елементи система успешно интегрисани у једну синхронизовану целину која успешно обавља тражени задатак и обезбеђен је поуздан рад свих модула система.

Једна од предности представљене реализације система јесте и коришћење паметних показивача (енг. *Smart Pointer*) који се сами старају о ослобађању динамички алоциране меморије, што спречава евентуално цурење динамичке меморије (енг. *heap*). Пре свега, смањује се вероватноћа да ће *developer* направити такву грешку.

Предлози за побољшање система:

* Омогућити *Web* подршку, односно *Web* верзију *RTSP* клијента;
* Имплементирати подршку за паузирање видео преноса по захтеву, као и пуштање насумичних делова снимка са приказом протеклог времена снимка;
* Имплементирати сервис за дијагностику, који би проверавао потрошњу коју изазива серверска апликација и благовремено зауставио неки од модула или ограничио његове ресурсе;
* Раздвојити модуле серверске апликације у више апликација, тако да се рад подели у више уређаја у систему:
  1. Aпликација која пласира живи видео пренос са камере и врши записивање видео снимака на дељеном мрежном диску;
  2. Aпликација која је задужена за услуживање видео преноса по захтеву;
  3. Aпликација која је задужена за евиденцију корисника система, управљање базом података и преусмеравање на видео пренос;
* Имплементирати *Watchdog* сервис, који би поновно покретао неактивне делове система;
* Лепши изглед корисничког интерфејса код клијентске апликације.

