

ТЕХНОЛОГИЧНО УЧИЛИЩЕ ЕЛЕКТРОННИ СИСТЕМИ КЪМ
ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ

ДИПЛОМНА РАБОТА

Тема: Мрежов анализатор с възможност за отдалечен анализ
посредством AngularJS 2 клиент

Дипломант: Ивайло Арнаудов

Научен ръководител: Стоил Стоилов

София, 2017

Списък на съкращения

VPN	Virtual Private Network
IP	Internet Protocol

Съдържание

0.1	Компютърни мрежи	5
0.2	Приложения на компютърните мрежи	5
0.2.1	Приложения на компютърните мрежи в бизнеса	5
0.2.2	Приложения на компютърните мрежи в дома	6
0.3	Изисквания към мрежов анализатор	6
1	Методи и технологии за реализиране на мрежови анализатори	7
1.1	Основни принципи, технологии и развойни среди за реализиране на мрежови анализатори	7
1.1.1	Основни принципи	7
1.1.2	Основни технологии	10
1.1.3	Основни развойни среди	11
1.2	Съществуващи решения и реализации	12
1.2.1	Wireshark	12
1.2.2	tcpdump	13
2	Проектиране на структурата на мрежов анализатор	15
2.1	Функционални изисквания към мрежов анализатор	15
2.1.1	Намиране и избиране на физически интерфейси	15
2.1.2	Интерпретиране на данни	15
2.1.3	Филтриране на данни	15
2.1.4	Отдалечен анализ	15
2.1.5	Сигурност на отдалечения анализ	16
2.1.6	Споделяне на анализа	16
2.1.7	Сигурност на споделяне на анализа	16
2.1.8	Графичен интерфейс и визуализация	16
2.2	Съображения за избор на програмни средства и развойна среда	16
2.2.1	C++	16
2.2.2	libpcap	17
2.2.3	Angular 2 и TypeScript	17
2.3	Проектиране на архитектура на мрежов анализатор с отдалечен достъп . . .	17
2.3.1	Цялостна архитектура	17
2.3.2	Архитектура на мрежовия анализатор	17
2.3.3	Архитектура на сървър	19
2.3.4	Архитектура на клиент	20
3	Програмна реализация на мрежов анализатор	21
4	Ръководство на потребителя	22
5	Заклучение	23
A	Изходен код	24

Библиография	25
Списък на фигурите	26
Списък на таблиците	27

Увод

0.1 Компютърни мрежи

Всеки от последните три века бива доминиран от някаква нова технология. Пример за това е ерата на механичните системи съпътстващи Индустриалната революция през XVIII век. За XIX век пък е характерен парния двигател. През XX век, ключовата технология е събирането, обработката и дистрибуцията на информация. С развитието ѝ човечеството става свидетел на инсталацията на глобални телефонни мрежи, изобретяването на радиото и телевизията, експоненциалния растеж на развитието на компютърната индустрия и, разбира се, Интернет. Като резултат от огромния технологичен прогрес в сферата на информационните технологии, през XXIV. разликите между съхраняване, транспортиране и обработка на информация изтъняват, а успоредно с това растат и изискванията на крайния потребител към комуникационните услуги.

Въпреки крехката възраст на компютърната индустрия (напр. в сравнение с автомобилната), тя прави значителен прогрес. През първите две десетилетия от съществуването им, компютърните системи са били силно централизирани. Университет или средно голяма фирма биха имали един или два компютъра, а по-големите институции - по няколко. Идеята за съществуването на малки устройства тип смартфон, които са взаимосвързани, е била по-скоро утопична.

Обединяването на компютрите и комуникациите оказва голямо влияние върху организацията на самите компютърни системи. Старият модел при който един компютър изпълнява заявките на цялата организация бива заменен от нов модел при който голямо количество отделни, но взаимосвързани компютри извършват обработка на дадена информация. Тези системи се наричат **компютърни мрежи**. [Tanenbaum and Wetherall \(2011\)](#) Неформална дефиниция за компютърна мрежа е множество от автономни компютри, взаимосвързани (можещи да обменят информация помежду си) от една технология.

0.2 Приложения на компютърните мрежи

0.2.1 Приложения на компютърните мрежи в бизнеса

Обикновено повечето компании имат голямо количество компютри, най-често по един за всеки служител. Изначално, те биха могли да работят в изолация един от друг, но в даден момент идва необходимост те да бъдат свързани с цел служителите да извършват работата си по-пълноценно чрез колаборация помежду си.

Един от основните проблеми, който решават компютърните мрежи е **споделянето на ресурси**. Целта е информацията да бъде достъпна от всеки в мрежата независимо от физическото му местоположение.

По-важен проблем, който решават компютърните мрежи е **споделянето на информация**. Компаниите са фундаментално зависими от дигиталната информация. Повечето компании имат записи за клиенти, за продукти и т.н. онлайн.

В допълнение, компютърните мрежи дават възможността да се използват вече изградената мрежова инфраструктура за телефонни разговори благодарение на технологията **Internet Protocol (IP) телефония**, или още известна като **Voice over IP (VoIP)**, предоставят механизми за по-богати форми на виртуална комуникация – споделяне на екрана (**Desktop sharing**), видеоконференции, споделена обработка на документи. Компютърните мрежи отварят вратите и за нов бизнес модел, наречен електронна търговия (или **e-commerce**), който се развива с големи темпове и става де факто стандарт при търговията от всякакъв тип. [Tanenbaum and Wetherall \(2011\)](#)

0.2.2 Приложения на компютърните мрежи в дома

В началото на компютърната индустрия причините за покупка на компютър от крайния потребител са се свеждали до нужда от обработка на текст и игри. През XXI век, основната причина е нуждата за достъп до Интернет. Аналогично на компаниите, крайните потребители могат да достъпят отдалечена информация, да комуникират посредством **социалните мрежи**, да купуват продукти и услуги чрез e-commerce системи, да използват електронно банкиране, да споделят мултимедия и софтуер, да колаборират посредством **wiki** сайтове (напр. Wikipedia).

