ТЕХНОЛОГИЧНО УЧИЛИЩЕ ЕЛЕКТРОННИ СИСТЕМИ
към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ

ДИПЛОМНА РАБОТА

Тема: Анализатор на мрежови трафик

Дипломант:
Андон Мицов

Научен ръководител:
гл.ас. Любомир Чорбаджиев

София

2014

Съдържание

Увод

В тази дипломна работа ще представя приложение анализиращо трафика в една локална мрежа. Идеята на това приложение е да разглежда пакетите от TCP/IP стек минаващи през мрежовата карта на компютъра. Това е много полезно за намиране на проблеми с мрежата и за прихващане на опити за проникване в системата от външни лица.

I Smell Something Funny (ISSF) е приложение, което предоставя възможност на потребителите си да разгледат информацията съдържаща се в хеадърите на пакетите, които влизат или излизат от техния компютър, изложена на графичен потребителски интерфейс. Това е бърз и удобен начин за намиране и анализиране на цялата информация за пакета. Предоставя възможност за филтриране на получените пакети, т. е. за избиране на кои пакети ще се разглеждат хедърите.

Използвани са технологии, които са достъпни и безплатни за всички. Основните са C++, pcap библиотеката и Qt framework. Приложението е имплементирано на езика C++ и използва pcap библиотеката за да получи достъп до мрежовата карта и да хваща пакетите минаващи през нея. То използва Qt за имплементацията на потребителския интерфейс. Приложението се придържа към отворените стандарти и спазва конвенциите за програмиране на C++.

Глава I:

Методи и технологии за мониторинг на мрежови трафик

1. Компютърни мрежи.

Компютърните мрежи са съвкупност от 2 или повече компютри или електронни уреди като принтери, скенери и други. Тяхната основна функционалност е предоставянето на потребителите връзка и достъп до ресурсите на отдалечени станции.

1.1. Видове компютърни мрежи.

- Локални мрежи (Local Area Network – LAN) – локалните мрежи са мрежи намиращи се в малки географски райони. Състоят се от 100 или по-малко станции. Имат широколентова скорост от 10 Mbit/s до 100 Mbit/s. Те могат да бъдат изградени с кабели или без кабели с помощта на радио сигнали(Wireless LAN – WLAN). Използваните кабели могат да са коаксиални кабели или кабели с осукана двойка.

- Мрежа през голяма област (Wide Area Network - WAN) – това са копютърни мрежи обхващащи големи географски региони. Те могат да бъдат поставени на различни места по света. Може да се смята, че ако една мрежа минава границите на няколко държави, то тя е мрежа през голяма област (например мрежа състояща се от станции, които се намират в България, Гърция и Румъния). Такава мрежа може да се разгледа като мрежа, свързваща множество локални мрежи. Използваните скорости са от 1Gbps до 100Gbps.

1.2. Устройства управляващи мрежовия трафик.

- Маршарутизатор (router) - устройство управляващо трафика между няколко локални мрежи. Маршарутизатора работи с протоколи от TCP/IP модела, по-точно с протоколи принадлежащи на физическия и мрежови слой (слой 1, 2 и 3 на OSI модела). Като разгледа информацията, която получава от получения пакет, определя към коя мрежа да го прати като, специфично IP адреса на получателя. Използва се за да раздели едно голямо адресно пространство на по-малки части, които са по-лесно поддържани. Някои

маршарутизатори са с вградени точки за достъп за безжична мрежа.

- Комотатори (switch) - това са устройства, които управляват трафика в локална мрежа. Работят с протоколи от физическия слой на TCP/IP (слой 1 и 2 на OSI модела). Определя към кой възел да прати получения пакет, като разгледа физическия адрес на получателя. То може да поддържа много повече връзки от маршарутизатора. Неговата функционалност е по-ограничена, но за сметка на това работи много по бързо от маршарутизатора. Може да се използва, за да раздели локалната мрежа на виртуални мрежи (VLAN).

- Точка за достъп за безжична мрежа (Access Point) - това е устройство, което позволява на безжични уреди достъп до жичната мрежа. През него се приемат пакети от безжичната за жичната мрежа и обратното. Може да се използва като отделно устройство или да е вградено в маршарутизатор.

2. Какво е пакет от данни?

Устройствата в компютърните мрежи си комуникират един с друг като изпращат физически сигнали. Логически информацията в тези сигнали се разглежда от устройствата като пакет от данни. В тези пакети се съдържа не само информация за потребителя на компютъра, но информация записана от протокола, който го изпраща.

3. Анализирането на пакети

Анализирането на пакети познато също като хващане на пакети е процес, в който машината копира получен пакет и дава това копие на приложение, което анализира информацията в него (tcpdump, Wireshark).

4. Модели на работата на компютърните мрежи

За да работи една компютърна мрежа удовлетворително, тя трябва да следва определен модел за работа. Има два стандартни модела: TCP/IP и OSI (Open

Systems Interconnection). Протоколите, които те обхващат са абстрахирани в слоеве в зависимост от тяхното предназначение.

4.1. TCP/IP модел

Интернет протоколен модел, е първият модел на работа разработен през 70-те години на 20 век. Основните принципи са взети от разработката на ARPANET. Това е отворен взаимно свързан модел на работа. Протоколите, които са част от него, са разделени в 4 слоя в зависимост от тяхната спецификация.

Слоевете са: физически, интернет, транспортен и приложен. Физическия слой отговаря за трансмисията на данни и управлението на достъпа до преносвателната среда. Интернет слоя отговаря за изпращането на пакетите по мрежата към правилният хост. Транспорт слоя отговаря за управлението на връзката между различните хостове, големината на изпратените пакети, кога и на какъв процес от приложния слой да се предаде получения пакет. Приложният слой отговаря за представянето на получената информация на потребителя. Протоколите принадлежащи на TCP/IP модела са:

- Интерфейсен слой: Ethernet, WLAN (802.11), Token Ring;
- Интернет слой: Internet Protocol (IP), IPSec;
-Транспортен слой: Transmission Controll Protocol (TCP), User Datagram Protocol (UDP), ICMP;
- Приложен слой: Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), Domain Name System (DNS), Simple Mail Transfer Protocol (SMTP), Simple Network Management Protocol (SMNP).

4.2. OSI (Open Systems Interconnection) модел

Този модел е разработен от Open Systems Interconnection проекта, принадлежащ на Международната Организация за Стандарти. Това е общо предназначен модел на работа, който описва как компютрите комуникират един с друг през една мрежа. Той поддържа всички протоколи от TCP/IP допълнително с други.

Протоколите са абстрахирани в 7 слоя: приложен, презентационен, сесиен,

транспортен, мрежови, канален и физически. Както при TCP/IP, когато пакетите трябва да се изпратят през мрежата те преминават през всеки един слой и в зависимост от какъв протокол се използва, пакетите се енкапсулират като се добавя до тях допълнителна информация. Слоевете са повече от тези на по-стария модел на работа, но групирани работят аналогично: приложния, презентационния и сесийния слой на OSI отговарят на приложния слой на TCP/IP. Транспортният слой е един и същ в двата модела. Мрежовият и интернет слоевете са аналогични. Функционалностите на физическия и канален слой са обединени в слоя за мрежови достъп.

Всеки едно ниво има определено предназначение:

- Приложният (application) слой отговаря за представянето на данните на потребителя представянето на този потребител връзка с мрежата.

- Презентационният (presentation) слой отговаря за форматирането на информацията в зависимост от какви протоколи ще се използват на по-ниски нива.

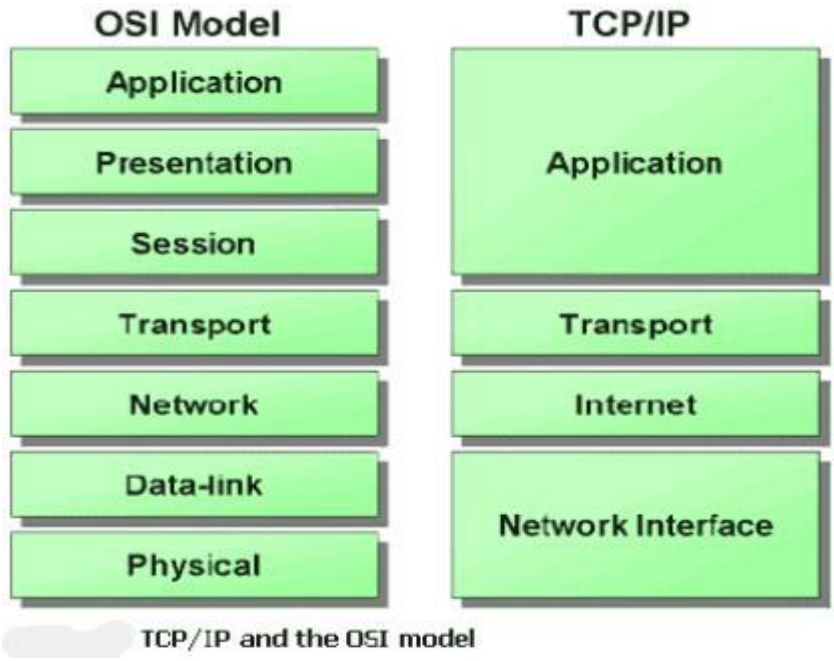
- Сесийното (session) ниво установяване, поддържане и терминиране на връзките с други устройства. Функционалността на транспортния слой се състои в това да осигури успешно предаване и получаване на информацията.

- Третото ниво (network) отговаря за насочване на пакетите с помощта на логическите адреси.

- Каналният (data-link layer) слой приготвя пакетите за изпращане по преносвателната среда. Протоколите на това ниво отговарят за достъпа до преносвателната среда, за приемането, получаването и отхвърлянето на пакетите.

- Физическото ниво представлява съвкупността от всички устройства и преносвателни среди участващи в пренасянето на

сигналите.



Фиг.1.1 Сравнение на слоевете на TCP/IP модела с тези на OSI модела

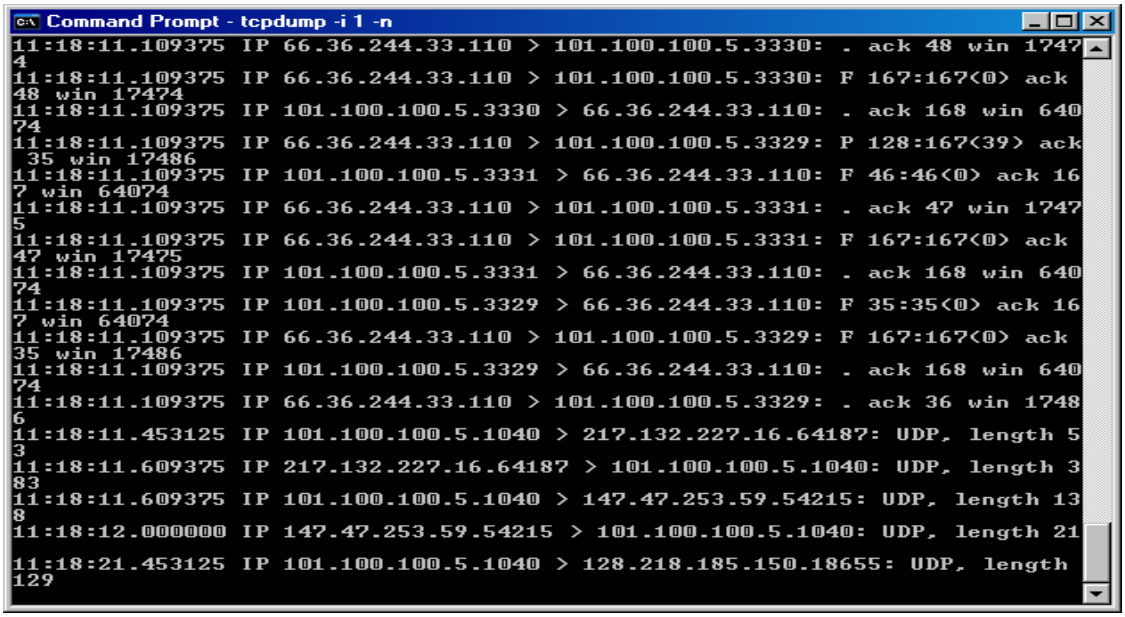
5. Приложения, анализиращи трафика на информация в една компютърна мрежа

При проектирането на компютърни мрежи възникват проблеми, чиито произход трудно се открива. За да се намери този проблем се анализира информацията, която минава през мрежата. За тази цел са създадени приложения, с които се анализира трафика в мрежата (пакетни анализатори)

5.1. tcpdump - конзолно приложение за анализиране на пакети

tcpdump е безплатно приложение с отворен код разработено за операционни системи от семейството на UNIX и разпространено под BSD лицензa. То принтира на конзолата описание на информацията, която се съдържа в пакетите, минаващи през избрания интерфейс. Има опция за филтрация, с която се избира какви пакети да се разглеждат.

Данните се представят в зависимост от това какви опции се използват. Ако не е избран филтър tcpdump трябва да изкара цялата информация за пакета. При използване на филтър се показва информация важна за избрания протокол.

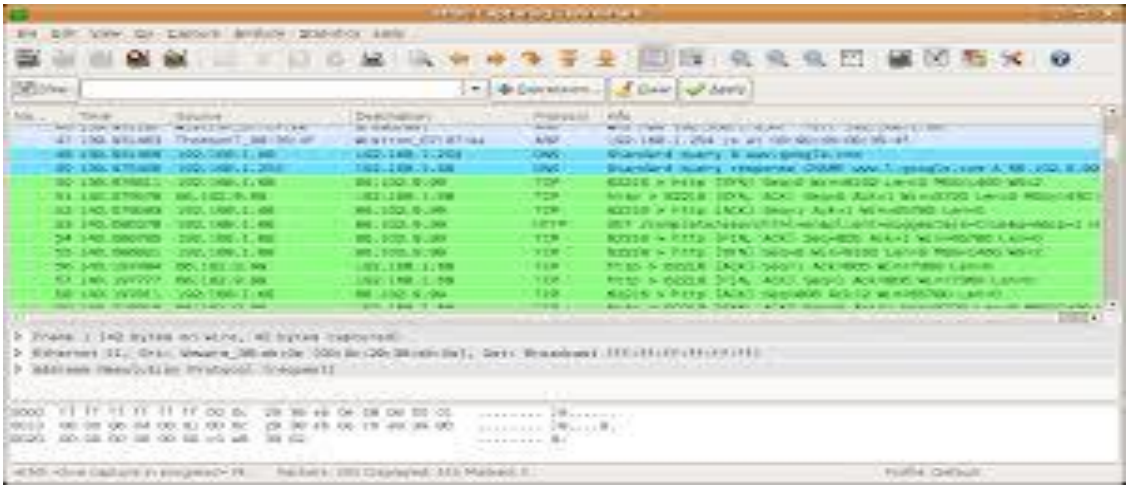


Фиг. 1.2 Информация изведена от tcpdump

tcpdump е много ефективно приложение за анализиране на мрежови трафик, но е неудобно за работа, карайки потребителя да въвежда нужната информация на ръка и понякога трудно се разбира изведената информация. Но то все още си остава едно от най-добрите начини за откриване на проблеми в мрежата.

