VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

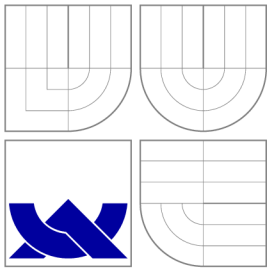
ANALYZÁTOR SÍŤOVÝCH PROTOKOLŮ

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

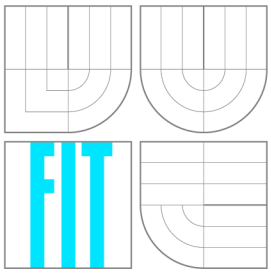
AUTOR PRÁCE
AUTHOR

JURAJ HLÍSTA

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

ANALYZÁTOR SÍŤOVÝCH PROTOKOLŮ

NETWORK PROTOCOL ANALYZER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JURAJ HLÍSTA
AUTHOR

VEDOUCÍ PRÁCE — Ing. JIŘÍ TOBOLA
SUPERVISOR

BRNO 2008

Abstrakt

Bakalárska práca sa zaoberá metódami detekcie protokolov na siedmej vrstve sieťového modelu ISO/OSI. Zameriava sa na návrh a implementáciu systému schopného identifikovať aplikačné protokoly na základe obsahu dátovej časti paketu. Táto časť paketu obsahuje signatúry, ktoré analyzátor vyhľadáva pomocou regulárnych výrazov. Aplikácia je vytvorená v jazyku C s využitím knižnice libpcap, ktorá poskytuje API na odchytávanie sieťovej komunikácie.

Kľúčové slová

Internet, Ethernet, vyhľadávanie reťazcov, sieťová komunikácia, libpcap

Abstract

Bachelor's thesis is concerned with methods of protocol detection at seventh layer of ISO/OSI network model. It focuses on proposal and implementation of a system which is able to identify aplication protocols from packet's data field. This part of packet contains signatures that the analyzer searches for using regular expressions. The application was written in C programming language using libpcap library that provides API for capturing network communication.

Keywords

Internet, Ethernet, pattern matching, network communication, libpcap

Citácia

Juraj Hlísta: Analyzátor síťových protokolů, bakalářská práce, Brno, FIT VUT v Brně, 2008

Analyzátor sít'ových protokolů

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením Ing. Jiřího Tobolu. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Juraj Hlísta
12. mája 2008

Pod'akovanie

Týmto by som chcel pod'akovat' Ing. Jiřímu Tobolovi za poskytnutú pomoc a konzultácie pri tvorbe tejto práce.

© Juraj Hlísta, 2008.
Táto práca vznikla ako školské dielo na Vysokom učení technickom v Brne, Fakulte informačných technológií. Práca je chránená autorským zákonom a jeho použitie bez udelenia oprávnenia autorom je nezákonné, s výnimkou zákonom definovaných prípadov.

Obsah

Kapitola 1

Úvod

V dnešnej dobe sa čoraz viac používa Internet, či už vo firmách alebo v domácnostiach. Vznikajú nové aplikácie, ktoré užívatelia používajú na komunikáciu, zdieľanie súborov, atď. So zväčšovaním sa počtu poskytovaných služieb na Internete rastie aj množstvo nových protokolov, ktoré popisujú pravidlá komunikácie. Rozšírenie Internetu predstavuje mnohé výhody, ale na druhej strane aj problémy. Jedným z nich môže byť zneužitie prístupu na Internet k iným činnostiam, než k akým bolo primárne plánované.

Vzniká tak potreba monitorovania sieťovej komunikácie, pri ktorej je možné detekovať aplikačný protokol a tým v niektorých prípadoch presne určiť aj použitú aplikáciu. Jedným z najznámejších sieťových analyzátorov je *Wireshark* [24], ktorý je schopný odhaliť protokoly na rôznych vrstvách sieťového modelu. Tento analyzátor je schopný identifikovať aplikačné protokoly len na základe čísla portu, žiadnym spôsobom neanalyzuje payload paketu. Ďalším podobným analyzátorom je *L7-filter* [22], ktorý využíva na popis aplikačných protokolov regulárne výrazy, pomocou ktorých analyzuje payload paketu. Analýzu dátovej časti paketu vykonávajú aj programy ako sú *Snort* [16] a *Bro* [21], ktorých úlohou je odhaľovanie útokov alebo vírusov prenášaných prostredníctvom počítačovej siete.

Práca sa v druhej kapitole zaoberá popisom sieťových modelov a ich jednotlivých vrstiev, pomocou ktorých je možné pochopiť princípy sieťovej komunikácie. Podrobnejšie je tu vysvetlený prenos dát na jednotlivých vrstvách sieťového modelu TCP/IP, konkrétne táto kapitola popisuje Ethernet, protokol IPv4 a rozdiely v prenose dát pomocou protokolov TCP a UDP. Ďalej sa táto kapitola zameriava na rôzne algoritmy používané na vyhľadávanie vzorov v texte.

Tretia kapitola sa venuje implementácii systému pre detekciu aplikačných protokolov. Systém tieto protokoly rozpoznáva na základe vyhľadania určitých reťazcov nachádzajúcich sa v payloade paketu, ktoré sú pre každý aplikačný protokol špecifické. Navrhnutý analyzátor sa skladá z rôznych častí (buffer, časovač, binárny vyhľadávací strom, atď), ktoré plnia rôzne funkcie nevyhnutné pre analýzu odchytenej komunikácie.

Piata kapitola obsahuje výsledky a hodnotenie navrhnutého systému. Bolo treba vykonať rôzne testy, podľa ktorých je možné určiť výpočtovo najnáročnejšie časti systému, ktoré sú limitujúce z pohľadu celkovej priepustnosti. Tieto časti je potrebné so zvyšujúcimi sa prenosovými rýchlosťami zdokonalovať, aby neobmedzovali výkonnosť systému ako celku.

Poslednou kapitolou je záver, v ktorom sa nachádza hodnotenie navrhnutého systému z pohľadu ďalšieho vývoja ako aj námety vychádzajúce zo skúseností s riešeným projektom. Hlavne sa jedná o časť analyzátora, ktorá vykonáva vyhľadávanie reťazcov v payloade paketu a je limitujúcou časťou navrhnutého systému.

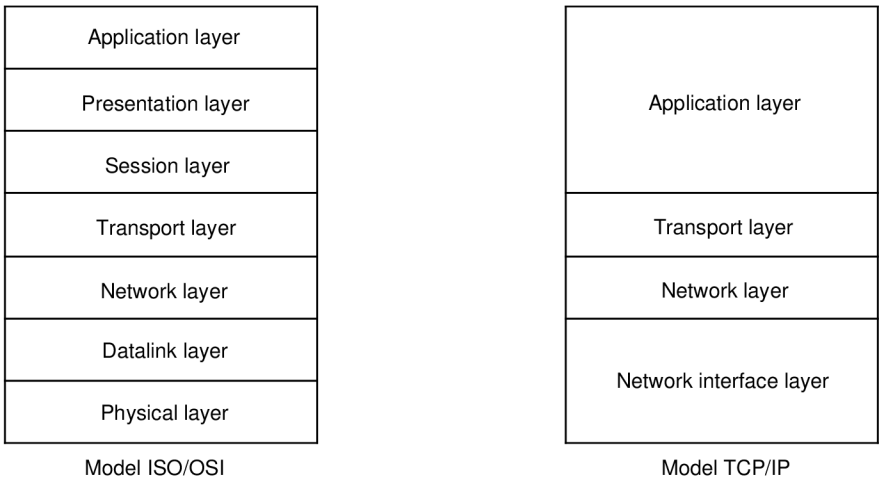
Kapitola 2

Teoretický rozbor

2.1 Modely ISO/OSI a TCP/IP

Pre popis sieťovej komunikácie sa v počítačových sieťach používa princíp vrstiev. Prvým vrstvovým modelom sa stal ISO/OSI, ktorý bol vytvorený medzinárodnou organizáciou ISO (International Standard Organization). Tento štandard je abstraktným modelom a poskytuje základňu pre vypracovanie noriem pre účely prepojovania systémov. Spočiatku vznikali firemné uzavreté sieťové architektúry, ktoré neumožňovali prepojovanie systémov od rôznych výrobcov, v dôsledku toho vznikol tlak na otvorenosť sieťových architektúr, ktorý nakoniec vyústil prijatím referenčného modelu ISO/OSI. Hlavnou úlohou štandardu je snaha o štandardizáciu počítačových sietí.

ISO/OSI nešpecifikuje implementáciu (realizáciu) systému, zameriava sa na všeobecné princípy sedemvrstvovej sieťovej architektúry. Popisuje predovšetkým účel vrstiev, funkcie priradené jednotlivým vrstvám, služby požadované od nižšej vrstvy a poskytované vrstve vyššej. Schéma tohto modelu je uvedená na obrázku 2.1.