Друго напоследък развиващо се приложение на мрежите е концепцията за **Internet of Things (IoT)**. Основната ѝ характеристика е че електронните устройства на крайните потребители се включват в компютърните мрежи; напр. душа в банята, който традиционно не е компютър, би могъл да записва какво количество вода е използвано и да праща информацията на приложение, което изчислява как водата да бъде използвана възможно най-ефикасно. [Tanenbaum and Wetherall \(2011\)](#)

0.3 Изисквания към мрежов анализатор

Споменатият етап на развитие на компютърните мрежи предразполага към по-комплексни инструменти и процеси за мониторинг и отстраняване на проблеми в мрежата. *Мрежовия анализатор*, *пакетен анализатор* или **packet sniffer** е инструмент, който помага на мрежовия администратор, предоставяйки услугата *анализ на пакети* (**packet analysis**), още известна като **packet sniffing** или **protocol analysis**. Анализът на пакети описва процеса на заснемане и интерпретиране на данни в реално време (т.е. в момента на преминаване през преносвателната среда). Изискванията към един такъв анализатор е да улесни мрежовия администратор със изпълнението на следните задачи:

- Идентифициране и задълбочено разбиране на процесите в мрежата
- Идентифициране на участниците в мрежата, потенциални причинители на атаки или злонамерена активност
- Изследване кой или какво използва наличния капацитет на преносвателната среда, както и на моментите на максимално използване (load) на мрежата

Представеното в текущата дипломна работа приложение има за цел да спази тези изисквания като, взимайки предвид вече описаното развитие на компютърните мрежи с оглед необходимостта от колаборация и децентрализираност, предостави на мрежовия администратор и възможност за отдалечено достъпване на мрежовия анализатор както и споделен преглед на анализа с други администратори.

Глава 1

Методи и технологии за реализиране на мрежови анализатори

1.1 Основни принципи, технологии и развойни среди за реализиране на мрежови анализатори

1.1.1 Основни принципи

Процеса на анализ на пакети включва кооперация между софтуера и хардуера и може да бъде разделен в следните три стъпки:

- **Събиране** В началната фаза на работата си, анализатора събира 'сурови', неинтерпретирани данни в двоичен вид директно от проводника. Типично, това става като съответно избрания мрежови интерфейс за анализ бива превключен в т.нар. **promiscuous mode**. В този режим, мрежовата карта може да 'слуша' за всевъзможен тип трафик по дадения мрежови сегмент, а не просто такъв, адресиран до станцията.
- **Конвертиране** В следващата фаза на работата си, анализатора конвертира събраните данни в разбираем формат за крайния потребител. Тук е мястото, където повечето добри анализатори спират с анализа. След тази стъпка, данните събрани от преносвателната среда са във вид, който може да бъде интерпретиран на много основно ниво; останалата по-голяма част от анализа се оставя на крайния потребител.
- **Анализ** В третата и финална фаза, мрежовият анализатор извършва реалния анализ на събраната и конвертирана информация. Анализатора взема събраните данни, отчита използвания мрежови протокол базирайки се на извлечените до момента данни и започва да анализира конкретните свойства на протокола.

С цел да бъде разбран процеса на работа, а и съответно модела на реализация на мрежов анализатор, е необходимо дефиниране на основните принципи на комуникация между компютърните системи.

Протоколни архитектури

За да се намали комплексността на решенията, повечето мрежи са организирани като стек от **слоеве** или **нива**, всеки изграден върху слоя под него. Броя на слоевете, имената на всеки от тях, съдържанието на всеки и функциите, които изпълнява са различни за различните мрежи. Целта на всеки слой е да предостави конкретни услуги на

разположените по-високо от него в йерархията слоеве като им спестява детайлите около имплементацията на тези услуги.

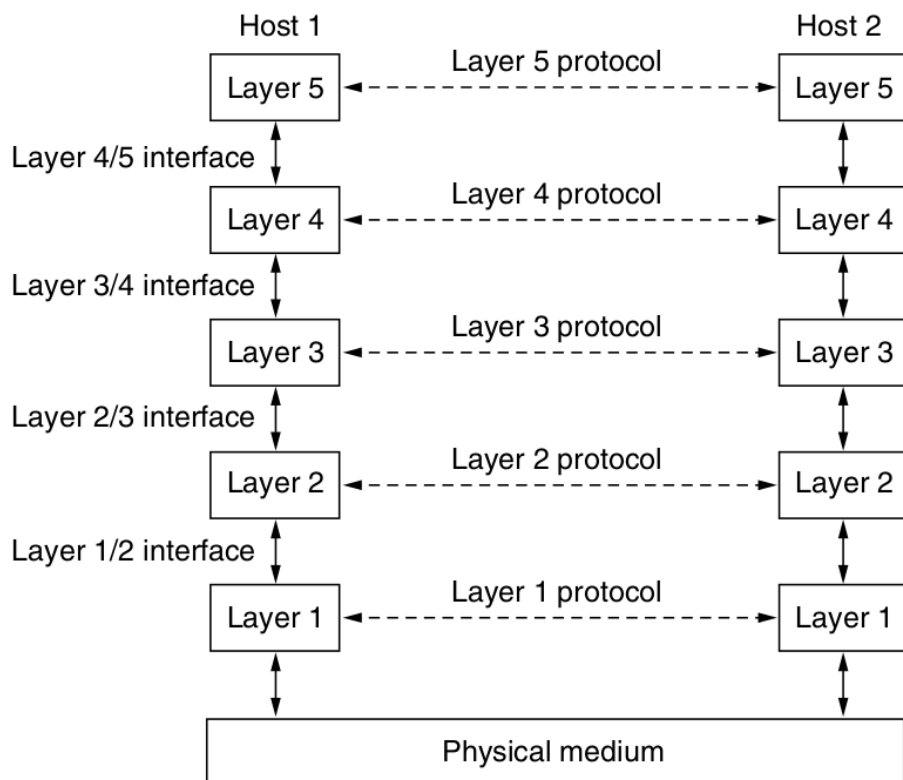
Тази концепция е широко популярна в компютърните науки, още известна е като криене на имплементационни детайли, абстрактни типове от данни, енкапсулация на данни и обектно-ориентирано програмиране. Фундаменталната идея е че дадена част от софтуера (или хардуера) осигурява услуга на потребителите си, но скрива от тях детайлите на вътрешното си състояние и алгоритмите.

Когато слой n на една машина е във връзка със слой n на друга машина, правилата и конвенциите използвани в тази връзка се наричат **протокол на n -ти слой**. На практика, протокол е договореност между комуникиращите страни относно това как протичат процесите на комуникацията между тях. [Tanenbaum and Wetherall \(2011\)](#) Протоколите могат да бъдат прости или комплексни. Някои от общите свойства, които традиционно споделят, макар и абстрактно представени тук, са:

- **Инициация на връзка** Дефинира кой инициира връзката, напр. клиентът или сървърът. При инициране на връзка може да е необходима и допълнителна служебна информация преди да протече обмен на полезна информация.
- **Договаряне на параметрите на връзката** Дефинира процес, в който двете страни се разбират — дали връзката е криптирана, как се пренасят ключовете за декриптиране, какъв тип е връзката (full/half duplex) и др.
- **Форматиране на данните** Дефинира подредбата на данните, в каква последователност се обработват от приемащата страна и др.
- **Откриване на грешки и корекция на грешки** Дефинира какво се случва при загуба на данни, как едната страна на връзката реагира при загуба на отговор от другата и др.
- **Терминиране на връзка** Дефинира как дадено крайно устройство сигнализира на друго че връзката е приключила, каква финална информация трябва да бъде предадена преди успешния край на връзката и др.