5.2. Wireshark - пакетен анализатор с потребителски интерфейс.

Wireshark Network Analyzer е безплатно приложение с отворен код за анализиране на мрежови трафик. То работи подобно на tcpdump. Обаче за разлика от него, за изобразяване на информацията от хванатите пакети и за настройването на хващането се използва потребителски интерфейс. Това прави Wireshark по-предпочитан за работа от tcpdump.



Фиг. 1.3Wireshark

Потребителят избира през кой интерфейс да се слуша за пакети и те са представени в текстово поле в момента, в който са хванати. Изписана е информацията за първия протокол, пакетирал данните. Когато се избере даден пакет, цялата информация за него е показана в друго поле т.е. хеадърите на всеки един протокол участвал в енкапсулацията на пакета се изписва там: Ethernet, IP, TCP, HTTP или други. Wireshark има своя имплементация на пакетна филтрация. Представя се булев израз, по който се филтрират пакетите. Пример: “ip.src_address == 192.168.0.1” - това казва на Wireshark да разглежда пакети с адрес 192.168.0.1 на изпращане.

6. Технологии за имплементация на анализатори на мрежови трафик

За да се имплементира правилно един пакетен анализатор, то трябва да използва правилните библиотеки.

Основната библиотека, която всеки мрежови анализатор трябва да има, трябва да позволява на приложението достъп до мрежовата карта за слушане за преминаващи пакети. Такива библиотеки са pcap и WinPcap.

Ако искаме да създадем приложение с потребителски интерфейс трябва да използваме приложно програмни интерфейси с такава спецификация. Това са: Qt, GTK+ и WPF.

Трябва също да се спомене, че ако искаме да създадем пакетен анализатор с потребителски интерфейс, приложението трябва да поддържа нишки. Една нишка за потребителския интерфейс, и една която да слуша за пакети. Библиотеки с такава възможност са: boost, POSIX, WinAPI.

6.1. pcap и WinPcap.

pcap е библиотека с отворен код разработена през 90те години на XX век от разработчици от Лаборатория Лауренц Бъркли при създаването tcpdump. Това е

библиотека, написана на C предоставяща достъп на ниско ниво за хващане на пакети. Използва се от протоколни анализатори, мрежови монитори и системи за разпознаване на интрузия в мрежата. Тя работи за операционни системи от тип UNIX и изисква администраторски права. WinPcap е портната версия на pcap за Windows.

pcap предоставя функции за достъп до всички мрежови адаптери на компютъра. Всеки един от тях може да бъде отворен в определен режим и да му се зададе колко време да слуша за пакети и колко голяма част от тяхната информация да се разглежда.

Библиотеката съдържа 2 метода, с които се хваща пакет: pcap_next() и pcap_loop(). Първият метод връща указател към сегмент в паметта където се намира копие на хванатия пакет. Вторият метод представлява цикъл, който слуша мрежовия интерфейс за пакети, и всеки път като хване преминаващ пакет вика функцията, подадена му като аргумент. На тази функция като аргументи се подават указател към структура, съдържаща информация за големината на пакета, часа в който е хваната и големината на хванатата част, и указател към сегмента от паметта, където се намира копието на хванатия пакет. В тази функция може да се анализира целият пакет.

Тук също така е реализирана опцията за филтрация. На метода pcap_compile() се подава низ от знаци, който е компилиран във филтърна програма. След това тази програма се подава на метода pcap_setfilter(), който го свързва с отворен pcap интерфейс.

6.2. Qt - C++ framework за разработване на потребителски интерфейс.

Qt е cross-platform framework с отворен код, имплементиран на C++, за разработване на графичен потребителски интерфейс. То идва със собствен IDE QtCreator или може да се използва като библиотека от други IDE-та.

Фиг. 1.4 QtCreator

Qt има собствен paint engine, който емулира вида на различните платформи. Това го прави по-лесно за портване. То използва програмните интерфейси на операционните системи, за да получи нужната информация за чертаенето на своите widget-и.

Qt има собствен Meta Object Compiler (MOC), който следейки за специфични макроси в кода, генерира допълнително C++ код с мета информация за класовете на приложението. Това позволява да се използват допълнителна функционалност, която не съществува в нормалния C++ като: сигнали и слотове, асинхронни функции и интроспекция.

Системата за сигнали и слотове на Qt предоставя удобен начин за комуникация между различните обекти в приложението. Идеята е много проста: един обект свързва специална функция, наречена сигнал с друга функция на друг обект,

наречена слот. Когато първият обект изпрати сигнал, другия обект вика слота. Qt има собствена имплементация на event и event listener системи, но сигналите и слотовете са много по-бърз и прост начин за работа.

6.3 GTK+ - GIMP Toolkit

GTK+ е cross-platform widget toolkit с отворен код за създаване на графичен потребителски интерфейс. Разпространява се под GNU LGPL лицензa. То е подобрена версия GTK widget toolkit. Създаден е през 1997г. от Спенсър Кимбал и Питър Матис за GIMP.

GTK+ е обектно-ориентиран toolkit написан първоначално на C/C++. Има поддръжка за Java, Python, JavaScript, Ruby и Lua.

GTK+ е с вградена GLib библиотека за подобряването на преносимостта между различните платформи. Последният компонент на GTK+ е Pango библиотеката за изписване на текст на различни езици.

6.4. WPF - Windows Presentation Foundation

WPF е подсистема на .NET framework за разработване на графичен потребителски интерфейс за Windows приложения. Това е система, с която е много удобно и бързо да се работи.

То използва Direct 3D за чертаене на графичния интерфейс. Използвайки Direct 3D позволява да се представи по-голям набор от теми и по-добри графики. Това е много добро решение, защото оставя част от работа на приложението на GPU-то.

Библиотеката идва с вградена функционалност за поддържане на медийни услуги като изображения, аудио и видео. Поддържат се: BMP, JPEG, TIFF, GIF, PNG, Windows Media Photo, Icon, MPEG, AVI и WMV.

WPF има функционалност за използване на шаблони. Шаблоните се използват за

дефиниране на различен облик за елементите на приложението.

6.5. Boost - сбор от библиотеки за разработване на приложения

Boost е сбор от библиотеки написани на C/C++ за подпомагането на разработването на приложения. Съдържа над 80 библиотеки. Те варират от библиотеки с обща цел като smart pointers библиотека до библиотеки, които са абстракции на библиотеките на операционната система. като Boost FileSystem и до библиотеки за Template Metaprogramming (TMP) и domain-specific language(DSL).

Boost има библиотека за работа с нишки. Това е абстракция на API на операционната система, която се занимава с това. Използваният клас boost::thread при конструирането на обект му се подава указател към функция и започва нова нишка.

6.6. Pthread

Pthread е част POSIX стандартите за UNIX написано е на C без ООП. Това е API, който определя начина за създаване на нишки на операционни системи от тип UNIX. Използват се структури за съдържане на информацията за нишката. Подобно на boost като се създаде нишка с Pthread тя се пуска веднага. Използва се ако създателите на приложението искат да е по оптимизирано за операционни системи от тип UNIX.

6.7. Windows API

Windows API или WinAPI е сбор от библиотеки за разработване на приложения за Windows написана на C/C++. То позволява достъп до ресурсите на операционната система и до специфични за системата функционалности. WinAPI предоставя възможност за създаване на нишки в Windows. Създаването на нишки не е обектно ориентирано също като при Pthread. При създаване на нишки може да се определи нивото на сигурност на нишката, големината на стека и дали да се пусне веднага след създаването си. WinAPI е най-добрият начин за създаване на оптимизирани приложения, които работят с много нишки, за Windows.

Глава II.

Имплементация на анализатор на мрежов трафик

1. Потребност от анализатор на мрежови трафик

Пакетните анализатори откриват голямо приложение. Те се използват не само за диагностициране на проблеми в компютърните мрежи, но и в тестването на сигурността на мрежата (penetration testing).

Примери за неговата употреба са:

- Проверяване за други устройства, които са се свързали с точката за достъп на безжична компютърна мрежа.
- Търсене на установени TCP връзки с други компютри.
- Следене на трафика в една отворена безжична мрежа.
- Следене за пакети с IP адреси, които не трябва да бъдат изпращани на определени адреси
- Наблюдаване на мрежата при изпращане на деаутентикиращи пакети при тестване на сигурността на мрежата.
- Откриване на опити за постигане на remote access чрез telnet.

Идеята на приложението е да се анализират пакетите, които се движат в мрежата. По този начин може да се определи дали има опити за връзка с компютъра, като се разгледа информацията в хеадърите на пакетите с изпратени от TCP протокола.

В безжични мрежи може да се открие BSSID на скрити мрежи и да се определи към коя точка за достъп да се свърже. Или да се открият опити за проникване в безжична мрежа.

Потребителят може да разгледа кои пакети от кои хостове идват или кои хостове изпращат broadcast и multicast съобщения.

Може да се разгледа цялата информация в HTTP хеадърите на пакетите, които се изпращат и приемат от компютъра и да се види какви сайтове се посещават. Това е много удобно при разработването на web services, които не изискват работа с browser, например мобилни client-server приложения.

2. Изисквания към анализатор на мрежов трафик

2.1. Приложението трябва да използва библиотеката pcap

Пакетният анализатор трябва да използва библиотеката pcap за хващане на пакети минаващи през мрежовата карта. Без нея основната функционалност на приложението няма изобщо да съществува. То трябва да използва основната функционалност, предоставена от pcap, т.е. да хваща пакети, които минават през мрежовата карта. След това тяхната информация трябва да бъде анализирана.

2.2. Приложенията трябва да анализира пакети от TCP/IP модел

Главната услуга, която предоставя това приложение е да представи информацията от получените пакети на потребителя в достъпен за четене формат. При получаването на пакетите те са само масив от байтове. Те трябва да бъдат превърнати в разбираем за хората формат. За това приложението трябва да ги раздели на части и да ги "преведе" в думи, разбираеми за потребителя.

2.3. Опция за филтриране на пакетите, минаващи през мрежовата карта.

ISSF трябва да може да поддържа филтрация на пакетите. Това се постига като се използва в pcap вградената опция за филтриране на пакети. Без него пакетния анализатор ще показва огромен поток от информация, който е много труден за разчитане. Ако потребителя иска да търси пакети от точно определен тип, ще трябва да следи много внимателно изписаната информация. Очевидно е, че това е най-неудобният начин за работа. Много по-лесно е да се приложи филтър, който да пропуска само пакетите, които искаме да видим.

2.4. Графичен потребителски интерфейс имплементиран с Qt

Приложението трябва да има добре разработен графичен потребителски интерфейс, на който да се извежда информацията за хванатите пакети. Трябва да предостави на потребителя възможност за спиране и пускане на процеса на хващане на пакети. Трябва да предостави две текстови полета за извеждане на информация: едно за представяне на всички пакети и едно за представяне на подробна информация за избран пакет. Също така ще предостави възможност за

избиране на точно определен интерфейс за слушане. В графичния интерфейс ще бъде имплементирано текстово поле за писане на филтри.

3. Използвани технологии и средства при разработването на приложението.

3.1. Езика C++

Езика C++ е обектно-ориентиран език създаден през 1979г. от Бярне Строуструп. Той е наследник на C и има функционалности близки до неговите. Това позволява да се използват библиотеки написани на C. При създаването на ISSF е използван този език, заради бързината си, възможността да използва pcap и Qt и обектно-ориентираната си насоченост.

3.2. pcap

pcap е библиотека, написана на C през 90-те години на XXв. Тя позволява достъп до мрежовата карта и е основната библиотека, която се използва за бързо и лесно хващане на пакети. Тя се използва и за имплементирането на филтрацията, тъй като тя прилага вече компилирания филтър върху мрежовата карта.

3.3. Qt

Qt е framework за разработването на графичен потребителски интерфейс написан на C++. Системата му за сигнали и слотове предлага удобен начин за реализиране на всички функционалности на приложението.

3.4. boost

Проектът използва функционалността на boost за използване на нишки за реализирането на multithreading в приложението. То предлага удобен начин за реализирането на поддръжката на множество нишки, като реализацията се абстрахира от native API на операционната система.

3.5. QtCreator

QtCreator е IDE направено да представя на своите потребители възможност за бързо и лесно създаване на приложения с графичен потребителски интерфейс. Всеки проект създаден с QtCreator има вградена поддъжка на Qt. Идва със собствен дизайнер за потребителски интерфейс. Целият потребителски интерфейс на приложението е направен по този начин.