Obrázok 2.1: Referenčný model ISO/OSI a model TCP/IP

Funkcie jednotlivých vrstiev sú popísané v nasledujúcich bodoch:

- **Fyzická vrstva** definuje všetky elektrické a fyzikálne vlastnosti zariadenia. Obsahuje

popis napäťových úrovní pri prenose nulových a jednotkových bitov, rýchlostí prenosu jedného bitu, prenosového média a ďalších parametrov, ktoré určujú mechanické a elektrické rozhranie. Fyzická vrstva nevenuje pozornosť významu prenášaných bitov.

- **Linková vrstva** poskytuje spojenie medzi dvoma susednými systémami a vytvára dátový spoj. Organizuje dátový tok do rámcov, čo sú bloky dát s veľkosťou desiatok až tisícov bajtov. Rámec má svoju presnú štruktúru a je v ňom zvyčajne uvedená zdrojová a cieľová adresa. Vrstva umožňuje detekovať chybné rámce a prípadne opraviť chyby, ktoré vznikli na prvej vrstve.

- **Sieťová vrstva** sa stará o smerovanie, poskytuje spojenie medzi systémami, ktoré spolu priamo nesusedia. Prenos môže prebiehať cez niekoľko uzlov. Pre zaistenie toku dát sieťou je potreba jednoznačnej identifikácie uzlu, ktorá je nezávislá na fyzickej adresácii. Táto identifikácia sa nazýva sieťová adresa a je priradená každému uzlu siete. Dáta sú pre účely smerovania delené na datagramy.

- **Transportná vrstva** zaručuje adresovanie koncových komunikujúcich prvkov, ktoré pôsobia v jednotlivých uzloch siete. Medzi tieto koncové prvky patria napríklad procesy alebo užívateľské relácie. Dáta sú prenášané v segmentoch, ktoré obsahujú adresu koncového prvku uzlu. Na úrovni transportnej vrstvy môžu existovať spojované aj nespojované služby. U spojovaných služieb je zaistené spoľahlivé doručovanie paketov. Naopak u nespojovaných služieb sa o spoľahlivé doručenie dát musí postarať priamo aplikácia. Transportná vrstva je v modely ISO/OSI preto, aby mohla poskytovať kvalitnejšie prenosové služby, aké dokáže poskytovať vrstva sieťová.

- **Relačná vrstva** umožňuje, aby si procesy alebo užívatelia stanovili medzi sebou relácie a následne mohli koordinovať výmenu dát medzi týmito uzlami.

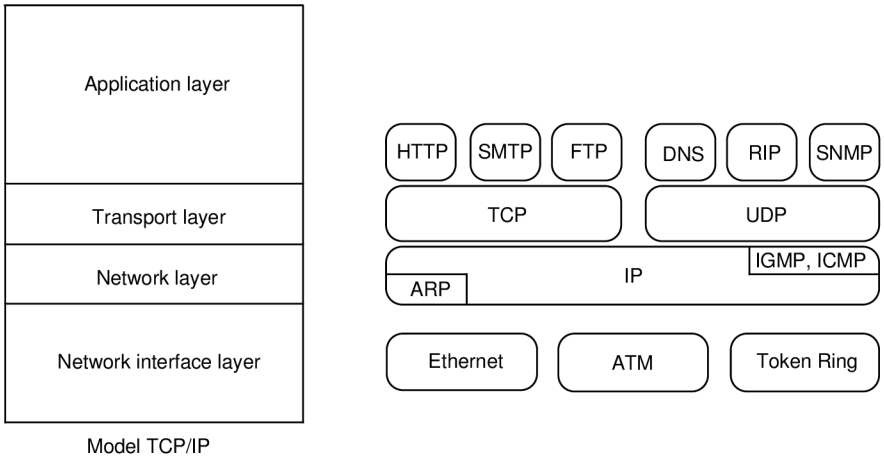
- **Prezentačná vrstva** obsahuje funkcie, ktoré sú vykonávané tak často, že je pre ne vhodné mať všeobecné riešenie. Užívatelia potom nemusia tieto funkcie riešiť vo vlastnej réžii. Medzi funkcie prezentačnej patrí typicky šifrovanie, prevod medzi rôznymi kódovaniami alebo komprimácia.

- **Aplikačná vrstva** realizuje aplikačne orientované služby. Poskytuje rôzne aplikačné rozhrania, ako napríklad služby pre implementáciu elektronickej pošty, prenosu súborov alebo elektronického obchodu. Súčasťou tejto vrstvy bývajú procesy, ktoré túto činnosť priamo plnia.

Aj keď je referenčný model ISO/OSI medzinárodne štandardizovaný a uznávaný, hlavnú úlohu v oblasti sieťových architektúr má v súčastnosti model TCP/IP. Tento model má na rozdiel od ISO/OSI jednoduchšiu a efektívnejšiu architektúru rozdelenú iba do štyroch vrstiev. Porovnanie oboch modelov znázorňujúcich vzájomne si zodpovedajúce vrstvy je uvedené na obrázku 2.1.

Model TCP/IP vznikol ešte v začiatkoch Internetu. V skutočnosti ide o celú sústavu (rodinu) jednotlivých protokolov, ktoré sú spojené spoločnou predstavou o tom, ako by mali vypadať a fungovať. Táto predstava vznikla v sedemdesiatych rokoch v USA pre potreby budovania vtedy sa rodiaceho Internetu, ktorý je na protokoloch TCP/IP vybudovaný. Hlavné koncepcie tohto modelu sú abstrakcia od fyzickej a linkovej vrstvy, definícia protokolu sieťovej vrstvy IP (Internet Protocol) pre jednotné adresovanie, definícia prevodných mechanizmov medzi linkovou a sieťovou vrstvou (protokol ARP) a definícia transportných

protokolov TCP a UDP. Model bol primárne navrhnutý pre použitie v Internete, neobsahuje žiadne centrálne časti, je veľmi robustný a odolný voči výpadkom častí siete.



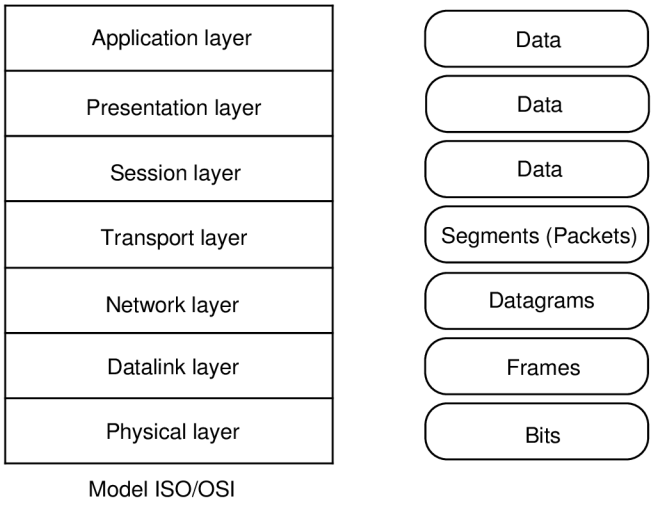
Obrázok 2.2: Sieťový model TCP/IP

- **Vrstva sieťového rozhrania** nie je modelom TCP/IP bližšie špecifikovaná, pretože je závislá na použitej prenosovej technológii (napr. ATM, Token Ring alebo Ethernet).
- **Sieťová vrstva** zakrýva rozdiely medzi jednotlivými prenosovými technológiami použitými v nižšej vrstve. Cieľom je vytvorenie jednotného komunikačného prostredia nezávislého na použitej technológii prenosu dát. Na tejto vrstve je definovaný protokol IP (Internet Protocol), ktorý zaisťuje dané požiadavky. V súčastnej dobe sú využívané dva typy IP protokolu: IPv4 [13] a nová verzia tohto protokolu IPv6 [4].
- **Transportná vrstva** ponúka spojovanú aj nespojovanú komunikáciu. Protokol TCP (Transmission Control Protocol) [14] zaisťuje spoľahlivosť prenosu a riadenie toku dát. Naopak, protokol UDP (User Datagram Protocol) [15] je nespojový a nezaručuje spoľahlivý prenos dát. Oba zo spomínaných protokolov majú svoje výhody aj nevýhody (napr. rýchlosť a spoľahlivosť prenosu dát).
- **Aplikačná vrstva** je najvyššou vrstvou sieťovej architektúry Internetu. Poskytuje služby, ktoré aplikácie využívajú pri komunikácii. Táto vrstva obsahuje protokoly ako napríklad HTTP, FTP, SMTP, DHCP, DNS a mnoho ďalších.

2.2 Prenos dát

Dáta sú cez sieť prenášané zvyčajne po častiach nazývaných PDU (Protocol Data Unit), ktoré majú na rôznych vrstvách rôzne názvy. Podľa modelu ISO/OSI sú to postupne od fyzickej vrstvy: bity, rámce, datagramy, pakety a od piatej po siedmu vrstvu dáta ako je zrejmé z obrázku 2.3. V procese prenosu dát po sieti dochádza k ich zapuzdrovaniu. Je to mechanizmus, ktorý poskytuje abstrakciu protokolov a služieb. Aplikácie medzi sebou komunikujú na najvyššej vrstve ISO/OSI modelu a pri tejto komunikácii využívajú nižšie vrstvy, ktoré si dáta predávajú a pridávajú im potrebné informácie. Komunikáciu je

možné realizovať iba medzi rovnakými vrstvami sieťového modelu, ktoré popisujú rovnaké protokoly.