Традиционно, тези протоколи не 'живеят' сами, а са основополагащи за цялостния процес на комуникация. Например, на Figure 1.1 е представен пет слоен модел на комуникация между два софтуерни процеса. Реално данни не се предават директно от слой n на едната машина до слой n на другата: комуникацията е **виртуална** (означена с прекъснати линии на Figure 1.1). Вместо това, всеки слой предава данни и контролна информация на този под него докато не се достигне най-ниския слой. Под първия слой е физическият, т.е. преносвателната среда през която реалната комуникация се случва. (означено с непрекъснати линии на Figure 1.1)

Между всяка съседна двойка слоеве има **интерфейс**. Този интерфейс дефинира какви операции и услуги долният слой предлага на горния. Интерфейсите между слоевете трябва да бъдат ясно дефинирани. Това впоследствие би улеснило замяната на един слой с напълно различен протокол или имплементация (напр. смяна на телефонни линии със сателитни такива), защото единственото, което се очаква от новия протокол, е да предлага *точно* същото множество услуги на горния слой като стария. Аналогично, този механизъм позволява един протокол да се промени в даден слой без знанието на слоевете под и над него.



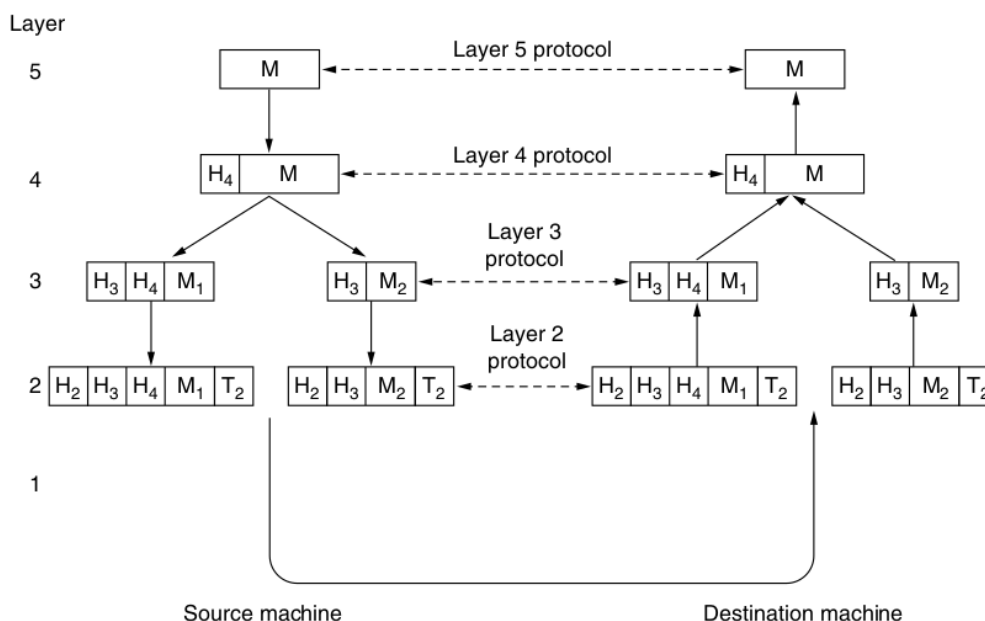
Фигура 1.1: Модел на пет слойна мрежа

Основен механизъм на междуслойна комуникация. Капсулиране и декапсулиране.

Основния механизъм на междуслойна комуникация може да бъде описан с помощта на пет слойния модел представен във Figure 1.1. Типично, даден процес (т.е приложение) иска да изпрати съобщението M . От петия слой, съобщението бива предадено на четвъртия слой за трансмисия. Четвъртият слой слага т.нар. **header** в началото на съобщението и предава резултата към трети слой. Този header включва служебна информация, напр. адреси, за да може съответния четвърти слой на приемната машина да достави съобщението. Други примери за служебна информация могат да бъдат числови поредици (**sequence numbers**), често използвани когато слоят на по-ниско ниво няма функционалност за запазване на последователността на съобщенията, размери и времена.

В много мрежи, често няма граница относно размера на съобщения на четвърти слой, но почти винаги има такава относно размера от самият протокол на трети слой. Следователно, третият слой трябва да раздели идващите съобщения на по-малки единици — пакети, като успоредно с това добавя header към всеки пакет. В случая на Figure 1.2, съобщението M се разделя на две части: M_1 и M_2 .

Третият слой аналогично предава пакетите на втория слой, който от своя страна освен че добавя header, добавя и **опашка (trailer)**. Резултата се предава на първия слой, който се занимава с физическия пренос на данните. Този процес е още известен като **капсулация (encapsulation)**. В приемащата страна, съобщението се декапсулира (**decapsulation**) като всеки header се отделя успоредно с "изкачването" на съобщението нагоре по слоевете. Нито един header за слоевете под n -тия не достига до n -ти слой. Едновременно с това, на Figure 1.2 ясно проличава виртуалната и реална комуникация, както и разликите между протоколи и интерфейси. Например, на четвърти слой процесите концептуално интерпретират комуникацията си като хоризонтална използвайки протокола на четвърти слой и биха имали функции от типа на `send()` и `receive()`, въпреки че в реалност те комуникират със по-ниските слоеве през 3/4 интерфейса, а не директно с другата страна.



Фигура 1.2: Междуслойна комуникация. Капсулиране и декапсулиране.

Имплементации на протоколни архитектури

OSI

TCP/IP

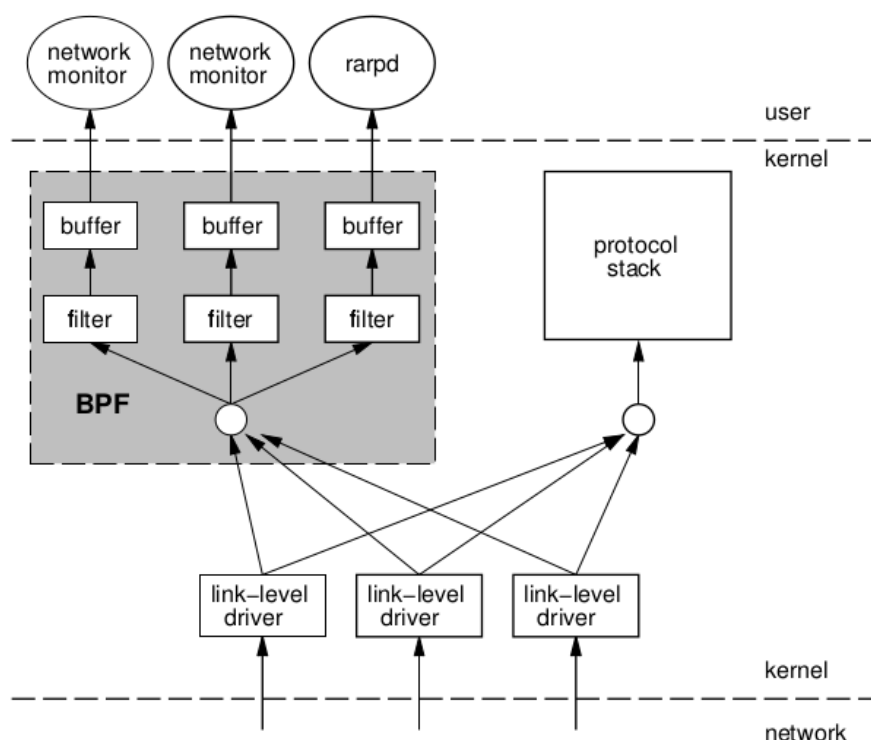
Основен принцип за прихваждане на данни по преносвателната среда
Обикновено когато мрежовата карта получи кадър (**frame**), проверява дали адресът на получателя съвпада с нейния собствен. Ако съвпада, тя генерира прекъсване към процесора. Функцията, обработваща прекъсването е драйверът за мрежовата карта в ядрото на операционната система. Драйверът "закача" времева щампа (**timestamp**) върху приетите данни и копира данните от буфера на картата в заделен блок от памет в ядрото на операционната система. След това системния протоколен стек обработва данните посредством декапсулация ги предава към потребителското приложение.