3.6. Github.

Проектът използва git version control system за управление на сорс кода. Проектът се съхранява в онлайн хранилището Github.Линк:https://github.com/dmitsov/ISSF

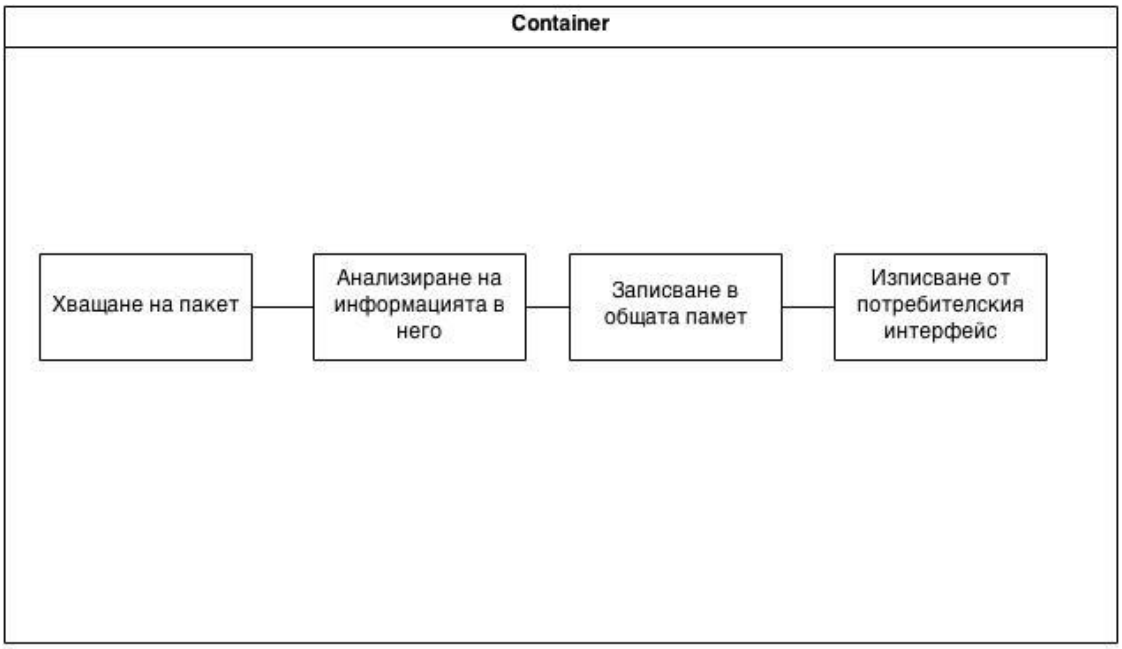
4. Имплементация на пакетен анализатор

4.1. Общ план на имплементацията на приложението

Приложението I Smell Something Funny има една основна функционалност - да извежда информацията за хванатите пакети от избрания порт на потребителски интерфейс. Това се постига, като се създадат две нишки: една за хващане и анализиране на пакетите и втора за потребителския интерфейс. Първо трябва да се хванат и разгледат пакетите и след това да се изпрати анализираната информация на нишката за потребителския интерфейс. Това изисква поддръжка на синхронизация на нишките и поддръжка на обща памет, която да съдържа хванатите пакети.

Алгоритъмът е следният:

1. Хваща се пакетът, минаващ през мрежовата карта и се анализира;
2. Записва се в контейнър, който е общ и за двете нишки;
3. Изпраща се сигнал до потребителския интерфейс, за да се изпише последният хванат пакет;
4. Изписва се получената информация;



,

Фиг 2.1.Блокова диаграма на работата на приложението

4.2. Хващане и анализиране на пакети

Първото нещо, което приложението прави е да отвори интерфейс, на който ще се слуша за пакети, и да пусне нишката за хващане на пакети. Това става с методът на pcap pcap_open_live(). На него се подава като cstring името на интерфейс и се казва в какъв режим да бъде. То връща указател към структура от тип pcap_t съдържаща информация за отворения интерфейс.
Този указател се подава на метода pcap_loop(). Той вика при всеки хванат пакет callback функцията PcapHandle().

PcapHandle() получава като аргументи указател от тип u_char към част от паметта, съдържащ хванатия пакет. Този указател се подава като аргумент на метода на класа SharedMemoryController, който управлява общата памет. Там този указател се използва от конструктура на класа Packet. Създаденият обект се добавя до специално направения контейнър, който е обект от класа Packet PacketVctr.

Класа Packet анализира цялата информация от тази част на паметта, след което нулира този указател, за да предотврати segmentation fault. Този клас съдържа като частни променливи указатели към други класове, отговорни за

анализирането от хеадърите на различни протоколи: EtherFrame, WLANFrame, IPPacket, TCPSegment.

Всеки един от тези класове копира част от байтовете на хванатия пакет в структура, която съдържа информацията за хеадърите на избрания протокол. Тези структури са:

- ether_info за EtherFrame
- wlan_info за WLANFrame
- ip_info за IPPacket
- tcp_info за TCPSegment

Като се завърши записването на пакета в общата памет се изпраща сигнал чрез указателя към обекта, който управлява потребителския интерфейс от тип ISSFApplication наследил QApplication. Този сигнал казва на нишката, управляваща потребителския интерфейс, че е пристигнал нов пакет и трябва да се изпише.

4.3. Имплементация на графичния потребителски интерфейс.

Графичният потребителски интерфейс е създаден с помощта на вградения в QtCreator дизайнер. Смислово той може да се раздели на 3 части: бутони управляващи хващането на пакети, част за филтрите и част за изписване на информацията, получена от хедърите на пакетите.
Бутоните за управление на хващането на пакети са: Interface, Play, Pause и Stop. Всеки един от тях изпраща сигнал, като е натиснат към определен слот на класа ISSFApplication.

Interface бутона при натискането си праща сигнал към слота onInterface, който създава прозорец с всички имена на мрежови адаптери на компютъра. Потребителят маркира името на интерфейса, който иска и натиска Ok бутона на прозореца, за да го избере. Cancel бутона на прозореца го затваря.

Play бутона изпраща сигнал към слота на ISSFApplication onStartThread. Той създава нишка за слушане на пакети.

При натискането на Pause се извиква слота onPauseThread, който спира работата на нишката, но не изчиства нито едно от текстовите полета и не запазва получената информация във файла log.txt.

Stop при своето натискане извиква слота onStopThread. Този слот спира напълно нишката за слушане. Създава прозорец, който запитва потребителя дали иска да запази цялата информация от хедърите на пакетите, и къде да запази тази информация.

В секцията на филтрите се намира текстово поле от тип QTextEdit за въвеждане на текст. При натискането на бутона Apply се взима текста в полето и чрез член променливата, която е указател към обект от клас Filter се извиква метода compileFilter. Този метод връща указател към структура bpf_program. Тази структура съдържа цялата информация за компилирания филтър.

Но преди да се използва филтъра се спира нишката за слушане. След това този филтър се прилага върху отворения интерфейс, чрез метода pcap_setfilter. Като стане всичко това се пуска нишката на ново.

При получаване на нови пакети, нишката за анализиране изпраща сигнал до ISSFApplication. Към този сигнал е свързан слота onPacketsCaptured. В него с метода на SharedMemoryController getLatesttPacket се получава константна препратка към последния получен пакет. След това с метода AbridgedHeaderInfo се получава стринг, съдържащ съкратена информация за пакета. Този стринг се използва за създаване на обект от тип QString. Този обект се добавя до текстовото поле за пакети чрез указателя m_PacketsCptrdField.

Когато искаме да разгледаме цялата информация за определен пакет, първо маркираме реда на който е изписан и натискаме бутона Show Full Info. Този бутон не спира работата на нишката за слушане, но спира изписването на получените пакети и изписва цялата информация за пакета в долното текстово поле.

При натискането се изпраща сигнал до слота onShowFullInfo. Този слот взима маркирания ред и открива кой пакет трябва да се разгледа. След това с методът на SharedMemoryCotroller се взима константна препратка към този пакет. Тук се използва метода HeaderInfo, който връща стринг с цялата информация за пакета. Този стринг се използва, за да се създаде QString, който се изписва на полето. Ако искаме да се продължи с изписването на пакетите, натискаме бутона Continue.

4.4. Имплементация на филтрарането на пакети

Създаването на филтри се извършва класа Filter. Обект от този клас е част от класа ISSFApplication. Когато искаме да създадем филтър, подаваме стринг, взет от текстовото поле. Този стринг е разделен на части: протокол, поле в хедъра, тип на операцията и стойност. Тези стрингове се подават на частния метод MakeFilter. Той съставя стринга за компилиране.

MakeFilter проверява за какъв протокол трябва да се направи стринг. След това се използват статичните методи на класове, които съставят стрингове определени протоколи: FilterEther, FilterIP, FilterTCP.

Всеки един от тези методи определя за кое поле от хедърите става дума и викат частни статични методи, на които се подават типа на операцията и стойността. Всеки един от тези методи създава стринг, който започва с името на протокола и частните методи го довършват.

След като се състави стринг за филтър частната променлива m_filter го присвоява. След това се създава динамично структура от тип bpf_program и се използва метода pcap_compile. Структурата съдържаща информация за компилираната програма се връща от CompileFilter.

5. Основни класове на приложението

5.1. Структури, съдържащи информация за хедърите на пакетите

Използваните структури за съдържане на информация са: ether_info, wlan_info, ip_info, tcp_info. Всяка една от тези структури кореспондира на хедърите на определен протокол: Ethernet, WLAN, IP и TCP.

Фиг2.1 Структури съдържащи информация за хедърите на пакета

5.2. Класове на анализирането на пакети

Основните класове за анализиране на получените пакети са Packet, EtherFrame, WLANFrame, IPPacket, TCPSegment и пакет. AbstrachHeader е абстрактен клас, който се наследява от всички класове за анализация, а LinkLayerData е клас, който е наследен от класове разглеждащи Layer 2 информация. Те съдържат в себе си като член променливи структурите, съдържащи информация за пакетите. Методите на тези класове предоставят достъп до полетата на структурите. Класът Packet обединява всички останали и главно чрез него се интерпретира получената информация.

Фиг 2.2.Класове използвани в анализирането на хванатите пакети

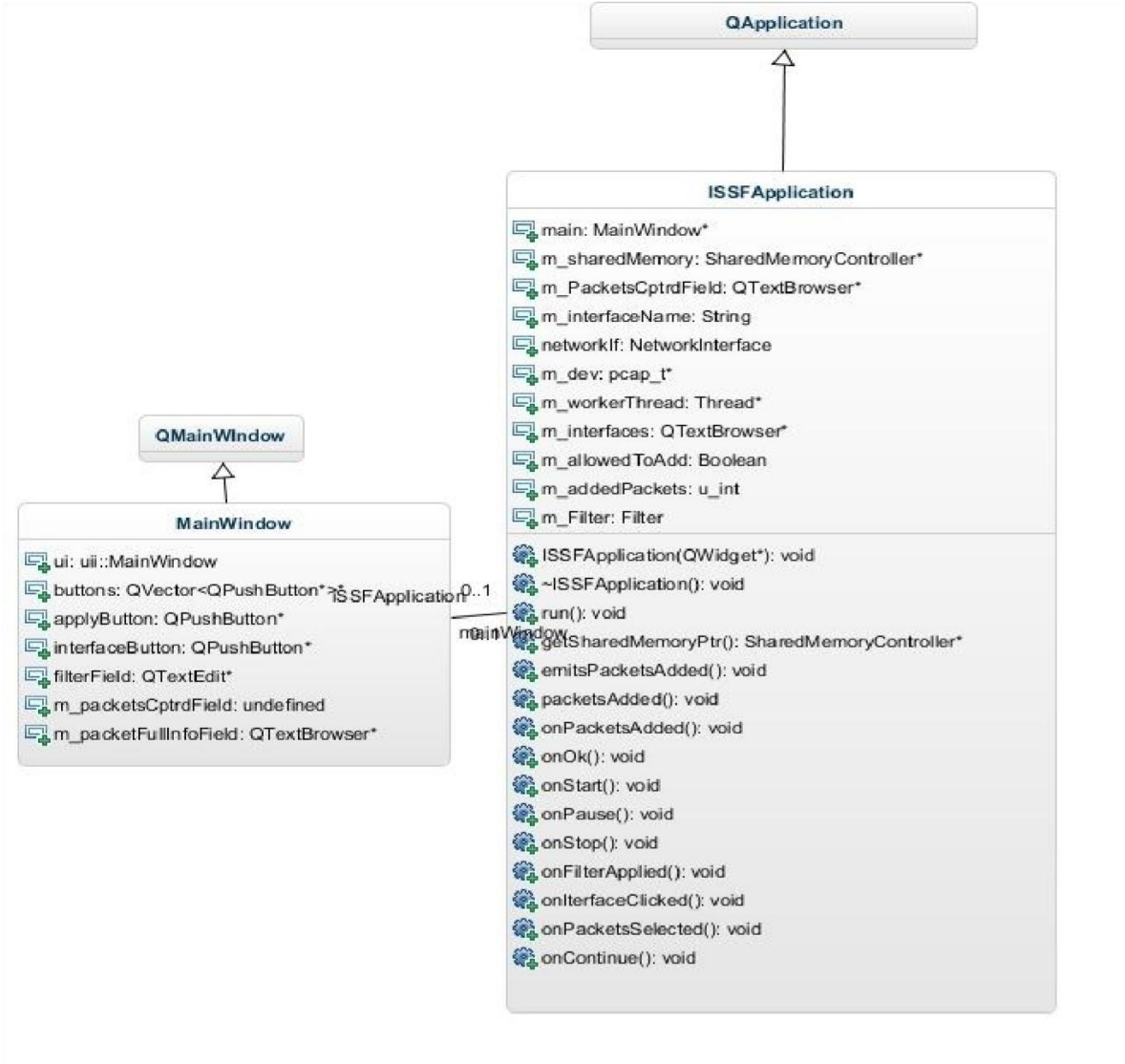
5.3. Класове, отговорни за филтрацията

Класовете отговорни за филтрацията са: Filter, FilterEther, FilterIP, FilterTCP. Filter създава филтър програма, която се прилага върху използвания интерфейс. Другите три класа съставят стринговете, които се използват от Filter при компилирането на филтър програмата. Всеки един тези класове създава стринг за съответен протокол.

Фиг 2.3 Класове съставящи филтри

5.4. Класове за графичния потребителски интерфейс

Класовете, който отговарят за потребителския интерфейс са ISSFApplication и MainWindow. ISSFApplication е класа, в който е реализирана цялата логика на потребителския интерфейс. Той изписва цялата информация за хванатите пакети, пуска и спира слушането, прилага филтрите. MainWindow е клас създаден от QtCreator, който предоставя достъп до елементите на потребителския интерфейс.



Фиг.2.4 Класове съдържащи логиката на потребителския интерфейс.