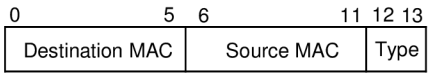


Obrázok 2.3: Názvy jednotlivých PDU

Postup zapuzdrovania dát by sa dal popísať nasledovne: k dátam na siedmej vrstve ISO/OSI modelu sa na štvrtej vrstve pridáva hlavička protokolu, ktorý je špecifikovaný na danej vrstve. Z dát zo siedmej vrstvy sa stáva paket (je to vlastne hlavička protokolu na štvrtej vrstve spojená s dátami). O vrstvu nižšie sa z paketu stáva datagram. Na tretej vrstve sa nachádza odlišný protokol plniaci inú funkciu a vkladá do svojej dátovej časti PDU z vyššej vrstvy (v tomto prípade je dátovou časťou datagramu paket). Na druhej vrstve sa datagram zapuzdruje do rámca, v ktorom sa tiež nachádza hlavička špecifická pre protokol nachádzajúci sa na tejto vrstve. Na fyzickej vrstve sa rámec mení na bity, pričom sa tu môžu vyskytovať nielen bity reprezentujúce rámec, ale aj ďalšie, ktoré plnia napríklad synchronizačnú funkciu.

2.2.1 Ethernet

Ethernet [7] je sieťová technológia, ktorá sa v modely TCP/IP nachádza na prvej vrstve a v modely ISO/OSI pokrýva prvé dve vrstvy. Je to štandard, ktorý popisuje fyzické rozhranie a komunikáciu na LAN (Local Area Network) sieťach a bude využitý v navrhovanom systéme. Popularita Ethernetu spočíva v jednoduchosti protokolu a tým aj jednoduchej implementácii a inštalácii. Z pohľadu analyzátora protokolov je dôležité poznať formát hlavičky Ethernetového rámca, ktorý sa nachádza na obrázku 2.4.



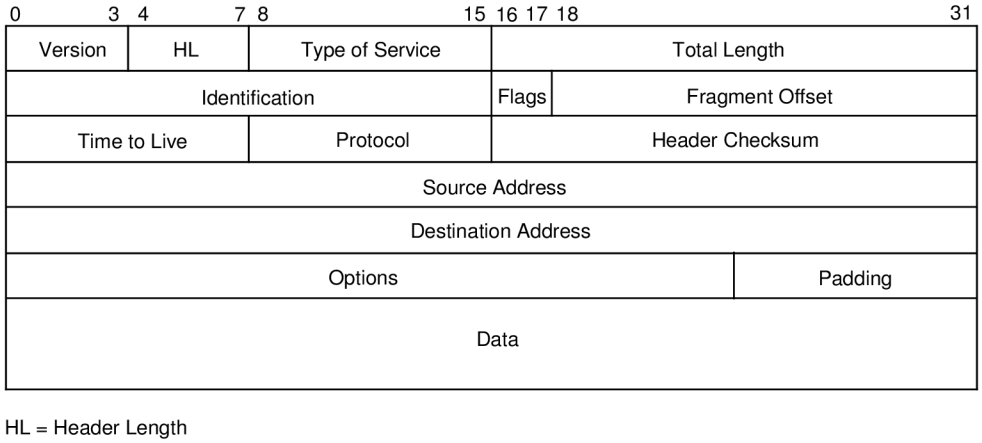
Obrázok 2.4: Formát Ethernetovej hlavičky

Na obrázku sa nachádza iba hlavička Ethernetového rámca, ktorý navyše obsahuje

payload (dáta, ktoré prenáša) a d'alšie informácie (Preamble, Start-of-Frame Delimiter, CRC Checksum), ktoré slúžia na detekciu začiatku rámca, prípadne odhalenie chyby v prenášaných dátach. Tieto údaje nie sú z hľadiska analyzátora sieťových protokolov podstatné, najdôležitejšiu úlohu má hlavička a payload. Ak je položka Type v hlavičke väčšia ako 600 hexadecimálne, určuje aký typ protokolu na nachádza na vyššej vrstve, v opačnom prípade sa jedná o veľkosť dát.

2.2.2 Protokol IP

Rovnako ako Ethernet aj sieťová vrstva popisuje formát hlavičky pre určitý protokol, ktorý sa na tejto vrstve nachádza. V modely TCP/IP je týmto protokolom IPv4 [13] alebo IPv6 [4]. Na obrázku 2.5 je formát hlavičky protokolu IPv4. Pretože IP protokol tvorí spolu s transportnou vrstvou základ Internetu, sú tieto protokoly a formáty ich datagramov(paketov) veľmi dôležité pri konštrukcii sieťových zariadení - od smerovačov cez firewally a IDS (Intrusion Detection System)/IPS (Intrusion Prevention System) systémy, ktoré často sledujú práve toky (flow) na základe IP adries a portov transportnej vrstvy. Dôležitými položkami pre analyzátor sú Header length (dĺžka hlavičky), Total length (cel-



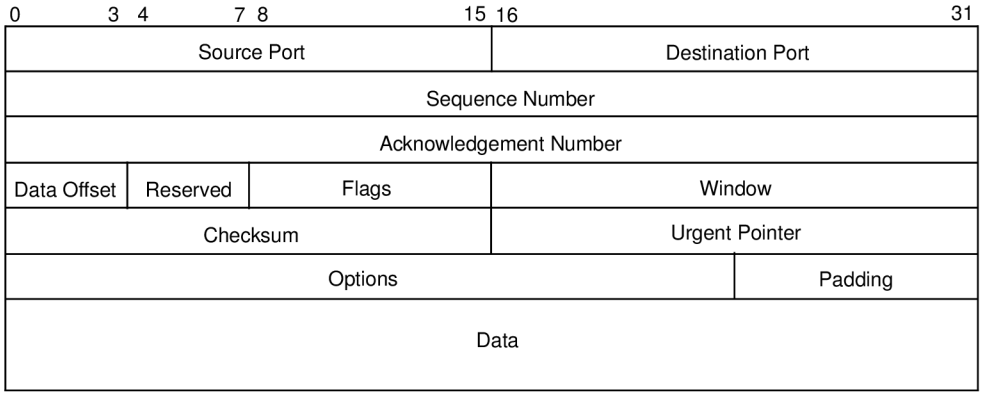
Obrázok 2.5: Formát hlavičky IPv4

ková dĺžka datagramu), zdrojová a cieľová IP adresa, Header Checksum (kontrolný súčet) a Protocol, ktorý určuje typ protokolu na vyššej vrstve.

2.2.3 TCP a UDP

Protokol TCP [14] poskytuje spoľahlivý prenos dát medzi koncovými aplikáciami, ktorý nie je v modele TCP/IP priamo zaistený protokolom sieťovej vrstvy. TCP využíva služby IP protokolu opakovaným odosielaním stratených paketov, preusporiadava prijaté pakety do správneho poradia. TCP zaisťuje riadenie koncového spojenia a dátového toku. Nadväzovanie spojenia sa realizuje pomocou mechanizmu *three way handshaking*. K rozlíšeniu komunikujúcich aplikácií TCP používa čísla portov, ktoré sú pridelené daným aplikáciám. Na obrázku 2.6 sa nachádza formát hlavičky protokolu TCP.

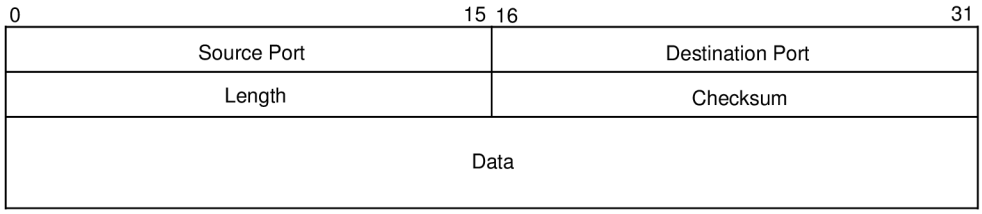
Z pohľadu analyzátora sú dôležitými hodnotami čísla portov ako aj sequence a acknowledgement number (sekvenčné a potvrdzovacie číslo). Tieto dve posledne spomenuté čísla



Obrázok 2.6: Formát TCP hlavičky

slúžia na detekciu straty paketu alebo na usporiadanie prijatých paketov do správneho poradia.

Protokol UDP [15] je oveľa jednoduchší ako TCP, jeho funkciou je iba rozlíšenie komunikujúcich aplikácii k čomu rovnako ako TCP používa čísla portov. UDP nie je spojovaný, nezaručuje spoľahlivý prenos dát, ani ich doručenie v správnom poradí. Z toho vyplýva že nepotrebuje sequence a acknowledgement number, a preto je hlavička UDP protokolu jednoduchšia, ako je vidieť na obrázku 2.7



Obrázok 2.7: Formát UDP hlavičky

Výhodou protokolu TCP je jeho spoľahlivosť, ale tým sa stáva prenos dát pomalejším, pretože je potreba zasielania potvrdenia o prijatí každého paketu. Naopak protokol UPD slúži na rýchly prenos dát, je síce nespoľahlivý, ale svoje uplatnenie má predovšetkým v prenose obrazu a zvuku (VoIP, audio-video-streaming).