Когато използваме мрежов анализатор, пакетите следват аналогичен път, но с малка промяна: драйверът копира приети или изпратени данни в част от ядрото на операционната система наречено пакетен филтър (**packet filter**) — пакетните филтри са в основата на функционалността на всеки анализатор. В следствие от местоположението си, те изискват административни привилегии, тъй като копирането на получени/изпратени пакети предполага риск за сигурността. По подразбиране, пакетните филтри пропускат всеки пакет, но поддържат комплексни филтри. На Figure 1.3 е илюстриран описания процес, имплементиран с Berkley Packet Filter (BPF).

1.1.2 Основни технологии

libpcap

libpcap е библиотека с отворен код, която осигурява интерфейс на високо ниво за системи за прихваждане на пакети от преносвателната среда. Създадена е през 1994г. от МакКейн, Лиърс и Якобсън — изследователи в Lawrence Berkeley National Laboratory към University of California като част от научен проект за изследване и подобрене на TCP. Основните цели, поставени пред библиотеката от авторите ѝ, са да се създаде



Фигура 1.3: Преглед на имплементацията на мрежов анализатор на ниво ядро на операционната система

библиотека която е с платформенно-независим API за да се елиминира нуждата от системно-зависими модули в приложенията на по-високо ниво, тъй като всяка операционна система имплементира свои собствени такива механизми.

Интерфейсът на `libpcap` е основно достъпен на програмния език C и C++. Съществуват и голямо количество библиотеки енкапсулиращи функционалността му на езици като Perl, Python, Java, C или Ruby. Откъм операционни системи, `libpcap` се поддържа на повечето UNIX-базирани операционни систем — Linux, Solaris, BSD и др. Съществува и Microsoft Windows версия, която се нарича **winpcap**.

WebSockets

AngularJS

1.1.3 Основни развойни среди

Eclipse

The CDT Project provides a fully functional C and C++ Integrated Development Environment based on the Eclipse platform. Features include: support for project creation and managed build for various toolchains, standard make build, source navigation, various source knowledge tools, such as type hierarchy, call graph, include browser, macro definition browser, code editor with syntax highlighting, folding and hyperlink navigation, source code refactoring and code generation, visual debugging tools, including memory, registers, and disassembly viewers.

1.2 Съществуващи решения и реализации

Преди да разгледаме съществуващите решения и реализации е важно да споменем критериите, необходими за оценяване на полезността на един мрежов анализатор:

- **Поддържани протоколи** Всички мрежови анализатори могат да интерпретират множество от протоколи. Повечето могат да интерпретират най-основните протоколи от мрежовия слой – напр. IPv4 и ICMP; от транспортния слой – TCP и UDP, както и от приложения – DNS и HTTP. Не всички обаче поддържат нетрадиционни или нови протоколи (напр. IPv6).
- **Потребителски интерфейс** От значение е цялостния изглед на приложението, колко лесно се инсталира, колко лесно се извършват необходимите операции посредством интерфейса. От значение е и опита на анализиращия – типично, по-опитният анализиращ би предпочел анализатор използващ командния ред, начинаещият – анализатор с графичен интерфейс.
- **Цена** Голямо количество от мрежовите анализатори са свободно-използваеми и конкуриращи се с платени такива. Обикновено разликата между платените и свободно-използваемите е при прегледа на анализа, който е по-пълноценен при платените.
- **Програмна поддръжка** Обикновено при изникването на проблем анализиращият трябва да има солидна база от източници на решения – документация на анализатора, публични форуми, блогове и т.н. Фундаментално при избора на анализатор е до колко са налични тези източници.
- **Поддръжка на операционната система** Не всеки анализатор поддържа всяка операционна система. Основополагащо за избора на анализатор е операционната система под която се очаква той да функционира.