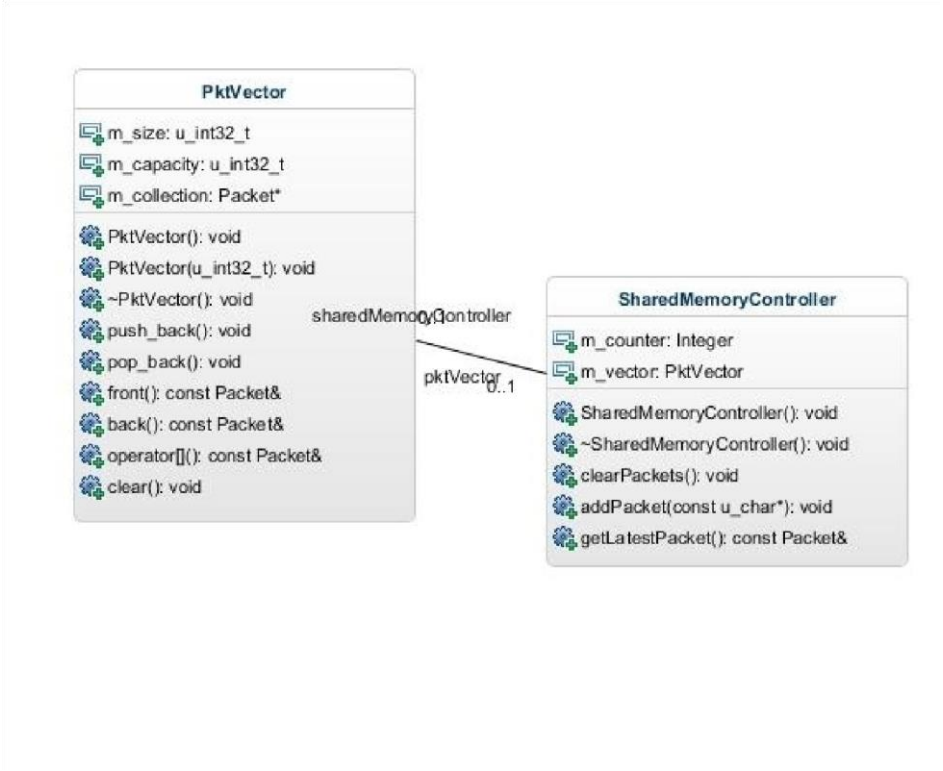
5.5. Класове за намиране на имената на интерфейсите на приложението и за поддръжката на нишки.

Класът NetworkInterface намира имената на всички мрежови интерфейси на компютъра. Класът Thread създава и управлява нишки. И двата класа се използват в ISSFApplication.

Фиг2.5. Класове използвани за избиране на интерфейс и пускане на нишки

5.6. Класове за управление на общата памет.

Класовете за управление на общата памет са SharedMemoryController и контейнерът PktVector. SharedMemoryController управлява общата за двете нишки памет. Той записва в нея получените пакети и предоставя достъп до тях. PktVector е тип специално направен контейнер за пакети. В него се съдържат всички хванати пакети.



Фиг.2.6 Класове управляващи общата памет

Глава III.

Програмна реализация на анализатор за мрежов трафик.

1.Намиране на имената на мрежови интерфейси.

Когато приложението трябва да намери имената на всички интерфейси на компютъра, използваме метода на класа NetworkInterface findDevices(). Той използва метода pcap_findalldevs на pcap. На този метод се подава указател към структура от тип pcap_if_t, която съдържа информация за интерфейса: име, адрес и др. Този указател сочи към началото на списък от интерфейси.

Списъкът се итерира и имената на инерфейсите се записват в частната променлива на класа m_devices.

```
void NetworkInterface::findDevices(){
  pcap_if_t* devList = NULL;
  char errbuf[PCAP_ERRBUF_SIZE];

  memset(errbuf,0,PCAP_ERRBUF_SIZE);

  if(pcap_findalldevs(&devList,errbuf) < 0){
    throw new NetworkInterfaceException();
  }

  if(!devList->name)
    printf("");
  do{
    string name(devList->name);
    m_devices.push_back(name);

    devList = devList->next;

  }while(devList);

}
```

Фиг 3.1 Метод за намиране на всички имена на интерфейси на компютъра

Ако искаме да вземем имената на всички мрежови адаптери използваме метода GetList().

```
list<string>& NetworkInterface::GetList(){
  return this->m_devices;
}
```

Фиг 3.2 Метод за достъп до всички имена на интерфейси

2.Имплементацията на структури и класове отговорни за анализирането на хванатите пакети.

Класовете EthernetFrame, WLANFrame, IPPacket и TCPSegment отговарят за анализирането на информацията на получения пакет. Те наследяват абстрактния

клас AbstractHeader, а EtherFrame и WLANFrame наследяват и класа LinkLayerData. Всеки един от тях при извикване на конструктор, получава указател до Protocol Data Unit (PDU) на протокола, който разглеждат.

Всички тези класове се използват в класа Packet. Там се анализира информацията на целия пакет с тяхна помощ.

2.1. Помощни функции и макроси

Във файла HelperFunctions.h, HelperFunctions.cpp и HeaderTypes.h са дефинирани функции и макроси, които се използват за анализиране на информацията от пакети.Макросите дефинират максималната големина на получения пакет, колко големи да са хедърите, големината на MAC адресите, какви маски да се прилагат и какви са стойностите за сравняване.

```
#define PACKET_BUFFSIZE 2446

#define ETHER_IF 1
#define WLAN_IF 2

#define ETHER_ADDR_LEN 6
#define ETHER_HEADR_LEN 14
#define ETHER_TYPE_IP 0x08

#define WLAN_HEADR_LEN 32
#define WLAN_HEADR_TYPE_MASK 0x3000
#define WLAN_FRAME_TYPE_MANAGEMENT 0x0000
#define WLAN_FRAME_TYPE_CONTROLL 0x1000
#define WLAN_FRAME_TYPE_DATA 0x2000

#define IP_HEADER_LENGTH 20
#define IP_ADDR_LEN 4

#define TCP_HEADER_LENGTH 20
#define IP_PROTO_TCP 0x06

#define IP_RF 0x8000
#define IP_DF 0x4000
#define IP_MF 0x2000
#define IP_OFFMASK 0x1fff
```

Фиг. 3.3 Макроси използвани за определяне на вида на пакета

Методът MACtoCString превръща един MAC адрес от масив от байтове в лесен за четене стринг. Използва метода BinToHexd да превърне един байт от двоичен код

в шестнадесетичен.

```
//this method converts an array of bytes into a cstring
char* MACtoCString(const u_int8_t *macAddress){
  char *address = new char[12];
  int k = 0;
  u_int8_t value;
  //here I'm converting my address to a cstring
  for(int i = 0; i < 6; i++){
     value =(macAddress[i] >> 4);
     address[k] = BinToHexd(value);
     k++;
     value = (macAddress[i]  & 15);
     address[k] = BinToHexd(value);
     k++;
  }
  address[12] = 0;
  return address;
}

//this method returns the equivalent hexadecimal digit of the given value
char BinToHexd(u_int8_t value){
  if(value < 10){
     return (char)(value + 48);
  } else if(value < 16){
     return (char)(value + 55);
  }

  return 0;
}
```

Фиг 3.4 Методи за превръщането на MAC адрес от масив от байтове в стринг

MACtoCString създава cstring и итерира масива. Първо използва побитово шифтване с 4 позиции, за да вземе първата половина на байта и го дава като аргумент на BinToHexd, който го връща в 16-чен код. След това се прави побитово умножение, за да се вземе втората половина на байта и пак се подава на BinToHexd.

BinToHexd връща променлива от тип char и приема като аргумент променлива от тип u_int8_t. Към аргумента се добавя 48, ако той е по-малко от 10, за да се получи цифрата, която е неговата стойност. Ако е по-малко 16, но по-голямо от 10, то към него се добавя 55, за да се получи една от буквите от A до F.

2.2. EthernetFrame.

Конструктура на EthernetFrame получава указател към началото на кадъра. Първо се извиква, конструктора на LinkLayerData, който на свой ред вика конструктора на AbstractHeader и инициализира m_packet указателя. След това вика memset функцията, за да нулира структурата и тогава копира части от пакета, кореспондиращи на полетата на Ethernet кадър във полетата на структурата ether_info.

```
EthernetFrame::EthernetFrame(const u_char *packet)
:LinkLayerData(packet)
{
  memset(&ethernetHeader,0,sizeof(ethernet_info));
  memcpy(&ethernetHeader.ether_dest_addr,packet,6);
  memcpy(&ethernetHeader.ether_source_addr,(packet+6),6);
  memcpy(&ethernetHeader.ether_type,(packet+12),2);
}

EthernetFrame::EthernetFrame(const EthernetFrame &other)
:LinkLayerData(other.m_packet)
{
  memset(&ethernetHeader,0,sizeof(ethernet_info));
  memcpy(&ethernetHeader,&(other.ethernetHeader),sizeof(ethernet_info));
}
```

Фиг. 3.5 Нормален и копиращ конструктор на EthernetFrame

Копиращият конструктор първо вика конструктора на LinkLayerData и му подава указателят към началото на другия кадър. След това се нулира структурата ether_info с memset и се копира съдържанието на структурата на другия EthernetFrame.

Методите за получаване на MAC адресите на Ethernet кадъра използват помощната функция дефинирани във HelperFunctions.h файла. Те връщат в cstring MAC адресите на пакета.

```
char* EthernetFrame::getSourceMACAddress() const {
  return MACtoCString(ethernetHeader.ether_source_addr);
}
char* EthernetFrame::getDestinationMACAddress() const {
  return MACtoCString(ethernetHeader.ether_dest_addr);
}
u_short EthernetFrame::getFrameType(){
  return ethernetHeader.ether_type;
}
```

Фиг 3.6 Методи, връщащи информация за Ethernet кадъра

getFrameType връща стойността на frameType полето на структурата, с което се определя типа на Layer 3 протокола, който ще се разглежда.

Метода NetworkLayerPacket е виртуален метод, наследен от LinkLayerData и връща указател към началото на PDU-то на Layer 3 протокол. Този указател се получава, след като към него се добави offset равен на големината на хедъра.

```
const u_char* EthernetFrame::NetworkLayerPacket(){
  return &m_packet[ETHER_HEADR_LEN];
}
```

Фиг 3.7 Метод връщащ указател към PDU на Layer 3

2.3. WLANFrame

Конструктора на WLANFrame като този на EthernetFrame получава като аргумент указател, сочещ към началото на кадъра в паметта. Той извиква първо конструктора на LinkLayerData. След това, първо нулира структурата wlan_info и копира в тази структура 30 байта от началото на кадъра. Полетата са подредени в паметта на структурата по същия начин, както в WLAN хедър. Това позволява да се копира цялата информация направо в структурата. Копиращият конструктор работи аналогично на EthernetFrame конструктора

```
WLANFrame::WLANFrame(const u_char *packet)
:LinkLayerData(packet)
{
  memset(&wlanHeadr,0,WLAN_HEADR_LEN);
  memcpy(&wlanHeadr,packet,WLAN_HEADR_LEN);
}

WLANFrame::WLANFrame(const WLANFrame &other)
:LinkLayerData(other.m_packet)
{
  memset(&wlanHeadr,0,WLAN_HEADR_LEN);
  memcpy(&wlanHeadr,&(other.wlanHeadr),sizeof(wlan_info));
}
```

Фиг. 3.8 Нормален и копиращ конструкто на WLANFrame

Методите за връщане на адресите getSourceMACAddress и getDestinationMACAddress разглеждат стойността на DS полето, за да определят кое от 4-те полета е за източник и кое е за получател. След това се извиква MACtoCString метода, за да се копира масива от байтове в cstring.

```
char* WLANFrame::getSourceMACAddress() const {
```

```
  u_int16_t typeOfDS = (wlanHeadr.frame_controll & 0x00C0);
  switch(typeOfDS){
     case 0: return MACtoCString(wlanHeadr.address2);
     case 0x0040: return MACtoCString(wlanHeadr.address3);
     case 0x0080: return MACtoCString(wlanHeadr.address2);
     case 0x00C0: return MACtoCString(wlanHeadr.address4);
  }
  return 0;
}

char* WLANFrame::getDestinationMACAddress() const {
  u_int16_t typeOfDS = (wlanHeadr.frame_controll & 0x00C0);
  switch(typeOfDS){
     case 0: return MACtoCString(wlanHeadr.address1);
     case 0x0040: return MACtoCString(wlanHeadr.address1);
     case 0x0080: return MACtoCString(wlanHeadr.address3);
     case 0x00C0: return MACtoCString(wlanHeadr.address3);
  }
  return 0;
}
```

Фиг. 3.9 Методи за намиране на MAC адресите

Методите isDataFrame, isControlFrame и isManagementFrame проверяват какъв е типа на кадъра: дали съдържа данни или управлява връзката с точката за достъп. Прилага се маска на frame_controll полето на wlan_info и след това се определя по стойността, получена от побитовото умножени какъв точно е кадърът.

```
bool WLANFrame::isDataFrame(){
  u_int16_t var = wlanHeadr.frame_controll;
  u_int16_t type = (wlanHeadr.frame_controll & WLAN_HEADR_TYPE_MASK);

  if(type == WLAN_FRAME_TYPE_DATA)
     return true;

  return false;
}

bool WLANFrame::isControllFrame(){
  u_int16_t type = (wlanHeadr.frame_controll & WLAN_HEADR_TYPE_MASK);

  if(type == WLAN_FRAME_TYPE_CONTROLL)
     return true;
  return false;
}

bool WLANFrame::isManagemetFrame(){
    u_int16_t type = (wlanHeadr.frame_controll & WLAN_HEADR_TYPE_MASK);
```

```
  if(type == WLAN_FRAME_TYPE_MANAGEMENT)
     return true;
  return false;
}
```

Фиг. 3.10 Методи за определяна на типа на кадъра

Подобно на EthernetFrame метода NetworkLayerPacket връща указател към началото на IP пакета. То се получава като се добави големината на хедъра като offset към указателя на WLANFrame.

```
const u_char* WLANFrame::NetworkLayerPacket(){
  return &m_packet[WLAN_HEADR_LEN];
}
```

Фиг. 3.11 Метод за получаване на указател към началот на IP адреса

2.4. IPPacket

Обектите от този клас се създават аналогично на предишните два класа: вика се конструктора на суперкласа, нулира се структурата и се копират 20 байта от началото на пакета в структурата ip_info. Копиращият конструктор е също еднакъв по действие като другите.

```
IPPacket::IPPacket(const u_char* packet)
:AbstractHeader(packet)
{

  memset(&m_ipHeader,0,IP_HEADER_LENGTH);
  memcpy(&m_ipHeader,packet,IP_HEADER_LENGTH);
}

IPPacket::IPPacket(const IPPacket &other)
:AbstractHeader(other.m_packet)
{
memset(&m_ipHeader,0,IP_HEADER_LENGTH);
    memcpy(&m_ipHeader,&(other.m_ipHeader),IP_HEADER_LENGTH);
}
```