2.2.4 Payload

Payload sa dá v modele TCP/IP definovať ako datagram bez TCP alebo UDP hlavičky. Sú to dáta, ktoré si medzi sebou posielajú komunikujúce aplikácie väčšinou na základe určitého protokolu. Payload môže mať formu ASCII textu alebo binárnych dát.

2.3 Metódy analýzy protokolu

Určovanie typu protokolu na prvej až tretej vrstve TCP/IP modelu nie je zložité, pretože táto informácia sa nachádza v hlavičke protokolu nižšej vrstvy, prípadne je potrebné poznať typ dátovej linky. Transportná vrstva, ktorá sa v modely TCP/IP nachádza pod aplikačnou priamo neurčuje aký typ protokolu je na vrstve vyššej. Poskytuje iba informáciu, na ktorom porte daná služba beží. Určité služby, ktoré sú v počítačových sieťach štandardne používané (napr. HTTP, DHCP, DNS) majú pridelené štandardné čísla portov [6].

Prvým spôsobom určenia typu protokolu aplikačnej vrstvy je číslo portu. Tento prístup je nepostačujúci z pohľadu bezpečnosti a rozpoznávania protokolov na siedmej vrstve, pretože mnohé aplikácie v dnešnej dobe využívajú dynamické prideľovanie čísla portu a tým sa stáva tento typ identifikácie protokolu neúčinným. Ďalším nedostatkom tohto spôsobu je nemožnosť overenia, že na určitom štandardnom porte naozaj beží aplikácia, ktorá tento port štandardne využíva.

Ďalší spôsob detekcie protokolu aplikačnej vrstvy ja založený na tzv. *signatúrach*, ktoré sa nachádzajú v payloade paketu. Sú to určité reťazce, ktoré sú špecifické pre daný protokol a dajú sa popísať regulárnymi výrazmi. Ich vyhľadávanie je v systémoch IDS výpočtovo najnáročnejšou časťou, ale rieši nedostatky identifikácie protokolov na základe čísel portov. Tento spôsob má tiež svoje nevýhody, napríklad je veľmi obtiažne popísať niektoré protokoly (Skype) regulárnymi výrazmi a priepustnosť takéhoto systému je často limitovaná rýchlosťou vyhľadania signatúr.

2.4 Algoritmy pre vyhľadávanie reťazcov

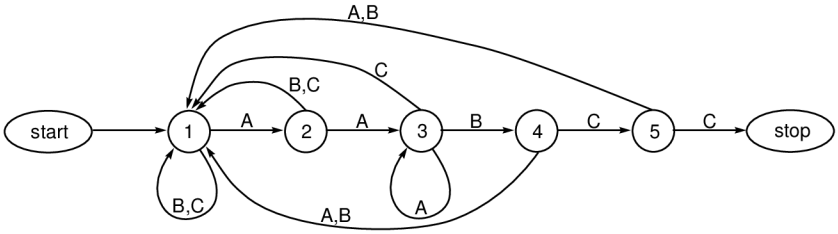
Vyhľadávanie signatúr je hľadanie podreťazca v reťazci a pre tento problém existuje veľké množstvo algoritmov, ktoré vylepšujú tzv. *naivný algoritmus*. Ten je založený na postupnom porovnávaní hľadaného podreťazca a vstupného textu, pričom sa po každom porovnaní posunie vo vstupnom texte o jeden znak. Počet porovnaní tohto algoritmu je konštantný a pre hľadaný podreťazec dĺžky m a prehľadávaný text dĺžky n má zložitosť vyhľadávania $O(m * n)$. Tento algoritmus je neefektívny a v praxi sa preto nevyužíva. Pre vyhľadávanie podreťazcov sú využívané efektívne algoritmy, ktoré sú popísané v nasledujúcich bodoch:

- **Boyer-Mooreov algoritmus** [2] – sa snaží redukovať počet porovnaní a tým vyhľadávanie urýchliť. Algoritmus neporovnáva každý znak v prehľadávanom texte, ak sa v texte nevyskytuje nejaký znak zo vzoru, je možné určité znaky preskočiť. Chovanie algoritmu závisí na kardinalite abecedy a na opakovaní podreťazcov vo vzore. Pokusy ukázali, že pre dĺžku vzorku $m > 5$ algoritmus robí približne 0,24 až 0,3 porovnaní z počtu znakov v prehľadávanom texte. Inými slovami, porovnáva asi jednu štvrtinu až jednu tretinu znakov prehľadávaného textu.

- **Nedeterministický konečný automat** [18] – pred vyhľadávaním je potrebné zostrojiť automat. Pomocou neho je možné vyhľadávať niekoľko vzorov naraz. V najhoršom prípade je počet porovnaní rovný súčtu dĺžok všetkých vyhľadávaných vzorov. Automat vytvára stavový priestor, ktorý je tvorený konečným množstvom stavov a prechodov. Vstupný text je spracovávaný po jednom znaku, pričom dochádza v automate k prechodu medzi stavmi. Tento automat je nedetermistický pretože pre každý znak, ktorý prijme, nie je jeho ďalší stav jednoznačne určený a môže prejsť do jedného z viacerých možných stavov. Stavový priestor je pomocou tejto metódy prehľadávaný

do hĺbky a dochádza k tzv. *backtrackingu* čo je návrat z určitého stavu do predošlého a následne pokračovanie prehľadávania nepreskúmaného stavového priestoru. Softwarové riešenie nie je dostatočne rýchle kvôli backtrackingu, ktoré vyhľadávanie značne spomaľuje.

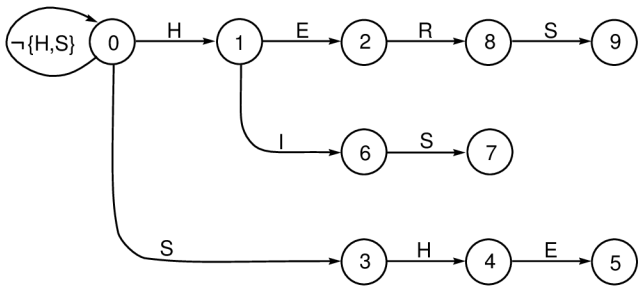
- **Deterministický konečný automat** [17] – sa od nedeterministického automatu líši hlavne v spôsobe prehľadávania stavového priestoru, ktorý prehľadáva do šírky. V najhoršom prípade je počet porovnaní rovný dĺžke najdlhšieho vzoru a vlastné vyhľadanie prebehne v n krokoch. Pri načítaní znaku zo vstupného textu môže automat prechádzať súčastne do viacerých stavov. Príklad takéhoto automatu je pre hľadaný vzor "AABCC" uvedený na obrázku 2.8. Nevýhodou je, že z každého stavu môže vychádzať až toľko znakov, koľko je mocnosť abecedy.



Obrázok 2.8: Deterministický konečný automat pre vyhľadávanie vzoru "AABCC"

- **Knuth-Morris-Prattov algoritmus** [10] – odstraňuje nevýhodu predchádzajúceho riešenia. Pre vyhľadávanie konštruuje špeciálny KPM graf, v ktorom vychádza z každého stavu iba dvojica hrán. Snahou algoritmu je v prípade nezhody vzoru s textom, vzor posunúť doprava tak, aby nebolo potrebné sa v prehľadávanom texte vrátiť. Celková zložitosť algoritmu je $O(m + n)$ a jeho nevýhodou je, že pre spracovanie znaku zo vstupu potrebuje niekedy viacej ako jedno porovnanie.

- **Aho-Corasickov algoritmus** [1] – vyhľadáva vzory z tzv. slovníka. Porovnáva všetky vzory so vstupný textom naraz v $m + n$ krokoch, kde m je súčet dĺžok všetkých hľadaných vzorov. Rovnako ako predchádzajúci algoritmus, je založený na konštrukcii grafu. Príklad takéhoto grafu pre vyhľadávanie retazcov "HE", "SHE", "HIS", "HERS" je na obrázku 2.9.



Obrázok 2.9: Aho-Corasickov graf pre vyhľadávanie vzorov "HE", "SHE", "HIS" , "HERS"

- **Rabin-Karpov algoritmus** [9]– je založený na tzv. *hashovacej funkcii*. Využíva sa na vyhľadávanie viacerých vzorov naraz. Namiesto porovnávania jednotlivých znakov používa hashovaciu funkciu na transformáciu hľadaných vzorov na čísla a v rovnako transformovanom texte potom hľadá podobnosť. Zložitosť tohto algoritmu je v najhoršom prípade $O(m * n)$ a to je dôvod prečo sa veľmi nepoužíva. Ďalšou nevýhodou je, že tento algoritmus nezaručuje vyhľadanie reťazca zodpovedajúceho danému vzoru, pretože pre dva rôzne reťazce môže byť výsledné transformačné číslo rovnaké.