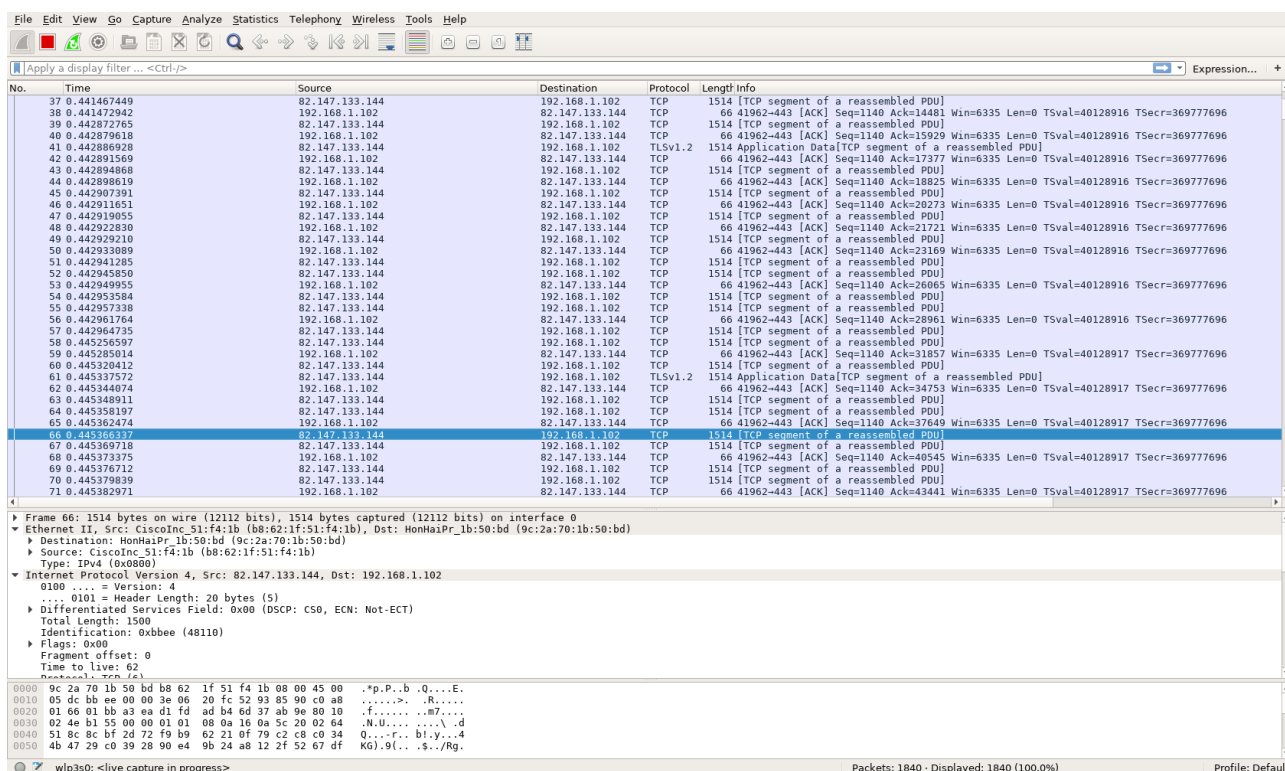
1.2.1 Wireshark

Wireshark има дълга история. Програмата е оригинално създадена от Джералд Комбс, студент по компютърни науки. Първата версия на приложението на Комбс се нарича Ethereal и за първи път е пусната през 1998г. под GNU Public License (GPL). Осем години след пускането на Ethereal, той напуска работа, но за съжаление неговият работодател има пълни права върху името Ethereal. Така през средата на 2006г. се ражда Wireshark.

Wireshark предлага няколко предимства които я правят подходяща за всекидневна употреба. Програмата таргетира както по-напредналите с мрежовия анализ, така и начинаещи.

- **Поддържани протоколи** Поддържа над 850 различни протокола — от по-популярните като IP и DHCP, до по-комплексни частни протоколи като AppleTalk и BitTorrent. Вземайки предвид факта че Wireshark се разработва на принципа на отворения код, нов протокол се добавя с всяка нова версия. В случай че необходим на анализиращия протокол не е имплементиран, той може да бъде имплементиран като разширение и изпратен до разработчиците на Wireshark.

- **Потребителски интерфейс** Един от най-лесните за употреба потребителски интерфейси. Типичен Graphical User Interface (GUI) с ясно описани контекстни менюта и интуитивно оформление. Поддържа цветово кодиране спрямо протокола както и детайлен преглед на неинтерпретираните данни. За разлика от tcpdump, Wireshark GUI е идеална за начинаещия в мрежовия анализ. Разделен е на три компонента: Packet List (списък от пакети), Packet Details (детайли за избрания пакет), Packet Bytes (преглед на пакета в неинтерпретиран вид). Цялостен изглед на интерфейса може да бъде видян на Figure 1.4.
- **Цена** Тъй като програмата се разработва на принципа на отворения код, тя е изцяло безплатна и може да се използва според GPL лиценза.
- **Програмна поддръжка** Тъй като програмата се разработва на принципа на отворения код, тя няма формална поддръжка — тя се базира на обществото от потребители на програмата. Уеб страницата на Wireshark има връзки към няколко форми за поддръжка: онлайн документация, wiki за разработчици, FAQ, както и връзки към официалната мейл листа.
- **Поддръжка на операционната система** Поддържа всички модерни операционни системи, вкл. Microsoft Windows, Mac OS X и Linux базирани операционни системи.



Фигура 1.4: Изглед на потребителския интерфейс на Wireshark.


1.2.2 tcpdump

Програмата е написана през 1987г. от Ван Якобсен, Крейг Леърс и Стивън МакКейн, които работят като изследователи в Lawrence Berkeley Laboratory. Макар че съществуват модерни анализатори с графичен интерфейс, **tcpdump** е изчистен, универсален и ефективен инструмент работещ в командния ред. Едно от основните предимства на програмата е удобството на ползване — използва се една единствена команда, за да се

пусне; работи през SSH сесия, няма нужда от window manager и е широкодостъпна на различни платформи. Понеже използва класически конвенции свързвания с командния ред (напр. писане в stdout) може да се използва в голямо количество ситуации.

- **Поддържани протоколи** Поддържа значително по-малко количество протоколи от Wireshark — Ethernet (FDDI/Token Ring), IP, IPv6, ARP, RARP, TCP, UDP.
- **Потребителски интерфейс** Минималистичен command-line interface (CLI). Типичната структура на една команда е представена на Figure 1.5. Поддържа различни нива на детайлност на анализа. На Figure 1.6 е представен средно-детайлен преглед чрез -vv опцията.

`tcpdump -i eth1 -vvn icmp or udp`



Фигура 1.5: Структура на tcpdump команда.