Фиг. 3.12 Нормален и копиращ конструктор на IPPacket

Методите за получаване на IP адресите getSourceIPAddress и getDestinationIPAddress използват функцията inet_ntoa, за да превърнат информацията в in_addr_t в cstring, който метода връща.

```
char* IPPacket::getSourceIPAddress(){
  return inet_ntoa(m_ipHeader.ip_src);
}

char* IPPacket::getDestinationIPAddress(){
  return inet_ntoa(m_ipHeader.ip_dst);
}
```

Фиг.3.13 Методи връщащи IP адресите

Някои от методите на класа връщат стойностите на полетата на IP хедъра като ttl, id, checksum, protocol, tos и др.

```
u_int8_t IPPacket::getTTL(){
  return m_ipHeader.ip_ttl;
}

u_int8_t IPPacket::getProtocolType(){
  return m_ipHeader.ip_p;
}

u_int16_t IPPacket::getPacketLength(){
  return m_ipHeader.ip_len;
}

u_int8_t IPPacket::getTypeOfService(){
  return m_ipHeader.ip_tos;
}

u_int16_t IPPacket::getID(){
  return m_ipHeader.ip_id;
}

u_int16_t IPPacket::getOffset(){
  return (m_ipHeader.ip_off & IP_OFFMASK);
}

bool IPPacket::isDFSet(){
  return (m_ipHeader.ip_off & IP_DF);
}
bool IPPacket::isMFSet(){
  return (m_ipHeader.ip_off & IP_MF);
}
```

Фиг. 3.14 Методи връщащи стойностите на полетата на IP пакета

IPPacket има методи, които проверяват опцията за фрагментация. Тези методи проверяват дали може да се фрагментира и дали има фрагментация или не може този пакет да се фрагментира.

```
//checks if the Don't Fragment flag is set
bool IPPacket::isDFSet(){
  return (m_ipHeader.ip_off & IP_DF);
}
bool IPPacket::isMFSet(){
  return (m_ipHeader.ip_off & IP_MF);
}
```

Фиг 3.15 Методи проверяващи за дефрагментация на пакета

Както останалите класове и този има метод, който връща указател към PDU-то на протокол от горния слой. Към указателя за началото на пакета се добавя offset с

големината на хедъра - 20B. Тогава се получава указател към сегмент на Транспортния слой.

```
const u_char* IPPacket::TransportLayerSegment(){
return &m_packet[IP_HEADER_LENGTH];
}
```

Фиг. 3.16 Метод за връщане на указател към сегмент от транспортния слой

2.5.TCPSegment

Този клас работи почти по същият начин като останалите класове. Конструкторът получава указател към началото на tcp сегмента. Извиква се конструкторът на AbstractHeader, нулира се структурата и се копират 20B(големината на tcp хедъра) в структурата tcp_info.

```
TCPSegment::TCPSegment(const u_char *packet)
:AbstractHeader(packet)
{
  memset(&m_tcpHeader,0,TCP_HEADER_LENGTH);
  memcpy(&m_tcpHeader,packet,TCP_HEADER_LENGTH);
}

TCPSegment::TCPSegment(const TCPSegment &other)
:AbstractHeader(other.m_packet)
{
  memset(&m_tcpHeader,0,TCP_HEADER_LENGTH);
  memcpy(&m_tcpHeader,&other.m_tcpHeader,TCP_HEADER_LENGTH);
}
```

Фиг. 3.17 Нормален и копиращ конструктор на TCPSegment

Класа има методи за получаване на стойностите на полетата на tcp_info структурата.

```
u_int16_t TCPSegment::getDestPort(){
  return m_tcpHeader.dest_port;
}

u_int16_t TCPSegment::getSourcePort(){
  return m_tcpHeader.source_port;
}

u_int32_t TCPSegment::getAckNum(){
  return m_tcpHeader.ack_num;
}

u_int32_t TCPSegment::getSeqNum(){
  return m_tcpHeader.seq_num;
}
```

```
u_int16_t TCPSegment::getWindowSize(){
  return m_tcpHeader.window_size;
}

u_int16_t TCPSegment::getUrgentPointer(){
  return m_tcpHeader.urgent_pointer;
}

u_int8_t TCPSegment::getOffset(){
  return TCP_OFFS(m_tcpHeader.offs_res);
}

u_int8_t TCPSegment::getRes(){
  return (m_tcpHeader.offs_res & TCP_RES_MASK);
}

u_int16_t TCPSegment::getChecksum(){
  return m_tcpHeader.checksum;
}
```

Фиг. 3.18 Методи за достъп до полетата на tcp хедъра

2.6. HTTPPacket

Класа HTTPPacket разглежда информацията, която се съдържа в хедъра на HTTP пакет. Тъй като цялата информация е представена в четим за хората формат се използва string за съдържане на хедъра. Когато имаме нужда от него просто викаме метода .

```
HTTPPacket::HTTPPacket(char *httpPacket){
  QString header(httpPacket);

  int doubleNewline = 0;
  for(QString::iterator it = header.begin(); it!= header.end();++it){

     m_headerHttp += *it;

     if(*it == '\n'){
        doubleNewline++;
     }
     if(doubleNewline == 2){
        break;
     }
  }
}

HTTPPacket::HTTPPacket(const HTTPPacket &other){
  m_headerHttp = other.m_headerHttp;
}
```

```
QString HTTPPacket::httpHeaderInfo() const{
  return m_headerHttp;
}
```

Фиг. 3.19 Конструктори и метод за достъп до хедъра

2.7.Packet

Класа Packet обединява всички класове за анализиране на хедъри в себе си. Конструкторът приема като аргументи указател, който сочи към началото на хванатия пакет в паметта и целочислено число, което обозначава дали пакета е Ethernet или wlan кадър. Първо се създават обектите за каналния слой, а след това се проверява дали съдържат IP пакети. Ако има се създава IPPacket обект. След създаването на IPPacket се проверява дали съдържа TCP сегмент. Ако е така се създава TCPSegment обект.

След създаването на различните обекти за пакети се нулират указателите, към пакетите им. Това се прави, за да се предотврати segmentation fault при извикването на деструктурите им, защото pcap ги държи само докато завърши callback метода, а след това ги изтрива от паметта.

```
Packet::Packet(const u_char *packet,pcap_pkthdr *pkthdr, u_int8_t ifType)
  :m_etherHeader(NULL),m_ipHeader(NULL),
   m_wlanHeader(NULL),m_tcpHeader(NULL),
   m_pkthdr(pkthdr)
{
  u_int16_t frameType;
  LinkLayerData *link = NULL;

  if(ifType == ETHER_IF){
     m_etherHeader = new EthernetFrame(packet);
     frameType = m_etherHeader->getFrameType();
     link = m_etherHeader;
  } else if(ifType == WLAN_IF){
     m_wlanHeader = new WLANFrame(packet);
     frameType = m_wlanHeader->isDataFrame() ? ETHER_TYPE_IP : 0;
     link = m_wlanHeader;
  }


  if(m_etherHeader->getFrameType() == ETHER_TYPE_IP){
     const u_char *ipPacket = link->NetworkLayerPacket();
     m_ipHeader = new IPPacket(ipPacket);
     if(m_ipHeader->getProtocolType() == IP_PROTO_TCP){
```

```
            const u_char *tcpSegm = m_ipHeader->TransportLayerSegment();
        }
    }
    SetPtrsToNull();
}
```

Фиг. 3.20 Конструктор на Packet

За да се може да се присвоява един Packet от друг, се дефинира опреатор за присвояване.

```
Packet& Packet::operator=(const Packet& other){

    if(other.m_etherHeader){
        if(m_etherHeader){
            memcpy(m_etherHeader,other.m_etherHeader,sizeof(EthernetFrame));
        } else {
            m_etherHeader = new EthernetFrame(*other.m_etherHeader);
        }
    } else {
        if(m_wlanHeader){
            memcpy(m_wlanHeader,other.m_wlanHeader,sizeof(WLANFrame));
        } else {
            m_wlanHeader = new WLANFrame(*other.m_wlanHeader);
        }
    }

    if(other.m_ipHeader){
        if(m_ipHeader){
            memcpy(m_ipHeader, other.m_ipHeader,sizeof(Packet));
        } else {
            m_ipHeader = new IPPacket(*other.m_ipHeader);
        }
    } else {
        if(m_ipHeader){
            delete m_ipHeader;
            m_ipHeader = NULL;
        }
    }


    if(other.m_tcpHeader){
        if(m_tcpHeader){
            memcpy(m_tcpHeader,other.m_tcpHeader,sizeof(TCPSegment));
        } else {
            m_tcpHeader = new TCPSegment(*other.m_tcpHeader);
        }
    } else {
        if(m_tcpHeader){
          delete m_tcpHeader;
          m_tcpHeader = NULL;
        }
```

```
    }

    if(other.m_httpHeader){
        if(m_httpHeader){
            memcpy(m_httpHeader,other.m_httpHeader,sizeof(HTTPPacket));
        } else {
            m_httpHeader = new HTTPPacket(*other.m_httpHeader);
        }
    } else {
        if(m_httpHeader){
            delete m_httpHeader;
            m_httpHeader = NULL;
        }
    }
}
```

Фиг.3.21 Оператор за присвояване

Метода AbridgedHeaderInfo връща string със съкратена информация за пакета. Този string съдържа MAC и IP адресите заедно с времето, когато е получено.

```
string Packet::AbridgedHeadersInfo() const{
    string temps;
    time_t time = m_pkthdr->ts.tv_sec;
    char tmbuf[64];

    strftime(tmbuf,64,"%Y-%m-%D %H:%M:%S ",localtime(&time));
    temps += tmbuf;
    temps += "Destination MAC: ";

    if(m_etherHeader)
        temps += m_etherHeader->getDestinationMACAddress();
    else temps += m_wlanHeader->getDestinationMACAddress();

    temps += " Source MAC: ";

    if(m_etherHeader){
        temps += m_etherHeader->getSourceMACAddress();
    } else {
        temps += m_wlanHeader->getSourceMACAddress();
    }

    temps += " Dest IP ";
    temps += m_ipHeader->getDestinationIPAddress();
    temps += " Source IP ";
    temps += m_ipHeader->getSourceIPAddress();

    return temps;
}
```

Фиг. 3.22 Метод за получаване на съкратена информация за пакета

Метода HeadersInfo връща string с цялата информация в хедърите пакета: Ethernet или WLAN, IP и TCP.

```
QString Packet::HeadersInfo() const{
  QString headersString;

  if(m_etherHeader){
     headersString.append(ethernetHeaderInfo().c_str());
     headersString.append("\n------------------\n");
  }

  if(m_ipHeader){
     headersString.append(ipHeaderInfo().c_str());
     headersString.append("\n------------------\n");
  }

  if(m_wlanHeader){
     headersString.append(wlanHeaderInfo().c_str());
     headersString.append("\n------------------\n");
  }

  if(m_tcpHeader){
     headersString.append(tcpHeaderInfo().c_str());
     headersString.append("\n------------------\n");
  }

  if(m_httpHeader){
     headersString += m_httpHeader->httpHeaderInfo();
  }

  return headersString;
}
```

Фиг. 3.23 Метод за получаване на пълната информация за пакета

Методите wlanHeaderInfo, ethernetHeaderInfo, ipHeaderInfo, tcpHeaderInfo връщат стринг, съдържащ информацията за определен header. Те се използват от HeadersInfo, за да се сглоби string с цялата информация за пакет.

```
string Packet::ethernetHeaderInfo() const {
  char *destMac = m_etherHeader->getDestinationMACAddress();

  char *sourceMac = m_etherHeader->getSourceMACAddress();

  u_short frameType = m_etherHeader->getFrameType();


  stringstream sEtherStream;
```

```
  sEtherStream << "Ethernet Header --- Source address: "
               << sourceMac << endl <<    "Destination address: "
               << destMac << endl << " Frametype: " << frameType << endl;


  delete [] sourceMac;
  delete [] destMac;
  return sEtherStream.str();
}

string Packet::wlanHeaderInfo() const{
  stringstream sWlanInfo;
  char *sourceMac = m_wlanHeader->getSourceMACAddress();
  char *dstMac = m_wlanHeader->getDestinationMACAddress();

  sWlanInfo << "Source MAC: " << sourceMac << endl
 << "Destination MAC: " << dstMac << endl
<< "Is Data Frame: "<<(m_wlanHeader->isDataFrame() ? "Yes":"No") << endl <<
"Is Control Frame: "<< (m_wlanHeader->isControlFrame() ? "Yes" : "No") << endl
<< "Is Management Frame: "<< (m_wlanHeader->isManagementFrame() ? "Yes" :
"No") << endl;

  return sWlanInfo.str();
}


string Packet::tcpHeaderInfo() const{
  stringstream sTcpInfo;

  sTcpInfo << "Source port: " << m_tcpHeader->getSourcePort() << endl
        << "Destination port: " << m_tcpHeader->getDestPort() << endl
        << "Sequence number: " << m_tcpHeader->getSeqNum() << endl
        << "Acknoledgment number: " << m_tcpHeader->getAckNum() << endl
        << "Offset: " << m_tcpHeader->getOffset() << endl

        << "TCP flags: " << m_tcpHeader->cwrFlag() << "." << m_tcpHeader-
>ecnFlag()
        << "." << m_tcpHeader->urgFlag() << "." << m_tcpHeader->ackFlag() << "."
        << m_tcpHeader->pshFlag() << "." << m_tcpHeader->rstFlag() << "."
        << m_tcpHeader->synFlag() << "." << m_tcpHeader->finFlag() << endl

        << "Window: " << m_tcpHeader->getWindowSize() << endl
        << "Checksum: " << m_tcpHeader->getChecksum() << endl
        << "Urgent Pointer: " << m_tcpHeader->getUrgentPointer() << endl;

  return sTcpInfo.str();
}
```

Фиг. 3.24 Методи връщащи информация за хедърите на пакета

2.8. Хващане на пакети

Хващането на пакетите се извършва от pcap. Използва се метода pcap_loop, на който се подава callback функцията PacketHandler. Но pcap_loop се пуска в друга нишка и затова се дефинира метода loopThread и макроса START_LOOP. В START_LOOP глобалната променлива, която е указател към обект от клас ISSFApplication qtApp, който се използва в PacketHandler, и се вика pcap_loop, за да се пусне хващането.