Kapitola 3

Systém pre detekciu protokolov

Táto kapitola popisuje implementáciu systému pre detekciu protokolov aplikačnej vrstvy. Ako implementačný jazyk bol zvolený jazyk C, ktorého kód je pri dodržiavaní štandardov prenositeľný medzi rôzne operačné systémy a výsledný program je v porovnaní s programami napísanými v jazykoch poskytujúcich vyššiu mieru abstrakcie (C++, Java) rýchlejší.

3.1 Odchytávanie dát

Navrhnutý systém používa na odchytávanie sieťovej komunikácie programátorské rozhranie (API) pcap (packet capture). Implementácia pcap pre Unixové systémy sa nazýva libpcap [23], je to knižnica napísaná v jazyku C. Toto API poskytuje možnosť zistenia typu dátovej linky, v navrhnutom systéme sa nachádza podpora pre dátovú linku typu Ethernet. Je možné odchytávať aj komunikáciu, ktorá nie je priamo určená pre dané zariadenie, sieťová karta však musí pracovať v tzv. promiskuitnom režime. Výhodou libpcap je, že poskytuje paketový filter tzv. *BSD packet filter* [11] a tým umožňuje filtrovať komunikáciu pred tým ako sú dáta spracovávané samotným analyzátorom. Tento filter dokáže filtrovať komunikáciu na tretej a štvrtej vrstve modelu TCP/IP.

Pri odchytávaní dát je dôležité, aby ich spracovanie bolo dostatočne rýchle. Čas ktorý prebehne od začiatku do konca spracovania jedného rámca by mal byť kratší ako rozdiel časov odchytenia dvoch za sebou idúcich rámcov. V opačnom prípade môže dochádzať k strate odchytených dát čo by mohlo znemožniť určenie typu aplikačného protokolu. Objem odchytených dát je preto vhodné redukovať spomínaným filtrom. Primárnou úlohou analyzátora je identifikovať protokoly aplikačnej vrstvy, a preto je dobré zvoliť filter, ktorý bude posielať analyzátoru iba vhodné dáta k analýze. Tieto dáta sú pakety prenášané pomocou protokolov TCP a UDP, filter by teda mal ostatnú komunikáciu blokovať. Toto je možné dosiahnuť nadstavením filtra výrazom `tcp or udp`.

Pri každom odchytenom rámci libpcap poskytuje ukazateľ na jeho začiatok a zároveň štruktúru, ktorá je definovaná nasledovne:

```
struct pcap_pkthdr
{
   sruct timeval ts;            /* time stamp */
   bpf_u_int32 caplen;          /* length of portion present */
   bpf_u_int32 len;             /* length this packet (off wire) */
};
```

`ts` je čas, v ktorom bol rámec odchytený, `caplen` predstavuje veľkosť odchytených dát analyzátorom a `len` skutočnú dĺžku dát. Je možné odchytávať iba časti komunikácie, napríklad v Ethernete len prvých 14 bajtov, a tak analyzovať iba Ethernetovú hlavičku. Navrhnutý systém odchytáva a analyzuje nielen časť komunikácie, ale všetky vrstvy sieťového modelu, preto odchytáva celé rámce vrátane payloadu. Libpcap poskytuje vysokú mieru abstrakcie od fyzickej a linkovej vrstvy, nie je potrebné poznať žiadne detaily týkajúce sa vrstvy sieťového rozhrania modelu TCP/IP.

3.2 Dekódovanie komunikácie

Analýza odchytených dát začína na druhej vrstve modelu ISO/OSI. Nie je možné hneď analyzovať siedmu vrstvu, najskôr sa treba k týmto dátam dostať. Postupná analýza začínajúca na linkovej vrstve je nevyhnutná, pretože na rôznych vrstvách sa nachádzajú rôzne protokoly, ktoré navyše nemusia mať pevne stanovenú dĺžku hlavičky. Z toho vyplýva, že sa nedá napríklad z Ethernetového rámca určiť, kde sa nachádza samotný payload paketu, treba sa k nemu prepracovať postupným dekódovaním hlavičiek protokolov na jednotlivých vrstvách. Určenie protokolu na vyššej vrstve zvyčajne závisí od hlavičky protokolu nižšej vrstvy, ktorá túto informáciu obsahuje. Toto pravidlo ale neplatí pre transportnú vrstvu.

Dekódovanie hlavičiek je proces, pri ktorom je potrebné poznať ich formát. Jednotlivé formáty hlavičiek protokolov, ktoré analyzátor podporuje, sú opísané v predchádzajúcej kapitole. Podľa formátu hlavičky je treba vytvoriť zodpovedajúcu štruktúru v jazyku C. Ethernetovej hlavičke z obrázku 2.4 prislúcha nasledujúca štruktúra:

```
typedef struct _ethernetHdr
{
   uint8_t dst[ETHER_ADDR_LEN];  /* destination MAC address */
   uint8_t src[ETHER_ADDR_LEN];  /* source MAC address */
   uint16_t type;                /* type or length */
} ethernetHdr;
```

Dĺžka tejto hlavičky je 14 bajtov a je pevne stanovená. Ak je dĺžka odchytených dát menšia ako dĺžka hlavičky, znamená to, že nastala chyba a analýza daného rámca je ukončená. Analýza na linkovej vrstve je len prvým krokom v procese dekódovania. Ak položka `type` obsahuje hodnotu väčšiu ako 600 hexadecimálne, potom táto hodnota predstavuje typ protokolu sieťovej vrstvy, v opačnom prípade sa jedná o dĺžku odchytených dát. Táto položka je kľúčová z pohľadu analyzátora, ktorý podľa tejto hodnoty určuje funkciu, ktorá bude v procese dekódovania použitá ako nasledujúca.

Princíp získavania údajov pomocou vytvorenej štruktúry spočíva v preložení štruktúry cez odchytené dáta, pričom je k dispozícii ukazateľ na začiatok dát. Odchytené dáta sú reprezentované bajtami a ukazateľ na ne je typu `unsigned char *`. Inými slovami sa dá povedať, že dochádza k pretypovaniu odchytených dát na štruktúru, ktorá popisuje hlavičku Ethernetového rámca.

Navrhnutý systém podporuje na sieťovej vrstve protokol IPv4, k jeho dekódovaniu dochádza v prípade, ak je položka `type` v Ethernetovom rámci rovná hodnote 800 hexadecimálne. Rovnako ako pri Ethernete, je potrebné vytvoriť štruktúru popisujúcu hlavičku protokolu IPv4, ktorá je uvedená na obrázku 2.5. Definícia danej štruktúry v jazyku C:

```
typedef struct _IPv4Hdr
{
   uint8_t ver_hlen;            /* version and header length */
   uint8_t tos;                 /* type of service */
   uint16_t len;                /* total length */
   uint16_t id;                 /* identification */
   uint16_t offset;             /* flags and fragment offset */
   uint8_t ttl;                 /* time to live */
   uint8_t proto;               /* protocol */
   uint16_t chsum;              /* checksum */
   struct in_addr src;          /* source IP address */
   struct in_addr dst;          /* destination IP address */
} IPv4Hdr;
```

Pred dekódovaním IPv4 hlavičky je potrebné posunúť ukazateľ v odchytených dátach na pozíciu, ktorá predstavuje začiatok tejto hlavičky, a teda aj začiatok IP datagramu. Posun bude v tomto prípade o dĺžku Ethernetovej hlavičky, takže sa ukazateľ posunie zo začiatku odchytených dát a bude ukazovať na trinástu pozíciu pri indexácii bajtov od nuly. Znova dochádza k pretypovaniu, teraz však na štruktúru hlavičky protokolu IPv4. Položka `proto` určuje, aký protokol sa nachádza na vyššej (transportnej) vrstve.

Systém na transportnej vrstve podporuje protokoly TCP a UDP, spôsob ich dekódovania je v podstate rovnaký ako u protokolu IPv4. Na obrázkoch 2.6 a 2.7 sa nachádzajú formáty hlavičiek protokolov TCP a UDP. Definícia štruktúry pre UDP hlavičku je nasledovná:

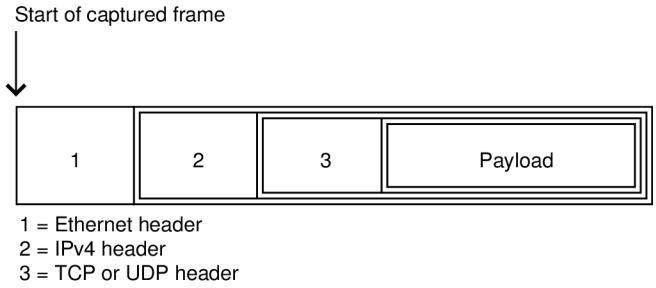
```
typedef struct _UDPHdr
{
   uint16_t src_p;              /* source port */
   uint16_t dst_p;              /* destination port */
   uint16_t len;                /* total length */
   uint16_t chsum;              /* checksum */
} UDPHdr;
```

Pred získaním samotného payloadu paketu je potrebné už iba odstrániť hlavičku protokolu transportnej vrstvy, čo sa dosiahne posunutím ukazateľa na príslušnú pozíciu. Princíp, podľa ktorého sa posúva ukazateľ v odchytenom rámci, je uvedený na obrázku 3.1. Po dekódovaní určitej hlavičky sa ukazateľ posunie o dĺžku tejto hlavičky na novú pozíciu, na ktorej sa nachádza hlavička protokolu vyššej vrstvy, prípadne samotný payload paketu.