- **Цена** Тъй като програмата се разработва на принципа на отворения код, тя е изцяло безплатна и може да се използва според BSD лиценза.
- **Програмна поддръжка** Тъй като програмата се разработва на принципа на отворения код, тя няма формална поддръжка — тя се базира на обществото от потребители на програмата.
- **Поддръжка на операционната система** Поддържа повечето UNIX-базирани операционни системи: Linux, Solaris, BSD, HP-UX и др. Microsoft Windows се поддържа през **WinDump**, която използва **WinPcap** — Microsoft Windows имплементацията на **libpcap**.

```
192.30.253.124.https > 192.168.1.102.50882: Flags [.], cksum 0x9687 (correct), seq 1, ack 1, win 31, options [nop,nop,TS val 1639104246 ecr 45537667], length 0
14:50:00.180575 IP (tos 0x0, ttl 1, id 0, offset 0, flags [DF], proto UDP (17), length 383)
  192.168.1.1.ssd > 239.255.255.250.ssd: [udp sum ok] UDP, length 355
14:50:00.181683 IP (tos 0x0, ttl 64, id 62855, offset 0, flags [DF], proto UDP (17), length 74)
  192.168.1.102.44862 > home-77-70-13-1.megalan.bg.domain: [udp sum ok] 59512+ PTR? 250.255.255.239.in-addr.arpa. (46)
14:50:00.183767 IP (tos 0x0, ttl 1, id 0, offset 0, flags [DF], proto UDP (17), length 320)
  192.168.1.1.ssd > 239.255.255.250.ssd: [udp sum ok] UDP, length 292
14:50:00.186880 IP (tos 0x0, ttl 1, id 0, offset 0, flags [DF], proto UDP (17), length 311)
  192.168.1.1.ssd > 239.255.255.250.ssd: [udp sum ok] UDP, length 283
14:50:00.190519 IP (tos 0x0, ttl 1, id 0, offset 0, flags [DF], proto UDP (17), length 375)
  192.168.1.1.ssd > 239.255.255.250.ssd: [udp sum ok] UDP, length 347
14:50:00.194025 IP (tos 0x0, ttl 1, id 0, offset 0, flags [DF], proto UDP (17), length 359)
  192.168.1.1.ssd > 239.255.255.250.ssd: [udp sum ok] UDP, length 331
14:50:00.197282 IP (tos 0x0, ttl 1, id 0, offset 0, flags [DF], proto UDP (17), length 320)
  192.168.1.1.ssd > 239.255.255.250.ssd: [udp sum ok] UDP, length 292
14:50:01.275923 IP (tos 0x0, ttl 64, id 14711, offset 0, flags [DF], proto TCP (6), length 52)
  192.168.1.102.36614 > sof02s18-in-f33.1e100.net.https: Flags [.], cksum 0x8958 (correct), seq 3122062306, ack 2706027691, win 1444, options [nop,nop,TS val 45552128 ecr 2919903715], length 0
14:50:01.275981 IP (tos 0x0, ttl 64, id 31657, offset 0, flags [DF], proto TCP (6), length 52)
  192.168.1.102.42908 > sof02s18-in-f46.1e100.net.https: Flags [.], cksum 0x42fc (correct), seq 2715452946, ack 4244149933, win 1444, options [nop,nop,TS val 45552128 ecr 3055914973], length 0
14:50:01.277676 IP (tos 0x0, ttl 64, id 63131, offset 0, flags [DF], proto UDP (17), length 72)
  192.168.1.102.37669 > home-77-70-13-1.megalan.bg.domain: [udp sum ok] 9446+ PTR? 33.212.58.216.in-addr.arpa. (44)
14:50:01.280412 IP (tos 0x0, ttl 58, id 43610, offset 0, flags [none], proto TCP (6), length 52)
  sof02s18-in-f33.1e100.net.https > 192.168.1.102.36614: Flags [.], cksum 0xe699 (correct), seq 1, ack 1, win 472, options [nop,nop,TS val 2919949791 ecr 45483150], length 0
14:50:01.280949 IP (tos 0x0, ttl 59, id 23273, offset 0, flags [none], proto TCP (6), length 52)
  sof02s18-in-f46.1e100.net.https > 192.168.1.102.42908: Flags [.], cksum 0x9ef1 (correct), seq 1, ack 1, win 358, options [nop,nop,TS val 3055961049 ecr 45483596], length 0
14:50:01.283832 IP (tos 0x0, ttl 61, id 37950, offset 0, flags [none], proto UDP (17), length 493)
  home-77-70-13-1.megalan.bg.domain > 192.168.1.102.37669: [udp sum ok] 9446 q: PTR? 33.212.58.216.in-addr.arpa. 2/6/11 33.212.58.216.in-addr.arpa. PTR
  sof02s18-in-f33.1e100.net., 33.212.58.216.in-addr.arpa. PTR sof02s18-in-f1.1e100.net. ns: 216.in-addr.arpa. NS x.arin.net., 216.in-addr.arpa. NS z.arin.
  net., 216.in-addr.arpa. NS y.arin.net., 216.in-addr.arpa. NS u.arin.net., 216.in-addr.arpa. NS r.arin.net., 216.in-addr.arpa. NS arin.authdns.ripe.net. a
  r: u.arin.net. A 204.61.216.50, u.arin.net. AAAA 2001:500:14:6050::ad:1, y.arin.net. A 192.82.134.30, y.arin.net. AAAA 2001:500:127::30, x.arin.net. A 19
  9.71.0.63, x.arin.net. AAAA 2001:500:31::63, z.arin.net. A 199.212.0.63, arin.authdns.ripe.net. A 193.0.9.10, arin.authdns.ripe.net. AAAA 2001:67c:e0::10
  , r.arin.net. A 199.180.180.63, r.arin.net. AAAA 2001:500:f0::63 (465)
```

Фигура 1.6: Изглед на потребителския интерфейс на tcpdump.

Глава 2

Проектиране на структурата на мрежов анализатор

2.1 Функционални изисквания към мрежов анализатор

2.1.1 Намиране и избиране на физически интерфейси

Преди да започне какъвто и да било анализ, мрежовия администратор трябва да получи списък с интерфейси поддържани от мрежовата карта. Списъкът трябва да съдържа освен имената на интерфейсите (напр. `eth0`), конфигурираният IP адрес, мрежова маска с цел по-лесното ориентиране в случай на голям списък. Анализатора трябва да предостави възможност на администратора за избор на конкретен интерфейс, който да бъде анализиран.

2.1.2 Интерпретиране на данни

Мрежовия анализатор трябва да може да интерпретира прихваната от преносвателната среда поредица от байтове. Тъй като типично тя е в двоичен вид, информацията за съдържанието на пренесения PDU трябва да е в разбираем от администратора вид, т.е. представена като символен низ. Фундаментална е функционалността за интерпретиране на най-често срещаните протоколи в TCP/IP стекът, а именно Ethernet, IP, TCP, UDP.

2.1.3 Филтриране на данни

Прихващането на трафик директно от преносвателната среда предполага огромно количество данни, особено в случай на голямо количество станции. Филтрирането позволява на администратора да се абстрахира от ненужния му трафик като настрои анализатора да приема конкретен тип трафик, напр. единствено IP трафик. В противен случай анализирането би отнело ненужно време и ресурси.

2.1.4 Отдалечен анализ

Посредством client-server архитектура, анализатора трябва да поддържа отдалечен анализ. Клиентът трябва да има възможност да задава команди на сървърът, а той да ги изпълнява, делегирайки част от тях на анализатора за изпълнение. Комуникацията между клиента и сървъра трябва да е full-duplex и в реално време с оглед принципа на работа на мрежовия анализатор.

2.1.5 Сигурност на отдалечения анализ

С оглед на допълнителна сигурност, анализатора трябва да използва библиотека за комуникация между сървърът и клиента поддържаща SSL, тъй като евентуалната липса на поддръжка на криптиран трафик би довела до eavesdropping на комуникацията между двете и следователно до анализирания трафик, достигащ до сървъра.

2.1.6 Споделяне на анализа

Анализатора трябва да има възможност за споделяне на анализа, т.е. да поддържа списък от клиенти, които едновременно да получават прихванатия трафик.

2.1.7 Сигурност на споделяне на анализа

Анализатора трябва да поддържа метод за аутентикация, който позволява единствено авторизирани мрежови администратори да достъпват текущата сесия на анализ. Така потребител (независимо злонамерен или не), достигнал случайно до адреса на уеб сървър, не би имал възможност да проследи целия трафик на машината, което би било еквивалентно на огромна дупка в сигурността.

2.1.8 Графичен интерфейс и визуализация

Анализатора трябва да поддържа лесен за използване и интуитивен графичен интерфейс. При създаване на нова сесия на анализ, мрежовия администратор трябва да има възможност за въвеждане на парола, за избиране на физически интерфейс, за въвеждане на филтри и стартиране на анализатора. Прозорецът на сесията трябва да съдържа списък с прихванати пакети и кратка информация за тях, както и компонент показващ детайлното съдържание на избран пакет, представяйки цялата му йерархична структура с оглед моделите разгледани на стр(?).