В PacketHandler се подава указателя на пакета като аргуемент addPacket метода на SharedMemoryController. Този клас управлява общата памет. Обектът добавя пакета към контейнър, специално направен за обекти от класа Packet. След това се вика методът emitPacketsAdded чрез qtApp. Този метод изпраща сигнал към потребителския интерфейс, за да съобщи, че е приет пакет.

```
ISSFApplication  *qtApp = NULL;

#define START_LOOP(_shMem,_pcap){pDev=_pcap; sharedMemoryController = _shMem; pcap_loop(_pcap,-1,(pcap_handler)PacketHandler,NULL);}


void PacketHandler(u_char *args, const struct pcap_pkthdr *pkthdr, const u_char *packet){
  sharedMemoryController->addPacket(packet,pkthdr,ETHER_IF);
  qtApp->emitPacketsAdded();
  boost::this_thread::interruption_point();

}

void loopThread(pcap_t *dev,SharedMemoryController *shMem){
  START_LOOP(shMem,dev);
}
```

Фиг. 3.2 Методи използвани в хващането на пакетите

3. Управление на общата памет

За да се управлява паметта използвана от двете нишки, са имплементирани класовете SharedMemoryController и PktVector.

3.1. SharedMemoryController

Този клас действа като wrapper на PktVector. Предоставя възможност за добавяне на пакети, достъп до тях и изтриването им от паметта. Използва се от нишката за хващане на пакети за добавяне на хванатите пакети и от потребителския

интерфейс за достъп до тях.

Когато се добавят пакети с addPacket се използва mutex, за да не се позволи достъп до пакетите при добавяне в m_vector. Потребителският интерфейс използва препратки за достъп до пакетите. При промяна в размера на контейнера се променят адресите на разглеждания пакетет. За това се спира достъпа на други нишки до него при добавяне на пакети, за да не се получи segmentation fault.

```
void SharedMemoryController::addPacket(const u_char *text, const pcap_pkthdr *pkthdr, u_int8_t ifType){
  boost::mutex mtx;
  boost::mutex::scoped_lock lock(mtx);
  m_counter++;
  m_vector.push_back(text,pkthdr,ifType);
}
```

Фиг. 3.26 Метод за добавяне на пакети в общата памет

Потребителският интерфейс, когато иска да запише последния получен пакет в полето за пакети се нуждае само от последния елемент във вектора. За това се използва методът getLatestPacket.

```
const Packet& SharedMemoryController::getLatestPacket(){
  if(m_counter<0)
     throw new SharedMemoryException();
  return m_vector.back();
}
```

Фиг. 3.27 Метод за получаване на последния получен пакет

За получаване на достъп до определен пакет и за намиранае на големината на вектора се използват методите getPacket и getVectorSize.

```
const Packet& SharedMemoryController::getPacket(int packetNum) const{
  return m_vector[packetNum];
}

int SharedMemoryController::getVectorSize(){
  return m_vector.size();
}
```

Фиг. 3.28 Методи за достъп до произволен пакет в контейнера и за получаване на големината му

3.2. PktVector

PktVector е специално създаден контейнер, който се използва за съхранение на получените пакети. Причината за създаването на този клас е, че е по-удобно да се работи със специален клас, чиято функционалност се ограничава само в съхранение на информацията за мрежови пакети.

PktVector има само два вида конструктори: по подразбиране и този, с който се определя първоначалния капацитет на контейнера. Деструкторът изтрива всички записани пакети.

```
PktVector::PktVector()
  : m_size(0), m_capacity(10),m_collection(new Packet[10])
{
  memset(m_collection,0,sizeof(Packet)*m_capacity);
}

PktVector::PktVector(u_int32_t capacity)
:m_size(0),m_capacity(capacity),m_collection(new Packet[m_capacity])
{
  memset(m_collection,0,sizeof(Packet)*m_capacity);
}

PktVector::~PktVector(){
  delete [] m_collection;
}
```

Фиг. 3.29 Конструктори и деструктори на PktVector

За записване във вектора се използва методът push_back. То създава Packet от подадения като аргумент указател, записва го като последния елемент в контейнера и се инкрементира m_size, която брои елементите в него. Ако m_size стане равно или по-голямо от капацитета на вектора се вика метода resize. Този метод разширява контейнера, като динамично заделя памет с по-голям капацитет и копира пакетите в нея и изтрива старата памет.

```
void PktVector::resize(){
  m_capacity *= 2;
  Packet *temp = m_collection;
  m_collection = new Packet[m_capacity];
  memset(m_collection,0,sizeof(Packet)*m_capacity);
  for(int i = 0; i < m_size; i++){
     m_collection[i] = temp[i];
  }
  delete [] temp;
}
```

```
void PktVector::push_back(const u_char *packet, const pcap_pkthdr *pkthdr,
u_int8_t ifType){
  if(m_size >= m_capacity)
     resize();
  m_collection[m_size] = Packet(packet,pkthdr,ifType);
  m_collection[m_size].SetPtrsToNull();
  m_size++;
}
```

Фиг. 3.30 Методи за добавяне в контейнера

Метода clear изчиства паметта от всички записани пакети.

```
void PktVector::clear(){
  delete [] m_collection;
  m_collection = new Packet[m_capacity];
  memset(m_collection,0,sizeof(Packet)*m_capacity);
  m_size = 0;
}
```

Фиг. 3.31 Метод за изчистване на контейнера

За достъп до елементите на вектора или намиране на големината на вектора се използват метода size, front, back и оператора за достъп до елемент. Методите за достъп връщат константна препратка, за да не се позволи промяна на записаните пакети.

```
int PktVector::size(){
  return m_size;
}

const Packet& PktVector::front(){
  return m_collection[0];
}

const Packet& PktVector::back(){
  if(!m_size)
     return m_collection[0];

  return m_collection[m_size-1];
}

const Packet& operator[](int i) const {
        if(i < 0 || i >= m_size)
           throw new PktVectorException();
        return m_collection[i];
}
```

Фиг. 3.32 Методи за получаване на достъп на контейнера

4.Имплементиране на филтрацията

Филтрацията се постига, като се използва вградената в pcap възможност за дефиниране на филтри. Метода на класа Filter compileFilter разглежда получения от потребителкси интерфейс string и съставя нов, който подава на метода pcap_compile и връща създадената филтър програма. За да се определи точно какъв филтър да се направи, се вика частния метод makeFilter. Той определя за какъв протокол трябва да се направи филтър и го подава на съответните методи, които го създават.

```
void Filter::makeFilter(vector<string>& protocol){
  string proto = protocol.front();
  vector<string> stringVector;
  for(int i = 1; i < protocol.size();i++){
     stringVector.push_back(protocol[i]);
  }
  if(ETHER_PROTO(proto.c_str())){
    m_filter = FilterEther::getEtherFilter(stringVector);
  } else if(IP_PROTO(proto.c_str())){
     m_filter = FilterIPv4::ipFilter(stringVector);
  } else if(TCP_PROTO(proto.c_str())){
     m_filter = FilterTCP::tcpFilter(stringVector);
  }
}

bpf_program* Filter::compileFilter(pcap_t *dev, string& filter){

  vector<string> filterExpression;
  string temp;

  for(string::iterator it = filter.begin();
          it != filter.end();++it){
    if((*it) != ' ' && it != filter.end()){
       temp += *it;
    } else {
       filterExpression.push_back(temp);
       temp.clear();
    }
  }
  makeFilter(filterExpression);

  bpf_program *filterProgram = new bpf_program;

  if(pcap_compile(dev,filterProgram,m_filter.c_str(),0,
          PCAP_NETMASK_UNKNOWN) < 0){
```

```
        pcap_perror(dev,"Compile");
    }

    return filterProgram;

}
```

Фиг. 3.33 Метод за създаване на филтър

4.1. FilterEther

Този клас съдържа само статични методи, които се използват за създването на филтър за Ethernet протокола.

```
string FilterEther::getEtherFilter(vector<string>& etherFilterStringVector){
    string etherFilter("ether ");

    string subfield = etherFilterStringVector.front();
    string operationType = etherFilterStringVector[1];
    string value = etherFilterStringVector[2];

    if(MAC_ADDRESS_DST(subfield.c_str())){
        etherFilter += dstMacAddressString(operationType,value);
    } else if(MAC_ADDRESS_SRC(subfield.c_str())){
        etherFilter += srcMacAddressString(operationType,value);
    } else if(ETHER_TYPE(subfield.c_str())){
        etherFilter += etherType(operationType,value);
    }

    return etherFilter;
}

string FilterEther::dstMacAddressString(string& operationType, string& value){
    string stringToReturn;

    if(OPERATION_EQUAL(operationType.c_str())){
        stringToReturn += string("dst ") + value;
    } else if(OPERATION_NOT_EQUAL(operationType.c_str())){
        stringToReturn += string("not dst ") + value;
    }

    return stringToReturn;
}
string FilterEther::srcMacAddressString(string& operationType, string& value){
    string stringToReturn;

    if(OPERATION_EQUAL(operationType.c_str())){
        stringToReturn += string("src ") + value;
    } else if(OPERATION_NOT_EQUAL(operationType.c_str())){
        stringToReturn += string("not src ") + value;
```

```
  }

  return stringToReturn;
}
string FilterEther::etherType(string& operationType, string& value){
  string stringToReturn;

  if(OPERATION_EQUAL(operationType.c_str())){
     stringToReturn += string("proto ") + value;
  } else if(OPERATION_NOT_EQUAL(operationType.c_str())){
     stringToReturn += string("not proto ") + value;
  }
  return stringToReturn;
}
```

Фиг. 3.34 Метод за съставяне на стринг за Ethernet филтър

4.2. FilterIP

Класа FilterIP създава филтър за IP протокола. Като изключим филтрирането на адресите, за полетата, се създават аритметични изрази. Ако условията на тези изрази са изпълнени, то пакета се приема.

```
string FilterIPv4::ipFilter(vector<string>& ipFilterStringVector){
  string ipFilter("ip");

  string& field = ipFilterStringVector.front();
  string& operation = ipFilterStringVector[1];
  string& value = ipFilterStringVector[2];

  if(IP_SRC_ADDRESS(field.c_str())){
     ipFilter+=srcIpAddrFilter(operation,value);
  } else if(IP_DST_ADDRESS(field.c_str())){
     ipFilter+=dstIpAddrFilter(operation,value);
  } else if(IP_VHL(field.c_str())){
     ipFilter+=vhlFilter(operation,value);
  } else if(IP_LENGTH(field.c_str())){
     ipFilter+=lengthFilter(operation,value);
  } else if(IP_TTL(field.c_str())){
     ipFilter+=ttlFilter(operation,value);
  } else if(IP_TOS(field.c_str())){
     ipFilter+=tosFilter(operation,value);
  } else if(IP_OFFS(field.c_str())){
     ipFilter+=offsFilter(operation,value);
  } else if(IP_PROTOCOL(field.c_str())){
     ipFilter+=protoFilter(operation,value);
  } else if(IP_ID(field.c_str())){
     ipFilter+=idFilter(operation,value);
  }
```

```
  return ipFilter;
}


string FilterIPv4::srcIpAddrFilter(string& operation, string& value){
  string stringToReturn;

  if(OPERATION_EQUAL(operation.c_str())){
     stringToReturn += string(" src ") + value;
  } else if(OPERATION_NOT_EQUAL(operation.c_str())){
     stringToReturn += string("not src ") + value;
  }

  return stringToReturn;
}

string FilterIPv4::dstIpAddrFilter(string& operation, string& value){
  string stringToReturn;

  if(OPERATION_EQUAL(operation.c_str())){
     stringToReturn += string(" dst ") + value;
  } else if(OPERATION_NOT_EQUAL(operation.c_str())){
     stringToReturn += string(" not dst ") + value;
  }

  return stringToReturn;
}
string FilterIPv4::vhlFilter(string& operation, string& value){
  string stringToReturn("[0]");
  return stringToReturn + operation + value;
}

string FilterIPv4::lengthFilter(string& operation, string& value){
  string stringToReturn("[2:2]");

  return stringToReturn + operation + value;
}

string FilterIPv4::ttlFilter(string& operation, string& value){
  string stringToReturn("[8]");

  return stringToReturn + operation + value;
}

string FilterIPv4::tosFilter(string& operation, string& value){
  string stringToReturn("[1]");

  return stringToReturn + operation + value;
```

```
}

string FilterIPv4::offsFilter(string& operation, string& value){
  string stringToReturn("[6:2]");

  return stringToReturn + operation + value;
}

string FilterIPv4::protoFilter(string& operation, string& value){
  string stringToReturn("[5]");

  return stringToReturn + operation + value;
}

string FilterIPv4::idFilter(string &operation, string &value){
  string stringToReturn("[4:2]");

  return stringToReturn + operation + value;
}
```

Фиг. 3.35 Методи за съставяне на стринг за филтър за IP протокола

4.3. FilterTCP

Методите на този клас отговарят създаването на филтри за TCP протокола. Филтрите представляват аритметични изрази подобни на тези съставени за IP.

```
string FilterTCP::tcpFilter(vector<string>& filterStringVector){
  string tcpFilter("tcp ");
  string& field = filterStringVector[0];
  string& operation = filterStringVector[1];
  string& value = filterStringVector[2];

  if(TCP_DST_PORT(field.c_str())){
     tcpFilter += dstPortFilter(operation,value);
  } else if(TCP_SRC_PORT(field.c_str())){
     tcpFilter += srcPortFilter(operation,value);
  } else if(TCP_ACK_NUM(field.c_str())){
     tcpFilter += ackNumFilter(operation,value);
  } else if(TCP_SEQ_NUM(field.c_str())){
     tcpFilter += seqNumFilter(operation,value);
  } else if(TCP_WINDOW(field.c_str())){
     tcpFilter += windowFilter(operation,value);
  } else if(TCP_FLAGS(field.c_str())){
     tcpFilter += tcpFlagsFilter(operation,value);
  }

  return tcpFilter;
```

```
}

string FilterTCP::dstPortFilter(string &operation, string &value)
{
  string stringToReturn("tcp[2:2] ");
  return stringToReturn + operation + value;
}

string FilterTCP::srcPortFilter(string &operation, string &value)
{
  string stringToReturn("tcp[0:2]");

  return stringToReturn + operation + value;
}

string FilterTCP::seqNumFilter(string &operation, string &value)
{
  string stringToReturn("tcp[4:4]");


  return stringToReturn + operation + value ;
}

string FilterTCP::ackNumFilter(string &operation, string &value)
{
  string stringToReturn("tcp[8:4]");

  return stringToReturn + operation + value;
}

string FilterTCP::tcpFlagsFilter(string &operation, string &value){
  string stringToReturn("tcp[tcpflags]");


  return stringToReturn + operation + value;
}

string FilterTCP::windowFilter(string &operation, string &value){
  string stringToReturn("tcp[16:2]");


  return stringToReturn + operation + value;
}
```

Фиг. 3.36 Методи за съставяне на стринг за TCP филтър

5. Създаване на нишки

За да може да се работи с нишки, трябва да се използва подходящата библиотека.