Počas procesu dekódovania je potrebné zhromažďovať niektoré informácie, ktoré sú kľúčové z hľadiska ďalšej analýzy popísanej v nasledujúcom texte. Tieto informácie sa nachádzajú na sieťovej a transportnej vrstve, patria sem napríklad zdrojové a cieľové IP adresy alebo flagy v TCP pakete.

3.3 Monitorovanie dátového toku

Pre analyzátor je dôležité triediť odchytenú komunikáciu do dátových tokov (spojení) a určiť aplikačný protokol pre daný tok. Rozpoznávanie dátového toku je založené na zdrojových a cieľových IP adresách a číslach portov. Sú to informácie, ktoré jednoznačne identifikujú spojenie medzi dvoma koncovými bodmi v sieti. Dátový tok by sa dal popísať



Obrázok 3.1: Ukazateľ v odchytenom rámci

ako súbor datagramov (paketov), ktoré majú rovnaké IP adresy a zároveň aj čísla portov. Základnou dátovou štruktúrou navrhnutého analyzátora je tzv. *hashovacia tabuľka* [3], v ktorej sa nachádzajú informácie popisujúce jednotlivé dátové toky. IP adresy a čísla portov hashovacia funkcia transformuje na číslo, ktoré predstavuje index v hashovacej tabuľke. Toto je relatívne rýchle, ale môže sa stať, že pre rôzne IP adresy a čísla portov bude výsledok transformačnej funkcie rovnaký, čiže na rovnakom indexe sa bude nachádzať viacero spojení. Pri každom odchytenom pakete sa zisťuje, či patrí do určitého spojenia, ktoré sa v systéme už nachádza. Ak je kombinácia IP adries a čísel portov v systéme nová, alokuje sa v hashovacej tabuľke pamäť pre nové spojenie.

3.3.1 Stav spojenia

Spojenie sa od jeho začiatku až do konca prenášania dát nachádza v určitých stavoch ktoré je potrebné monitorovať a meniť. Zmenu stavu môžu vyvolať rôzne udalosti, ktoré sa odohrávajú predovšetkým na transportnej vrstve. Analyzátor využíva aj popis spojenia na základe architektúry klient/server a to tak, že je schopný určiť na základe IP adries, ktorý koncový bod spojenia je server a ktorý klient. Keďže zmeny stavu spojenia sa odohrávajú na transportnej vrstve, sú závislé na protokoloch TCP a UDP.

Protokol TCP je stavový, to znamená, že princíp jeho fungovania je založený na určitých stavoch. Pri nadväzovaní spojenia, ktoré sa nazýva *three-way handshake* a je zobrazené na obrázku 3.2, klient posiela serveru SYN paket. Podľa toho je analyzátor schopný určiť, na ktorej strane spojenia je klient a server. Zároveň sa mení stav daného spojenia na `TCP_SYN`. Server odpovedá paketom, ktorý má nadstavené flagy SYN a ACK, klient nakoniec zasiela serveru ACK paket a tým je spojenie nadviazané, mení sa aj jeho stav na `TCP_ESTABLISHED`. Nasleduje vlastný prenos dát a spojenie je v stave `TCP_ACTIVE`. Z tohto stavu sa po prenose dát spojenie zvyčajne dostáva do stavu `TCP_CLOSED`, pri detekcii paketu s nadstaveným flagom FIN. Analyzátor obsahuje ešte ďalšie stavy špecifické pre TCP spojenie, pre názornosť však boli uvedené len tie základné.

Protokol UDP je bezstavový, ale pre potreby analyzátora bolo nevyhnutné vytvoriť stavy aj pre tento protokol. UDP je vo svojej podstate od TCP jednoduchší, preto neobsahuje ani veľký počet stavov. Prvý paket v spojení určuje, ktorá strana je klient a server. Klientom je vždy tá strana, ktorá pošle prvý paket. Ak je počítač, na ktorom prebieha analýza, klientom v UDP spojení, stav spojenia sa mení na `UDP_REQUEST_SENT`, v opačnom prípade na `UDP_REQUEST_RECV`. Ak príde zo strany servera odpoveď, stav sa mení na `UDP_ACTIVE`. V UDP spojení je problém zistiť, kedy komunikácia skončila, preto je potrebné využiť

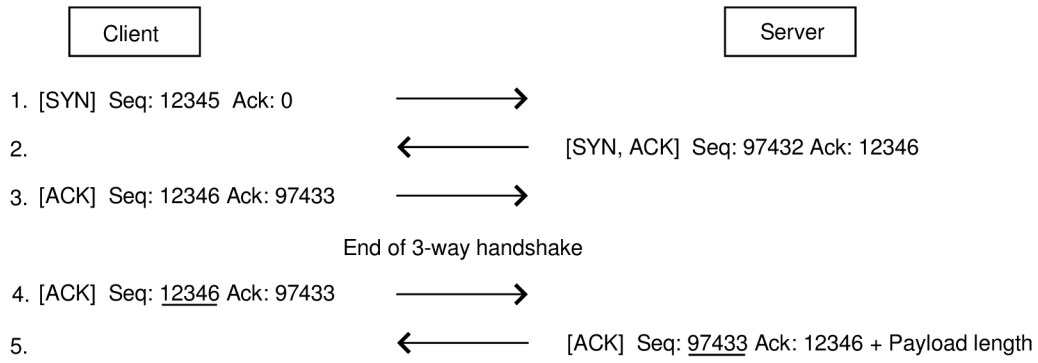
časovač, ktorý zisťuje, či je daný dátový tok aktívny podľa času posledného odchyteného paketu v danom spojení.

3.3.2 Buffer pre payload

Pre analýzu payloadu paketu je z pohľadu analyzátora zaujímavý prvý kilobajt prenášaných dát v spojení. Niekedy je postačujúce menšie množstvo, ale to je závislé na jednotlivých aplikačných protokoloch. Táto počiatočná komunikácia obsahuje spomínané signatúry, podľa ktorých sa dá detekovať aplikačný protokol. V architektúre klient/server sú dve strany spojenia, preto aj analyzátor obsahuje dva typy bufferov, jeden pre odchytávanie komunikácia od klienta a druhý od servera.

V TCP spojení je možné určiť na základe sekvenčného a potvrdzovacieho čísla, či odchytené dáta majú byť uložené do bufferu. Ten pri nadväzovaní spojenia určuje, ktoré sekvenčné čísla budú predstavovať počiatočný dátový tok. Tento princíp je uvedený na obrázku 3.2. Buffer v TCP spojení je ďalej schopný detekovať na akú pozíciu v buffery má byť odchytená komunikácia uložená. To vykonáva tiež na základe sekvenčného čísla daného paketu. Pri nadväzovaní spojenia sú kľúčové prvé dva pakety. SYN paket z klientskej strany určuje, aké sekvenčné číslo bude zodpovedať začiatku odchytených dát v klientskom buffery. Odpoveďou na SYN paket je zvyčajne paket zo strany servera, ktorý má nadstavené flagy SYN a ACK. Z tohto paketu je možné určiť sekvenčné čísla, ktoré predstavujú začiatky klientského rovnako aj serverového buffera.

Sekvenčné, ako aj potvrdzovacie číslo sú 32 bitové a pri prekročení ich rozsahu dochádza k pretečeniu, to znamená, že sa začína počítať znova od počiatočnej hodnoty. Buffer musí riešiť aj tento problém, preto využíva funkciu [12], ktorá určuje, či dané sekvenčné číslo patrí do určitého intervalu, pričom zohľadňuje aj pretečenie.



Obrázok 3.2: Nadväzovanie spojenia pomocou protokolu TCP

Naopak v UDP spojení sa sekvenčné a potvrdzovacie čísla nepoužívajú a nie je preto možné kontrolovať, či dáta prichádzajú v správnom poradí alebo či nedochádza k stratám paketov. Do buffera sa preto ukladá odchytená komunikácia v takom poradí, v akom prichádza, až kým sa buffer nezaplní.

Odchytená komunikácia, ktorá sa ukladá do buffera, môže obsahovať aj znak konca reťazca `\0`, to je však nežiadúce z pohľadu analýzy payloadu spôsobom, ktorý je popísaný neskôr. Tento znak je preto v TCP buffery nahradený znakom `0` a v UDP buffery sa `\0` vôbec neukladá, nachádza sa iba na konci buffera. Existencia dvoch bufferov je potrebná,

pretože signatúry môžu byť obsiahnuté buď v dátach zasielaných klientom alebo serverom alebo v komunikácii oboch zúčastnených strán.