2.2 Съображения за избор на програмни средства и развойна среда

2.2.1 C++

Езикът C++ е език с общо предназначение, използван предимно в сферата на системното програмиране. Създаден през 1979г. от Бьорн Струструп, езикът е комбинация от механизмите за абстракция на Simula и бързината и ефективността на C. Отчитайки поддръжката на системно програмиране, програмния код написан на C++ лесно взаимодейства със софтуер написан на други езици. Тази необходимост от взаимодействие е отчетена още от началния етап на дизайна на езика и поддръжката на C, Assembler и Fortran не изисква допълнително процесорно време или преобразуване на структурите от данни. [Stroustrup \(2013\)](#) С оглед на факта, че `libc++` е имплементирана на C и характеристиките на C++, езикът е натурален избор взимайки предвид необходимото бързодействие при работата с високоскоростни мрежови връзки.

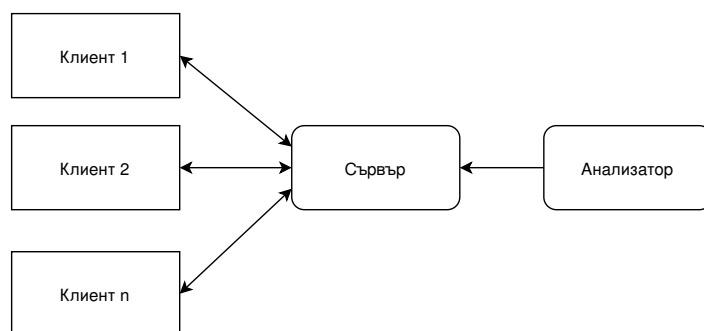
2.2.2 libpcap

2.2.3 Angular 2 и TypeScript

2.3 Проектиране на архитектура на мрежов анализатор с отдалечен достъп

2.3.1 Цялостна архитектура

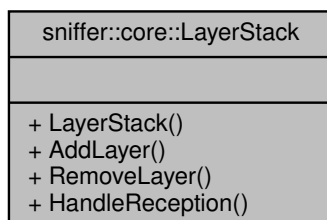
На Figure 2.1 е представен абстрактен поглед върху цялостната архитектура на приложението. Със стрелките е означен потока от данни между клиентите (мрежовите администратори), сървърът и анализаторът. След като е прихванат от преносвателната среда, пакетът се обработва, подава на сървъра и бива изпратен до всички автентикирани клиенти (broadcast). Междувременно комуникацията между сървъра и клиентите е двупосочна (full-duplex) — те могат да изпращат различни команди, напр. да се автентикират към сървъра, да проверят дали съществува сесия, да стартират сесия и т.н.



Фигура 2.1: Абстрактен поглед върху архитектурата

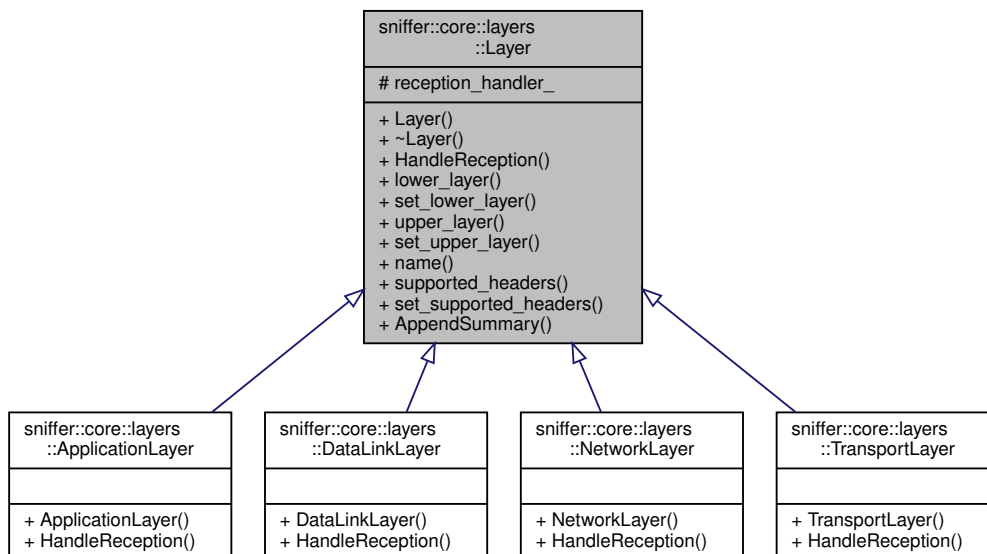
2.3.2 Архитектура на мрежовия анализатор

В основата на архитектурата на мрежовия анализатор са класовете `LayerStack` и `Layer`. Двата класа представляват частично модифицирана имплементация на шаблона за дизайн (design pattern) *протоколен стек*. Решението, освен разделяне (decoupling) на отговорността на слоевете, позволява динамична промяна на слоевете на стека, напр. ако е нужно 'подпъхване' на междинен слой между два вече съществуващи с цел криптиране, дебъгване и т.н. `LayerStack`, публичният интерфейс на когото е описан на Figure 2.2, представлява двусвързан списък от слоеве с възможност за динамично добавяне и премахване.



Фигура 2.2: UML диаграма на класа `LayerStack`

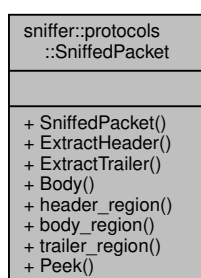
Layer е базов клас за всички слоеве както е представено на Figure 2.3. Индивидуалните слоеве се достъпват чрез указатели от този тип. Предимството е, че конкретния тип на горния и долния слой е неизвестен за имплементацията на даден слой.



Фигура 2.3: UML диаграма на класа **Layer** и наследниците му

Всеки обект от типа **Layer** има списък от поддържани header-и под формата на обекти от типа **HeaderMetadata**. Например, този списък за **TransportLayer** би съдържал обекти от типовете **UserDatagramHeaderMetadata** и **TransmissionControlMetadata**. На ?? е представен класът **HeaderMetadata**. В себе си той съдържа (??????????????).

На Figure 2.4 е показан публичният интерфейс на **SniffedPacket** класа. Той представлява частично модифицирана имплементация на шаблона за дизайн *протоколен пакет*, още известен като *протоколен буфер* или *многослоен буфер*. Взимайки предвид описанието на (?), всеки слой добавя/премахва свой header или trailer. Това предразполага към имплементация, в която всеки слой заделя или освобождава нов буфер отчитайки новия размер. Изложения шаблон дава просто и ефективно решение на този проблем.