Тука се използва boost::thread. Класът Thread е wrapper на boost::thread. Той наследява QObject и използва Q_OBJECT макрос, за да може moc компилаторът да го разглежда като обект от Qt.

Конструкторът на Thread приема като аргументи callback метод, който се пуска в нишката, указатели към SharedMemoryController и отворения интерфейс. Той свързва сигналите startThread и stopThread към слотовете onStartThread и onStopThread. Те отговарят за пускането и спирането на нишката.

```
Thread::Thread(void (*workerFunction)(pcap_t *dev,
SharedMemoryController *shMem),pcap_t *dev,SharedMemoryController *shMem)
  :m_dev(dev), m_sharedMemory(shMem), m_method(workerFunction),
m_workerThread(NULL)
{
  connect(this,SIGNAL(startThread()),this,SLOT(onStartThread()));
  connect(this,SIGNAL(stopThread()),this,SLOT(onStopThread()));
}
```

Фиг. 3.37 Конструктор на класа

Сигналите се пускат с методите start и stop.

```
void Thread::start(){
  m_started = true;
  emit this->startThread();
}

void Thread::stop(){
  m_started = false;
  emit this->stopThread();
}
```

Фиг. 3.38 Методи за сигнализиране за пускане и спиране на нишка

Променливата m_started се връща от метода isStarted, който се използва за да разберем дали нишката е пусната или не.

Слотовете onStartThread и onStopThread отговарят за създаването на нишка. onStartThread създава обект от клас boost::thread, който пуска нишката. onStopThread използва метода interrupt, за да я спре.

```
void Thread::onStartThread(){
  if(!m_workerThread)
     m_workerThread = new boost::thread(m_method,m_dev,m_sharedMemory);
}
```

```
void Thread::onStopThread(){
  if(m_workerThread)
     m_workerThread->interrupt();
  delete m_workerThread;
  m_workerThread = NULL;
}

bool Thread::isStarted(){
  return m_started;
}
```

Фиг. 3.39 Методи за пускане и спиране на нишка

6.Имплементация на графичен потребителски интерфейс

За имплементирането на потребителския интерфейс е използван Qt и QtCreator. Имплементирани са два класа ISSFApplication и MainWindow. ISSFApplication отговаря за цялата логика зад потребителския интерфейс. MainWindow дава достъп до всички елементи на интерфейса.

6.1.MainWindow

Този клас е създаден от QtCreator. Предоставя методи, които връщат указатели към елементите от интерфейса. Те се използват от ISSFApplication, за да се свържат с подходящата логика.

```
MainWindow::MainWindow(QWidget *parent) :
  QMainWindow(parent),
  ui(new Ui::MainWindow)
{
  ui->setupUi(this);
  buttons = new QVector<QPushButton*>();
  buttons->push_back(ui->playButton);
  buttons->push_back(ui->pauseButton);
  buttons->push_back(ui->stopButton);

  applyButton = ui->applyButton;
  filterField = ui->filterField;
  m_packetsCptrdField = ui->packetsCptrd;
  m_packetFullInfoField = ui->packetInfo;
  interfaceButton = ui->Interface;

  showFullInfoButton = ui->m_ShowFullInfoButton;
  continueButton = ui->m_continueButton;
}
```

```
QVector<QPushButton*>* getProcessButtons(){
    return buttons;
}

QPushButton* getApplyButton(){
    return applyButton;
}
QTextEdit* getFilterField(){
    return filterField;
}

QTextBrowser* getPacketsCptrdField(){
    return m_packetsCptrdField;
}

QTextBrowser* getPacketsFullInfoField(){
    return m_packetFullInfoField;
}

QPushButton* getInterfaceButton(){
    return interfaceButton;
}

QPushButton* getShowInfoButton(){
    return showFullInfoButton;
}

QPushButton* getContinueButton(){
    return continueButton;
}
```

Фиг. 3.40 Конструктор на класа и методи за достъп

6.2. ISSFApplication

В този клас е реализирана цялата логика на потребителския интерфейс. Той отговаря за пускане на хващането на пакетите, за прилагането на филтрите и за показване на цялата информация за пакета.

В конструктора се свързват всичките сигнали на бутони или на ISSFApplication, нужни за работата на потребителския интерфейс със слотовете, които изпълняват определена функционалност: стартиране на слушането за пакети, прилагане на филтри и др. Въвеждат се и размерите на главния прозорец.

```
ISSFApplication::ISSFApplication(int &argc, char **argv)
  :QApplication(argc,argv),m_allowedToAdd(true), m_AddedPackets(0),
```

```
    m_filter(new Filter())
{

  main = new MainWindow();
  main->setGeometry(200,200,500,600);
  main->show();


  //creating the filter segment
  QPushButton *apply = main->getApplyButton();
  QPushButton *interface = main->getInterfaceButton();
  m_PacketsCptrdField = main->getPacketsCptrdField();
  m_packetsFullInfo = main->getPacketsFullInfoField();
  m_sharedMemory = new SharedMemoryController();
  QPushButton *showFullInfo = main->getShowInfoButton();
  QPushButton *continueButton = main->getContinueButton();
  QVector<QPushButton*> *buttons = main->getProcessButtons();


  m_workerThread = new Thread(loopThread,m_dev,m_sharedMemory);
  connect(buttons->at(0),SIGNAL(clicked()),this,SLOT(startWorkerThread()));
  connect(buttons->at(2),SIGNAL(clicked()),this,SLOT(stopWorkerThread()));
  connect(this,SIGNAL(packetsAdded()),this,SLOT(onPacketsAdded()));
  connect(apply,SIGNAL(clicked()),this,SLOT(onFilterApplied()));
  connect(interface,SIGNAL(clicked()),this,SLOT(onInterfaceClicked()));
  connect(showFullInfo,SIGNAL(clicked()),this,SLOT(onPacketSelected()));
  connect(continueButton,SIGNAL(clicked()),this,SLOT(onContinue()));

  qtApp = this;
}
```

Фиг. 3.41 Конструктор на ISSFApplication

Методът run пуска приложението. Първо извиква метода findDevices на NetworkInterface, за да намери всички мрежови адаптери на компютъра. След това използва наследения метод exec, за да се пусне приложението.

```
void ISSFApplication::run(){
  networkIf.findDevices();
  this->exec();
}
```

Фиг. 3.42 Метод, от който се пуска приложението

Слота onInterfaceClicked е свързан с Interface бутона. Той създава нов прозорец, в който са показани имената на всички интерфейси. Интерфейс се избира, като се маркира името и се натисне бутона Ок. Той изпраща сигнал, свързан с onOk. В този слот се взима маркирания текст. Ако е пусната нишката за слушане, се спира работата и докато се отвори новия интерфейс. За излизане от прозореца без избиране на каквото и да е име се натиска Cancel.

```
void ISSFApplication::onInterfaceClicked(){
  QWidget *interfaceWindow = new QWidget();
  interfaceWindow->setGeometry(200,200,200,200);
  m_interfaces = new QTextBrowser(interfaceWindow);
  QPushButton *ok = new QPushButton(interfaceWindow);
  QPushButton *cancel = new QPushButton(interfaceWindow);

  ok->setText(*(new QString("Ok")));
  cancel->setText(*(new QString("Cancel")));

  QGridLayout *interfaceWindowLayout = new QGridLayout(interfaceWindow);
  QGridLayout *firstRowLayout = new QGridLayout(interfaceWindow);
  QGridLayout *secondRowLayout = new QGridLayout(interfaceWindow);

  interfaceWindowLayout->addLayout(firstRowLayout,1,1,1,1);
  interfaceWindowLayout->addLayout(secondRowLayout,2,1,1,1);

  firstRowLayout->addWidget(m_interfaces);
  secondRowLayout->addWidget(ok,1,1,1,1);
  secondRowLayout->addWidget(cancel,1,2,1,1);

  list<string>& interfaceList = networkIf.GetList();
  for(list<string>::iterator it = interfaceList.begin();
     it != interfaceList.end(); it++){
     QString pcapIf((*it).c_str());
     m_interfaces->append(pcapIf);
  }

  connect(ok,SIGNAL(clicked()),this,SLOT(onOk()));

  connect(cancel,SIGNAL(clicked()),interfaceWindow,SLOT(close()));
  interfaceWindow->show();
}
void ISSFApplication::onOk(){
  bool wasStarted = false;
  if(m_workerThread->isStarted()){
     m_workerThread->stop();
     delete m_workerThread;
     wasStarted=true;
  }

  QString intfName = m_interfaces->textCursor().selectedText();
  m_interfaceName = string(intfName.toUtf8().constData());

  m_dev = openPcapInterface(m_interfaceName.c_str());

  m_workerThread = new Thread(loopThread,m_dev,m_sharedMemory);
  ((QWidget*)m_interfaces->parent())->close();
  if(wasStarted){
     m_workerThread->start();
```

```
    }
}
```
Фиг. 3.43 Методи за избиране на интерфейс

За пускане, паузиране и спиране на нишки, се използват бутоните Start, Pause и Stop. Start изпраща сигнал към onStartThread, който пуска нишката. Pause изпраща сигнал до onPauseThread. Този слот спира нишката. Stop е свързан с onStopThread, който спира нишката и записва всички получени пакети в файла log.txt, намиращ се в директорията на приложението.

```
void ISSFApplication::onStartWorkerThread(){
  m_workerThread->start();
}

void ISSFApplication::onStopWorkerThread(){
  m_workerThread->stop();

  fstream logStream;
  logStream.open("./log.txt",fstream::out);
  logStream << m_PacketsCptrdField->document()->toPlainText().toUtf8().data();
  m_PacketsCptrdField->clear();
}

void ISSFApplication::onPauseWorkerThread(){
  m_workerThread->stop();
}
```
Фиг. 3.44 Методи, с които се спира и пуска слушането

Записването на получените пакети е постигнато с методите emitPacketsReceived и onPacketsReceived. emitPacketsReceived изпраща сигнала packetsReceived, свързан със слота onPacketsReceived, който записва съкратената информация за последния записан пакет в текстовото поле за хванати пакети m_PacktetsCptrdField с поредния му номер.

```
void ISSFApplication::onPacketsAdded(){
  if(m_allowedToAdd){
     m_AddedPackets++;
     const Packet& packet = m_sharedMemory->getLatestPacket();
     stringstream sPacketStream;
     sPacketStream << m_AddedPackets << ". ";

     string packetInfo = sPacketStream.str();
     packetInfo += packet.AbridgedHeadersInfo();

     QString qEtherInfo(packetInfo.c_str());
```

```
      m_PacketsCptrdField->append(qEtherInfo);
   }
}

void ISSFApplication::emitPacketsAdded(){
   emit this->packetsAdded();
}
```

Фиг. 3.45 Методи за изписване на информация за получени пакети

За да се разгледа цялата информация за даден пакет, той се маркира и след това се натиска бутона Show Full Info. Този бутон е свързан с onPacketSelected слота. Там се взима номера на пакета от маркираната част и от SharedMemoryController се взима препратка към пакета в паметта. В текстовото поле m_packetsFullInfo се записва цялата информация за този пакет и се забранява записването на нови пакети m_PacketsCptrdField. Когато искаме да продължим със изписването, натискаме Continue. Пакетите, които не са дописани, се допълват и се позволява добавянето на пакети в m_PacketsCptrdField.

```
void ISSFApplication::onPacketSelected(){
   m_allowedToAdd = false;
   QString row = m_PacketsCptrdField->textCursor().selectedText();
   string temps;

   for(QString::iterator it = row.begin(); it != row.end();++it){
      if(*it!='.'){
         char digit = it->toLatin1();
         temps.append(&digit);
      } else {
         break;
      }
   }
   stringstream stemps(temps);
   u_int packetNum=0;
   stemps >> packetNum;
   const Packet& packet = m_sharedMemory->getPacket(packetNum-1);
   QString packetFullInfo = packet.HeadersInfo();

   m_packetsFullInfo->clear();
   m_packetsFullInfo->append(packetFullInfo);
}

void ISSFApplication::onContinue(){
   for(int i = m_AddedPackets;
         i < m_sharedMemory->getVectorSize(); i++){
      const Packet& packet = m_sharedMemory->getPacket(i-1);
```

```
        stringstream sPacketStream;
        sPacketStream << i << ". ";

        string packetInfo = sPacketStream.str();
        packetInfo += packet.AbridgedHeadersInfo();

        QString qEtherInfo(packetInfo.c_str());

        m_PacketsCptrdField->append(qEtherInfo);\

    }
    m_allowedToAdd = true;
}
```

Фиг. 3.46 Методи за изписване на цялата информация в пакета

Прилагането на пакети става като се натисне бутона Apply. Взима се текста от полето за филтри и се подава на метода compileFilter на класа Filter. Той връща компилираната програма, след което се спира се хващането на пакети, прилага филтъра на интерфейса и се пуска на ново слушането.