# 8. Литература

1. <https://sites.google.com/site/izradasajtovatippingpoint/livestreaming---direktni-internet-video-prenosi> , интернет чланак о директном интернет видео преносу, страница *izradasajtova.rs*, приступљено: октобар 2021.
2. <https://www.youtube.com/> , интернет страница *YouTube*, приступљено: октобар 2021.
3. <https://www.aver.com/AVerExpert/what-you-should-know-about-livestreaming-for-business> , интернет чланак о пословном *Livestreaming*-у, страница *AVer*, приступљено: октобар 2021.
4. <http://es.elfak.ni.ac.rs/rmif/Prenos-podatak-februar-2011/Pre.-pod-%202010/Pdf-2010/Pogl-09-Ethernet%20%28210-225%29.pdf> , Етернет, Истраживачки центар за ембедед системе, Електронски факултет, Универзитет у Нишу, приступљено: октобар 2021.
5. <https://sr.wikipedia.org/sr-ec/Етернет> , Етернет, *Wikipedia* чланак, приступљено: октобар 2021.
6. [https://sr.wikipedia.org/sr-ec/ОСИ\_модел](https://sr.wikipedia.org/sr-ec/%D0%9E%D0%A1%D0%98_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB) , ОСИ модел, *Wikipedia* чланак, приступљено: октобар 2021.
7. <http://etsntesla.edu.rs/SAJT%20EC3/racmreze/predavanja/Modeli_umrezavanja.pdf> , Модели умрежавања, ЕТШ „Н. Тесла“, Панчево, приступљено: октобар 2021.
8. <https://www.bpa.edu.rs/FileDownload?filename=5f7860cc-8f2f-47ca-badd-cbf59eb5d92c.pdf&originalName=9.%20RM_Mrezni%20standardi.pdf> , Рачунарске мреже, Д. Љубичић, Београдска академија пословних и уметничких струковних студија, приступљено: октобар 2021.
9. Милојко Јевтовић. *Комуникациони протоколи интернета*. Академска мисао, 2015.
10. <https://sr.wikipedia.org/wiki/Интернет_протокол> , Интернет протокол, *Wikipedia* чланак, приступљено: октобар 2021.
11. <https://www.bpa.edu.rs/FileDownload?filename=2fb93757-004f-4bd5-ae82-92cc16728fd9.pdf&originalName=6.%20RM_IP%20adresiranje.pdf> , Рачунарске мреже, Д. Љубичић, Београдска академија пословних и уметничких струковних студија, приступљено: октобар 2021.
12. <https://sr.wikipedia.org/wiki/UDP_(protokol)> , *UDP* протокол, *Wikipedia* чланак, приступљено: октобар 2021.
13. <https://www.educative.io/edpresso/tcp-vs-udp> , *TCP vs UDP*, страница *educative.io*, приступљено: октобар 2021.
14. <https://retrosound-shop.ru/bs/windows-8/chto-takoe-brodkast-multikast-i-yunikast-unicast-multicast-broadcast-chto.html> , Шта је емитовање, *Multicast* и *Unicast* – типична *IPTV* мрежна схема, страница *retrosound-shop.ru*, приступљено: октобар 2021.
15. <http://www.steves-internet-guide.com/introduction-multicasting/> , разумевање *IP Multicasting*-а, страница *Steve’s Internet Guide*, приступљено: октобар 2021.
16. <https://sr.wikipedia.org/wiki/Трансмисиони_контролни_протокол> , Трансмисиони контролни протокол, *Wikipedia* чланак, приступљено: октобар 2021.
17. Henning Schulzrinne, Anup Rao, and Robert Lanphier. *Real time streaming protocol (RTSP)*. (1998).
18. <https://sr.wikipedia.org/sr-ec/Интернет_телефонија> , Интернет телефонија, *Wikipedia* чланак, приступљено: октобар 2021.
19. <https://pdfcoffee.com/rtp-rtcp-rtsp-3-pdf-free.html> , *RTP-RTCP-RTSP* презентација, Факултет електротехнике у Тузли, С. Дудић, А. Хукић и Ј. Бешић, приступљено: октобар 2021.
20. <https://www.oxfordwebstudio.com/da-li-znate/sta-je-codec.html> , интернет чланак о видео кодеку, страница *Oxford Web Studio*, октобар 2021.
21. <https://www.makeuseof.com/tag/all-you-need-to-know-about-video-codecs-containers-and-compression/> , интернет чланак о видео кодеку и компресији, страница *Make Use Of,* приступљено: октобар 2021.
22. Richardson, Iain E. *The H. 264 advanced video compression standard*. John Wiley & Sons, 2011.
23. <https://www.vcodex.com/an-overview-of-h264-advanced-video-coding/> , интернет чланак о *H.264* видео кодеку, страница *VCodex*, приступљено: октобар 2021.
24. Алемпије Вељовић, и Мирољуб Захорјански. *Увод у базе података*. CET, 2014.
25. <http://msts-travnik.net/wp-content/uploads/2013/10/baze_podataka.pdf> , Базе података, МСТШ Травник, приступљено: октобар 2021.
26. <https://kompjuteras.com/baze-podataka/> , интернет чланак о бази података, страница ИТ блог Компјутераш, приступљено: октобар 2021.
27. <https://www.techsofttutorials.com/the-ansi-sparc-three-level-architecture-with-diagram/> , интернет чланак о *ANSI-SPARC* трослојној *SQL* архитектури, страница *Tech Soft Tutorials*, приступљено: октобар 2021.
28. <https://sr.wikipedia.org/sr/SQL> , *SQL*, *Wikipedia* чланак, приступљено: октобар 2021.
29. <https://sh.wikipedia.org/wiki/Kriptografija> , Криптографија, *Wikipedia* чланак, приступљено: октобар 2021.
30. <https://www.tutorialspoint.com/cryptography/cryptosystems.htm> , интернет чланак о крипто системима, страница *Tutorials point*, приступљено: октобар 2021.
31. <https://www.ssl2buy.com/wiki/symmetric-vs-asymmetric-encryption-what-are-differences> , интернет *Wiki* чланак о разликама између симетричне и асиметричне енкрипције, страница *ssl2buy*, приступљено: октобар 2021.
32. <https://cmake.org/> , званична страница *CMake*, *About* секција, приступљено: октобар 2021.
33. <https://en.wikipedia.org/wiki/OpenCV> , *OpenCV*, *Wikipedia* чланак, приступљено: октобар 2021.
34. <https://en.wikipedia.org/wiki/FFmpeg> , *FFmpeg*, *Wikipedia* чланак, приступљено: октобар 2021.
35. <https://wiki.qt.io/About_Qt> , званична *Wiki* страница *Qt*, *About* секција, приступљено: октобар 2021.
36. <https://www.logosvgpng.com/the-qt-company-logo-vector/> , лого компаније *Qt*, интернет страница *Logo SVG PNG*, приступљено: октобар 2021.
37. <https://en.wikipedia.org/wiki/MySQL> , *MySQL*, *Wikipedia* чланак, приступљено: октобар 2021.
38. <https://1000logos.net/mysql-logo/> , *MySQL* лого, интернет страница *1000logos.net*, приступљено: октобар 2021.
39. <https://en.wikipedia.org/wiki/Crypto%2B%2B> , *Crypto++*, *Wikipedia* чланак, приступљено: октобар 2021.
40. <https://en.wikipedia.org/wiki/File:Crypto%2B%2B-logo.png> , *Crypto++* лого, *Wikipedia* чланак, приступљено: октобар 2021.
41. <https://www.geeksforgeeks.org/socket-programming-cc/> , интернет чланак о *socket* програмирању, страница *Geeks for Geeks*, приступљено: октобар 2021.