Фигура 2.4: UML диаграма на класа **SniffedPacket**

Буферът стандартно се разделя на три области: header, body и trailer. Алгоритъма на декапсулиране, представен визуално на Figure 2.5, е следния:

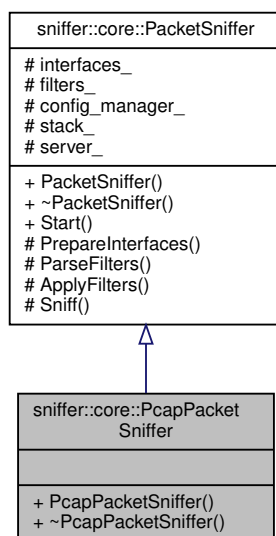
1. Полученият пакет се създава с всички байтове в body областта. В този начален момент, header и trailer областите имат нулева дължина.
2. Слой 1 изважда своите header и trailer области, двете области се "отрязват" от body областта. Размера на body областта се намалява.

3. Слой 2 също изважда своите header и trailer области, двете области отново се "отрязват" от body областта. Размера на body областта се намалява.
4. Аналогичен е процесът при трети слой, след който се получава изначалната body област.



Фигура 2.5: Диаграма на разпределение на областите при получаване на пакет

Представен на Figure 2.6 е абстрактния клас `PacketSniffer` и имплементацията му `PcapPacketSniffer`. Абстракцията е необходима с цел поддръжка на библиотеки на ниско ниво различни от `libpcap`. Публичният интерфейс на класа е прост — метод, който стартира анализатора, като в имплементацията си извиква последователно частни виртуални методи за подготовка на физическите интерфейси, за интерпретиране на зададените филтри и прилагането им (template method). Всеки обект от типа `PacketSniffer` е композиран от обект от типа `LayerStack`. С оглед имплементацията описана в (?), `LayerStack` е де факто аналогичен, паралелен на системния протоколен стек, но локален за мрежовия анализатор.

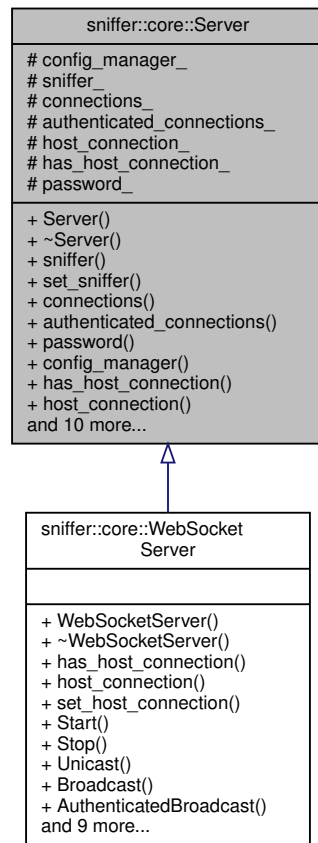


Фигура 2.6: UML диаграма на класа `PacketSniffer`

2.3.3 Архитектура на сървър

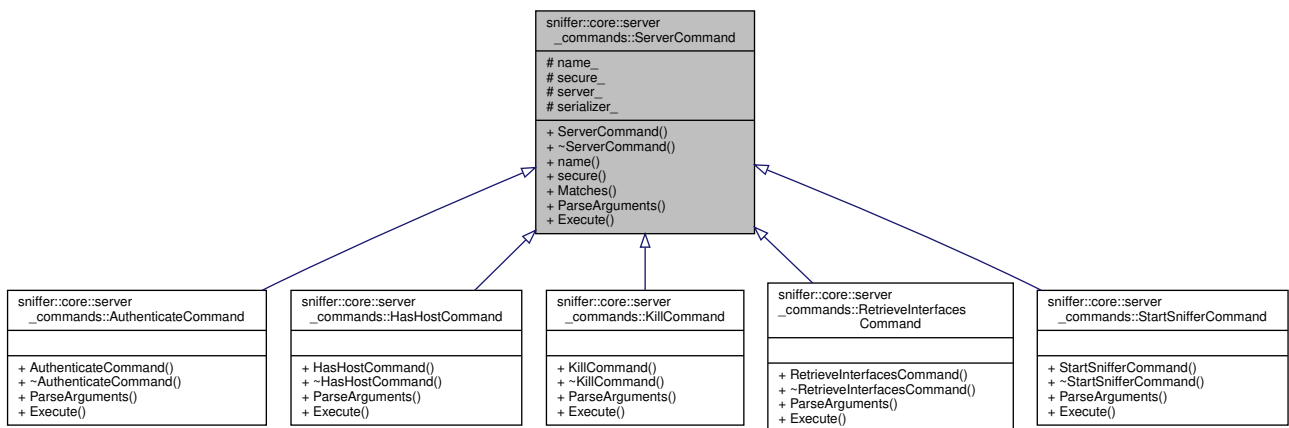
Ядро

В ядрото на сървърната част на цялостната архитектура е `Server` класа. Неговият интерфейс описва абстрактна функционалност за добавяне на нови връзки, за множествено изпращане (broadcast), за автентикация и т.н. Абстракцията на този клас предполага лесна подмяна на `WebSocketServer` класа, представен на Figure 2.7, с друг тип сървър.



Фигура 2.7: UML диаграма на класовете **Server** и **WebSocketServer**

Поддръжка на команди



Фигура 2.8: UML диаграма на класа **ServerCommand** и наследниците му

2.3.4 Архитектура на клиент

Глава 3

Програмна реализация на мрежов анализатор

Глава 4

Ръководство на потребителя

Глава 5

Заключение

Приложение А

Исходен код

Библиография

B. Stroustrup. *The C++ programming language*. Addison-Wesley, Upper Saddle River, NJ, fourth edition edition, 2013. ISBN 978-0-321-56384-2.

A. S. Tanenbaum and D. Wetherall. *Computer networks*. Pearson Prentice Hall, Boston, 5th ed edition, 2011. ISBN 978-0-13-212695-3. OCLC: ocn660087726.

Списък на фигурите

1.1	Модел на пет слойна мрежа	9
1.2	Междуслойна комуникация. Капсулиране и декапсулиране.	10
1.3	Преглед на имплементацията на мрежов анализатор на ниво ядро на операционната система	11
1.4	Изглед на потребителския интерфейс на Wireshark.	13
1.5	Структура на tcpdump команда.	14
1.6	Изглед на потребителския интерфейс на tcpdump.	14
2.1	Абстрактен поглед върху архитектурата	17
2.2	UML диаграма на класа LayerStack	17
2.3	UML диаграма на класа Layer и наследниците му	18
2.4	UML диаграма на класа SniffedPacket	18
2.5	Диаграма на разпределение на областите при получаване на пакет	19
2.6	UML диаграма на класа PacketSniffer	19
2.7	UML диаграма на класовете Server и WebSocketServer	20
2.8	UML диаграма на класа ServerCommand и наследниците му	20

Списък на таблиците