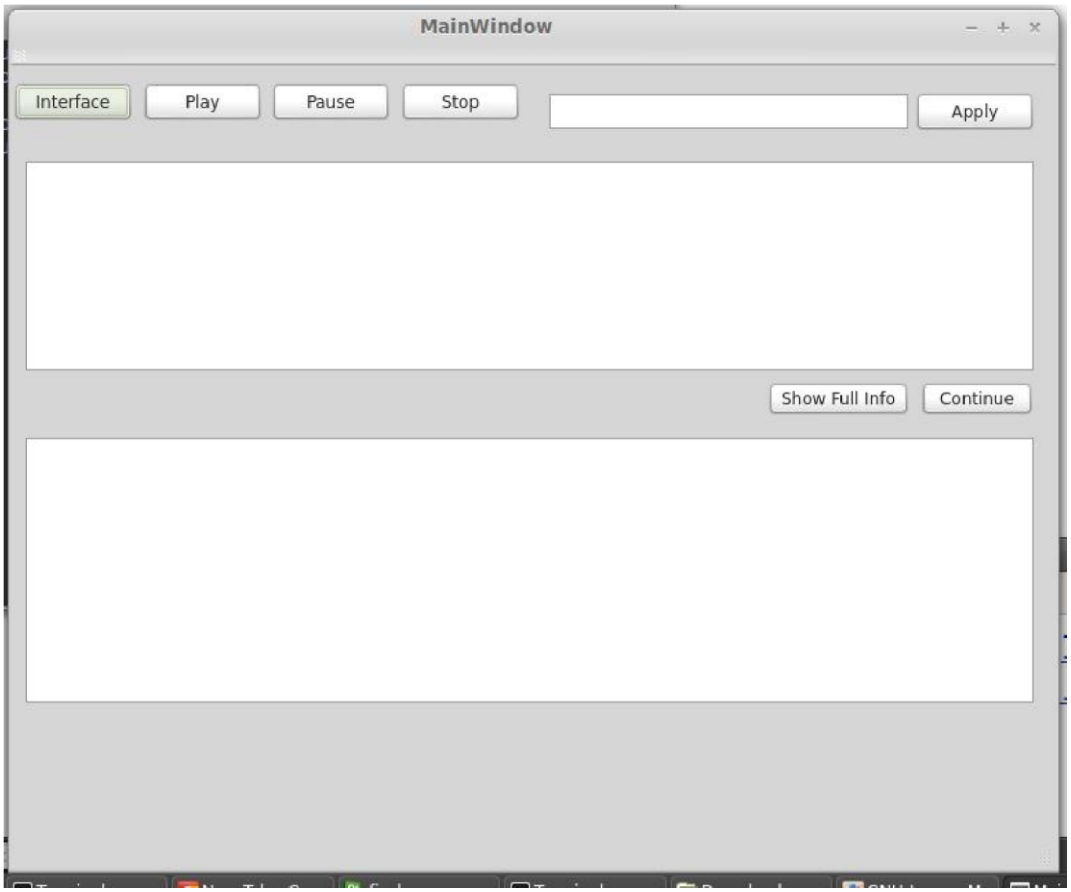
```
void ISSFApplication::onFilterApplied(){
    QTextEdit *filterField = main->getFilterField();
    QString filter = filterField->textCursor().currentFrame()->
                    document()->toPlainText();

    string filterToCompile = filter.toUtf8().data_ptr()->data();

    bpf_program *compiledFilter = m_filter->compileFilter(m_dev,filterToCompile);

    m_workerThread->stop();
    pcap_setfilter(m_dev,compiledFilter);
    m_workerThread->start();
}
```

Фиг. 3.47 Метод за прилагане на филтър

Глава IV.

Ръководство за потребителя

1.Главния прозорец на приложението.



Фиг. 4.1 Главния прозорец на приложението

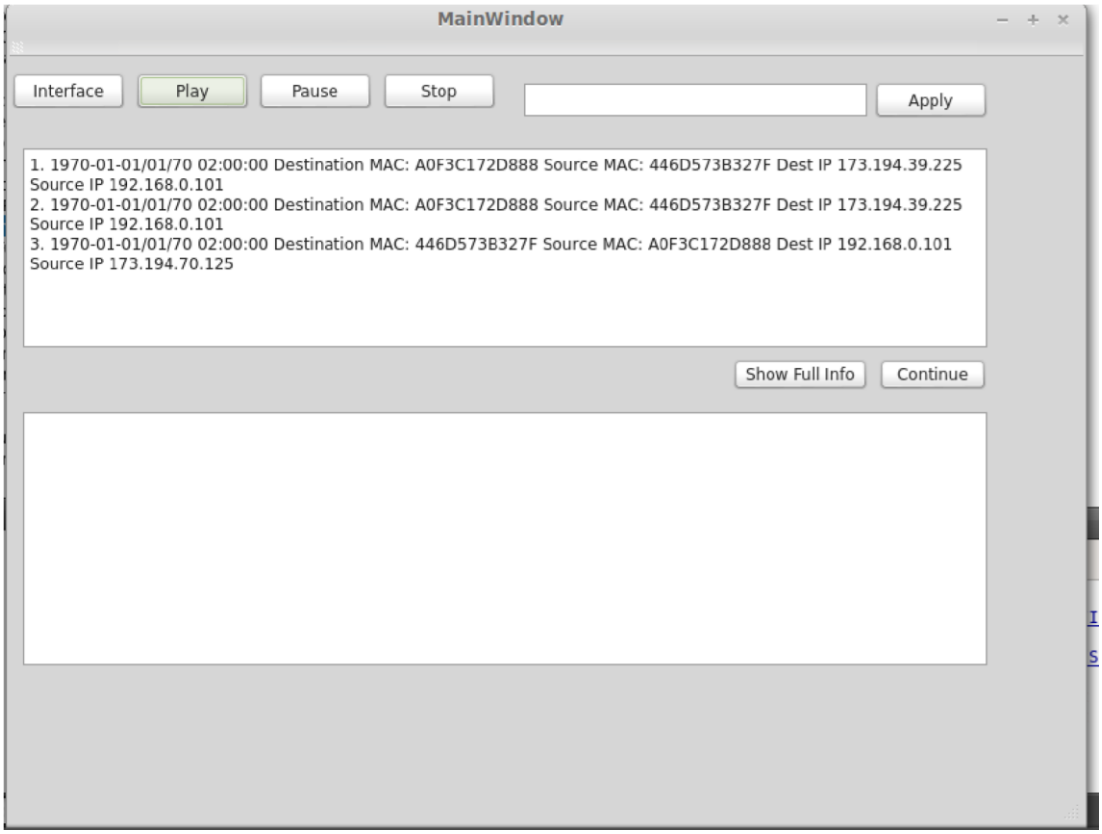
2. Как да изберем интерфейс

Преди да се пусне приложението, трябва да се избере интерфейс за слушане. Това става, като се натисне бутона Interface. Отваря се малък прозорец с всички налични интерфейси на компютъра. Маркираме името и натискаме Ok.

Фиг. 4.2 Прозорец с имената на интерфейсите

3. Управлението на хващане на пакети

Слушането на пакетите се пуска като се натисне Start бутона. Pause спира хващането на пакетите, но не изчиства полето, в което те са записани. Stop спира напълно хващането, изчиства полето и записва информацията от хванатите пакети в log.txt.



Фиг. 4.3 Изписване на хванатите пакети

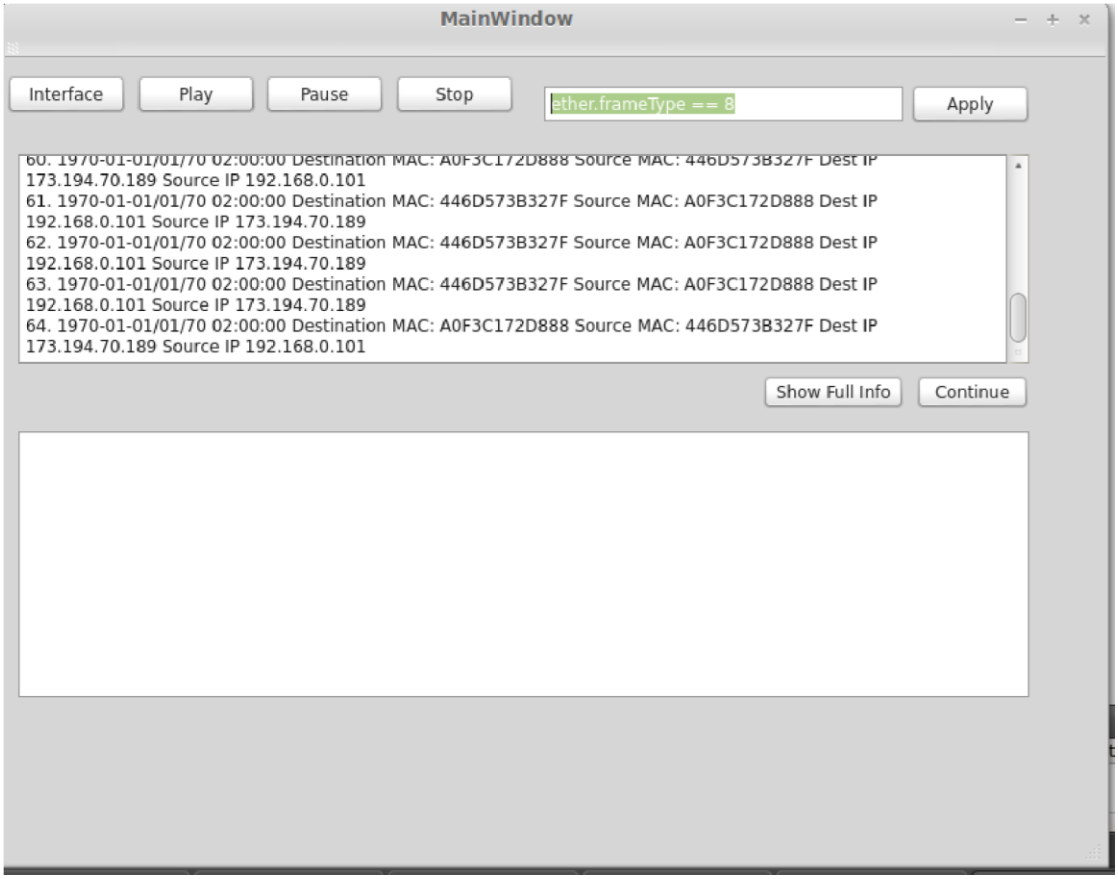
4.Прилагане на филтри

Подобно на Wireshark филтрите изглеждат като С структури.

Общият им вид е:

proto.field comparison **value**

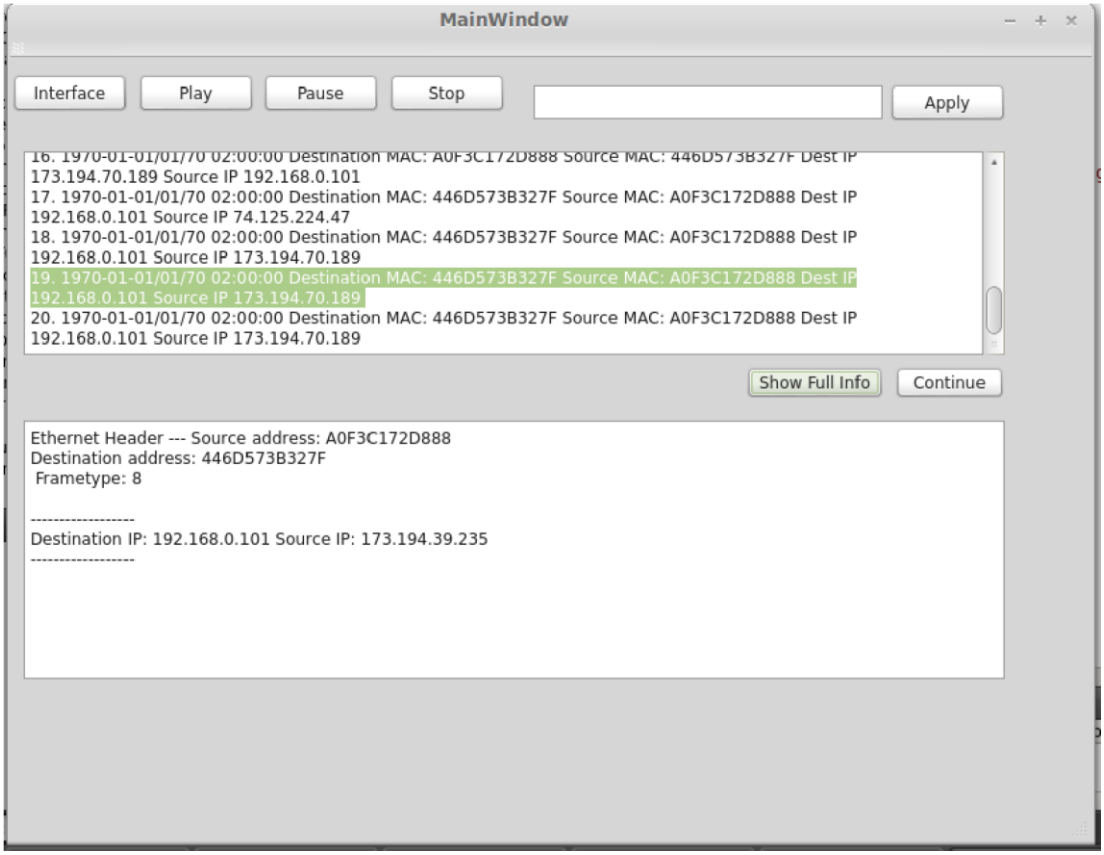
Когато се въведе в текстовото поле, се натиска бутона Apply, за да се приложи.



Фиг. 4.4 Прилагане на филтър

5. Показване на цялата информация за пакет.

Когато искаме да разгледаме цялата информация в пакета, маркираме реда и натискаме бутона Show Full Info. Избраният пакет се изписва в долното текстово поле. Изписването на получените пакети е спряно и ако искаме да продължат да се изписват, натискаме Continue.



Фиг. 4.5 Изписване на цялата информация за пакета

Заключение

Пакетните анализатори предоставят едно много удобно средство за наблюдаване на трафика в мрежа. При възникване на проблеми те са назаменимо средство за диагностициране. Използват се от специалисти в едни от големите корпорации в света до хобиисти, които искат да научат повече за компютърните мрежи.

Идеята за създаването на приложението ISSF е да се предостави възможност за анализиране на пакетите, пристигащи в или излизащи от компютъра. Приложението предоставя информация за хедърите на протоколите, които изпращат този пакет. Основните протоколи, които се разглежда са: Ethernet, WLAN, IP, TCP и HTTP.

Планове за развитие:

- По-подробно изписване на информацията;
- Намиране на BSSID на пакетите от безжична мрежа;
- Добавяне на възможност за използване на файлове със записани филтри;
- Опция за избиране, в какъв файл да се запише информацията за пакетите;

Използвана литература

- pcap библиотека за хващане на мрежови пакети - http://tcpdump.org
- Qt framework за разработване на потребителкси интерфейс - http://qt-project.org/
- boost библиотека за разработване на приложения - http://www.boost.org/
- What’s The Difference Between The OSI Seven-Layer Network Model And TCP/IP? - http://electronicdesign.com/what-s-difference-between/what-s-difference-between-osi-seven-layer-network-model-and-tcpip

Използвани термини

- Стринг(string) – низ от символи
- Хеадър(header) – частта от пакета съдържаща информация за изпращащият протокол
- Фреимъоурк(framework) – сбор от библиотеке за разработване на софтуер