Ak je proces analýzy komunikácie spustený v čase, kedy už existuje jedno alebo viac spojení, analýza takýchto tokov nebude s najväčšou pravdepodobnosťou vykonaná vôbec alebo nebude možné z odchytených dát určiť typ protokolu aplikačnej vrstvy. Pri TCP spojení je potrebné poznať ktoré sekvenčné čísla zodpovedajú začiatku a koncu dátového buffera. Tieto čísla je možné získať len počas nadväzovania spojenia. Ak analýza začne prebiehať napríklad už pri prenose dát v danom toku, do buffera sa nebudú ukladať žiadne odchytené dáta a analýza payloadu sa nevykoná. Pri UDP spojení neprítomnosť sekvenčných čísel spôsobí, že komunikácia sa bude do buffera ukladať, ale analýza payloadu bude neúspešná, pretože je potrebné odchytiť dáta zo začiatku dátového toku.

3.3.3 Časovač

Spojenie v protokole TCP je možné ukončiť viacerými spôsobmi. Prvým je detekcia paketu obsahujúceho FIN flag. Ak klient žiada nadviazanie spojenia, server jeho požiadavku nemusí prijať a posiela mu RST paket. Jedná sa o odmietnutie alebo reset spojenia zo strany servera a tým pádom aj o jeho ukončenie.

Ak je spojenie určitý čas neaktívne, tj. od odchytenia posledného paketu v danom spojení uplynul určitý čas, spojenie by sa malo stať z pohľadu analyzátora ukončeným. Na tento účel slúži časovač, ktorý kontroluje aktívne spojenia a porovnáva aktuálny čas s časom odchytenia posledného paketu. Ak tento čas prekročí určitú predom stanovenú hranicu, spojenie sa stáva ukončeným. V takomto prípade sa spojenie dostáva do stavu `TIME_TERM`. Pre protokol UDP je to aj jediný spôsob, akým je možné ukončiť spojenie.

Vypršanie časovača spúšťa mechanizmus kontroly aktívnych spojení. Táto kontrola však zaťažuje procesor, preto by sa mohla analýza čiastočne spomaliť, či zastaviť a po kontrole spojení znova pokračovať. Z tohto dôvodu je potrebné rozložiť kontrolu spojení do viacerých častí a tým skrátiť čas jej vykonávania. V analyzátore je kontrola rozdelená na dve časti. Najskôr sa kontrolujú TCP spojenia a po uplynutí určitého času UDP spojenia. Je to cyklus, ktorý sa stále opakuje.

3.4 Analýza payloadu

Analyzátor využíva pre popis signatúr jednotlivých aplikačným protokolov regulárne výrazy, ktoré je možné prevziať z linuxového *L7-filtra* [22]. Regulárny výraz je reťazec zvyčajne popisujúci celú množinu reťazcov. Najčastejšie sa používajú na vyhľadávanie vzorov v texte, tzv. *pattern matching* a rovnakú úlohu majú aj v navrhnutom systéme. Samotný regulárny výraz sa skladá z literálov textu, ktoré sa majú zhodovať a špeciálnych znakov, ktoré nie sú súčasťou hľadaného textu. Slúžia pre popis alternatív, množín, počtov výskytov a prepínačov.

Funkcie a štruktúry pre prácu s regulárnymi výrazmi sa nachádzajú v hlavičkovom súbore `regex.h` [19]. Regulárny výraz je potrebné pred jeho použitím preložiť pomocou funkcie `regcomp()`, ktorá vytvára z daného výrazu určitú štruktúru. Touto štruktúrou je deterministický konečný automat, ktorého popis sa nachádza na strane 10. Je jednou z možností efektívneho vyhľadávania vzorov v texte. Každý protokol je popísaný jedným regulárnym výrazom, takže aj jedným deterministickým konečným automatom. Regulárny výraz môže popisovať buď klientskú alebo serverovú časť komunikácie, to je dôvod, prečo

boli vytvorené dva typy bufferov. Pomocou funkcie `regexec()` sa potom vykonáva samotná analýza dát v buffery, ktorý musí byť ukončený znakom `\0`.

Súbor, v ktorom sa nachádzajú názvy jednotlivých protokolov a im prislúchajúce regulárne výrazy má nasledovný formát:

```
protocol_name; [t|u]; [port_numbers]
regex
```

`protocol_name` je názov aplikačného protokolu, písmeno `t` alebo `u` určuje, či daný protokol využíva služby TCP alebo UDP. `port_numbers` definuje, na ktorom porte alebo portoch je detekcia daného protokolu očakávaná. Tento údaj je voliteľný. Položka `regex` predstavuje samotný regulárny výraz popisujúci daný protokol. Mená protokolov spolu s preloženými regulárnymi výrazmi sú uložené do dvoch dynamických polí, ktoré sú rozlíšené podľa typu protokolu na transportnej vrstve (TCP alebo UDP). Vytvorenie dvoch polí má svoj význam, pretože sa tým skracuje čas potrebný na určenie aplikačného protokolu.

Veľmi dôležitou časťou pri detekcii protokolu je určiť, pre ktorý protokol sa má vykonať analýza ako prvá. Rýchlosť analýzy v tomto bode ovplyvňuje priepustnosť celého systému. Analyzátor používa rôzne spôsoby pre skrátenie času potrebného na identifikáciu protokolu. Tieto spôsoby sú popísané v nasledujúcich bodoch:

- **Využitie čísel portov** – do súboru, v ktorom sa nachádzajú názvy aplikačných protokolov spolu s príslušnými regulárnymi výrazmi, je možné pridať predpokladané číslo alebo čísla portov, na ktorých sa očakáva výskyt daného protokolu. Toto je veľmi dôležitá vlastnosť navrhnutého analyzátora, pretože významе urýchľuje jeho činnosť. Podľa výskumov [5] zhruba pre 93% odchytenej komunikácie (závisí na použitých aplikáciách) je možné správne určiť aplikačný protokol na základe čísla portu. Čísla portov sa nachádzajú v dátovej štruktúre nazývanej binárny vyhľadávací strom. Analyzátor obsahuje dva takéto stromy, jeden je určený na vyhľadávanie čísel portov pre TCP spojenia a druhý pre UDP spojenia. V určitom spojení sa číslo portu určuje z hlavičky protokolu transportnej vrstvy. Ak je počítač klientom a odchytený paket pochádza zo servera, číslom portu je potom zdrojový port.

- **Určenie poradia protokolov** – je spôsob, ktorý sa využíva v prípade, že analýza na základe čísla portu bola neúspešná. Analýza pokračuje zisťovaním, či odchytený payload obsahuje signatúry ostatných aplikačných protokolov v poradí, v akom sú uvedené v danom súbore.

- **Rozdelenie protokolov do dvoch skupín** – urýchľuje analýzu, ak sa nepodarilo zistiť aplikačný protokol využitím čísla portu. Regulárne výrazy sú rozdelené do dvoch skupín, TCP a UDP. Každá z nich obsahuje popis určitého počtu protokolov. Ak sa dáta prenášajú pomocou protokolu TCP, potom sa prehľadáva iba táto skupina aplikačných protokolov. Rovnako je to aj s aplikačnými protokolmi využívajúcimi UDP. Ak by sa tieto protokoly nerozdelili do dvoch skupín, všetky by sa nachádzali iba v jednej, potom by sa počet vyhľadávaní pomocou regulárnych výrazov zvýšil. To by malo za následok spomalenie celej analýzy.

Analýza payloadu prebieha vo dvoch fázach. Vo väčšine prípadov je možné určiť aplikačný protokol na základe prvých pár desiatok bajtov. Analýza payloadu preto prebieha už pri odchytení prvého paketu obsahujúceho dáta. Napríklad u protokolu HTTP, klient po nadviazaný spojenia zasiela serveru požiadavku, ktorá je uložená do klientského buffera

a vykoná sa jeho analýza. Ak sa nepodarí určiť aplikačný protokol, analyzátor pokračuje v ukladaní komunikácie do buffera. Na požiadavku server odpovedá, odpoveď je uložená do buffera určeného pre odchyt komunikácie zo strany servera. Následne je vykonaná aj analýza týchto dát. Ak sa ani teraz nepodarilo určiť aplikačný protokol, čaká sa na zaplnenie bufferov. V prípade zaplnenia buffera prichádza druhá fáza analýzy, ktorá je v podstate rovnaká ako fáza prvá, ale je vykonávaná pri zaplnenom buffery. Ten obsahuje dlhšiu vzorku odchytenej komunikácie v porovnaní s prvou fázou. Môže sa stať, že už dáta z prvého paketu zaplnia celý buffer, v takom prípade je už analýza ukončená a do druhej fázy sa nedostane.

Ak sa podarí určiť aplikačný protokol, ďalšia analýza takéhoto dátového toku už neprebieha, bola by zbytočná a zaťažovala by procesor. Po úspešnej detekcii sa uvoľní pamäť, ktorú využívali buffery. V tomto prípade sa monitoruje už iba stav spojenia, popísaný na strane 15. V prípade neúspešného určenia aplikačného protokolu aj po druhej fáze analýzy sa ďalšia analýza payloadu nevykonáva. To znamená, že nie je možné určiť typ protokolu, pretože analyzátor neobsahuje regulárny výraz popisujúci daný protokol alebo tento protokol je obtiažne popísať regulárnym výrazom, ktorý navyše nefunguje spoľahlivo. V takomto prípade sa postupuje ako keby bol aplikačný protokol úspešne detekovaný (uvoľnenie bufferov, monitorovanie stavu spojenia).

Kapitola 4

Výsledky a hodnotenie navrhnutého systému

V tejto kapitole sa nachádzajú rôzne testy, rozdelené do viacerých kategórií. Pri hodnotení navrhnutého systému je najdôležitejším kritériom čas, za ktorý sa vykoná analýza odchytených dát. Podľa testov je možné odhadnúť priepustnosť celého systému, určiť, ktoré časti analýzy sú výpočtovo a teda aj časovo najnáročnejšie. Všetky testy boli vykonané v sieti Ethernet s rýchlosťou prenosu dát 100 Mb/s. Namerané časy sú závislé aj na použitom hardware, hlavne na výkone procesora. Testy boli uskutočnené na počítači s procesorom Intel Pentium Centrino 1,83 GHz, 512 MB RAM.

4.1 Meranie času

Princíp všetkých testov je založený na meraní času, za ktorý sa vykoná určitá časť programu. Na tento účel je vhodné použiť hlavičkový súbor `time.h` [20], v ktorom sa nachádzajú prototypy funkcií a definície štruktúr určených pre prácu s časom. Definícia štruktúry na ukladanie času použitá pri testoch je nasledovná:

```
struct timespec
{
   time_t tv_sec;                   /* seconds */
   long tv_nsec;                    /* nanoseconds */
};
```

Pomocou funkcie `clock_gettime()` sa do štruktúry uloží aktuálny čas. Pri testoch bol meraný reálny čas potrebný na analýzu, z neho sa dá neskôr vypočítať skutočná priepustnosť systému. Čas sa pri teste ukladá na dvoch miestach programu (na začiatku a na konci meraného úseku). Kód popisujúci meranie času pri testoch:

```
struct timeval start;
struct timeval end;

clock_gettime(CLOCK_REALTIME, &start);
do_something();
clock_gettime(CLOCK_REALTIME, &end);
```

4.2 Testy a ich výsledky

Jednotlivé testy a ich výsledky sú popísané v nasledujúcich bodoch:

- **Dekódovanie hlavičiek** – pri tomto teste sa meria doba, za ktorú prebehne dekódovanie všetkých hlavičiek protokolov na jednotlivých vrstvách. Začiatok merania začína pri dekódovaní Ethernetového rámca a končí dekódovaním hlavičky paketu na transportnej vrstve. Táto fáza analýzy nie je výpočtovo najnáročnejšou časťou systému. Testy ukázali, že doba dekódovania hlavičiek protokolov na jednotlivých vrstvách sa pohybuje v rozmedzí 1 955 až 3 800 nanosekúnd. V konfiguračnom súbore `sniffer.conf` sa nachádzajú parametre, s ktorými je analyzátor spustený. Jedným z nich je aj možnosť zapnutia kontroly nazývanej cyklický redundantný súčet (CRC checksum), ktorý slúži na odhaľovanie chýb v prenášaných dátach. Kontrola sa vykonáva na sieťovej a transportnej vrstve. Výpočet kontrolného súčtu sa pri testovaní výraznejšie neprejavil, namerané časy sú totožné.

- **Vyhľadanie reťazca** – je test, v ktorom sa meria doba analýzy reťazca jedným regulárnym výrazom (na tento účel sa používa funkcia `regexec()` z hlavičkového súboru `regex.h` [19]). Inými slovami, je to čas, ktorý je potrebný na zistenie, či sa v buffery nachádza vzor, ktorý je popísaný regulárnym výrazom. Rôzne regulárne výrazy vytvárajú rozdielne deterministické konečné automaty, preto aj čas potrebný na prechod jednotlivými automatmi bude rozdielny. Z tohto dôvodu bol test vykonaný na väčšom množstve regulárnych výrazov, konkrétne sa jedná o regulárne výrazy popisujúce protokoly DHCP, DNS, FTP, HTTP a SSH. V tabuľke 4.1 sa nachádzajú časy namerané pri analýze payloadu, ktorý skutočne obsahoval signatúry pre daný protokol, čiže detekcia aplikačného protokolu bola úspešná.

protokol	test 1 [μs]	test 2 [μs]	test 3 [μs]	test 4 [μs]	test 5 [μs]	priemer [μs]
DHCP	20,114	11,454	12,292	12,013	12,292	13,633
DNS	53,638	66,489	65,651	62,857	45,536	58,834
FTP	19,276	52,800	24,864	52,521	49,727	39,438
HTTP	67,327	75,708	67,606	71,796	94,425	75,371
SSH	25,422	25,701	23,466	23,187	18,997	23,355

Tabuľka 4.1: Výsledné časy pri úspešnej detekcii protokolu

  Pre každý protokol bolo nameraných päť časov a vypočítaný ich priemer. Tieto časy závisia od miery zaplnenia buffera. Ešte väčší vplyv na čas má však regulárny výraz. Čím je zložitejší a obsahuje viac špeciálnych znakov, tým stúpa aj čas potrebný na analýzu. Toto potvrdzujú aj výsledky, z ktorých je podľa nameraných časov vidieť, že zložitejšie regulárne výrazy sú výpočtovo náročnejšie. V ďalšom teste boli namerané časy analýzy payloadu, ktorý neobsahuje signatúry pre daný protokol. Výsledky tejto analýzy sa nachádzajú v tabuľke 4.2. Z nameraných časov je zrejmé, že neúspešná detekcia protokolu je časovo náročnejšia, závisí hlavne od zložitosti regulárneho výrazu a veľkosti prehľadávaného payloadu.

- **Analýza bez použitia čísel portov** – je test, pri ktorom sa sekvenčne vyhľadáva regulárny výraz, ktorý popisuje dáta uložené v buffery. V tomto teste je do výsledných časov zahrnutá doba analýzy hlavičiek, zmena stavu spojenia, alokovanie pamäte

protokol	test 1 [μs]	test 2 [μs]	test 3 [μs]	test 4 [μs]	test 5 [μs]	priemer [μs]
DHCP	8,939	31,289	7,822	41,067	25,981	23,020
DNS	101,410	100,571	67,380	53,431	89,917	82,542
FTP	13,130	10,895	13,689	12,013	7,029	11,351
HTTP	100,292	1054,044	1060,750	62,019	1062,984	668,018
SSH	103,924	23,187	93,587	28,940	15,644	53,057

Tabuľka 4.2: Výsledné časy pri neúspešnej detekcii protokolu

pre buffer a nakopírovanie payloadu do bufferu. Je to test, na základe ktorého sa dá odhadnúť priepustnosť celého systému. Namerané časy sa nachádzajú v tabuľke 4.3 pre aplikačné protokoly využívajúce TCP spojenie. Jednotlivé regulárne výrazy sa v buffery vyhľadávajú v poradí, v akom sú uložené v súbore pre ne určenom. Pri tomto teste sa protokoly využívajúce TCP prehľadávajú v poradí: HTTP, FTP, SSH. Pretože aj pri tomto teste boli merané reálne časy, je vidieť, že pri analýze má veľ-

počet neúspechov	detekovaný protokol	čas analýzy [μs]
0	HTTP	87,720
0	HTTP	106,717
0	HTTP	98,337
1	FTP	167,619
1	FTP	132,698
1	FTP	151,974
2	SSH	93,029
2	SSH	89,117
2	SSH	94,305
3	-	1445,112

Tabuľka 4.3: Sekvenčná analýza vybraných protokolov

ký vplyv na rýchlosť aj aplikácia, ktorá je v čase analýzy spustená. Predpokladalo sa, že pri väčšom počte neúspechov bude stúpať aj doba analýzy. Pri protokoloch HTTP a FTP toto platí, počas analýzy aplikácia využívala procesor približne v rovnakej miere (bol použitý ten istý prehliadač). U protokolu SSH sa očakávalo, že čas analýzy bude väčší ako pri FTP alebo HTTP. Tento čas je ale približne rovnaký ako u protokolu HTTP, pretože aplikácia, ktorá bola počas analýzy spustená nie je tak výpočetne náročná ako tomu bolo u HTTP a FTP. Ak v payloade nebol detekovaný žiadny aplikačný protokol, čas analýzy by mal byť väčší, ale záleží aj v tomto prípade na použitej aplikácii.

- **Analýza s použitím čísel portov** – je to v podstate rovnaký test ako predošlý, rozdiel je len v poradí, v akom sa budú vyhľadávať regulárne výrazy v buffery. Do výsledných časov, ktoré sa nachádzajú v tabuľke 4.4 je zahrnutá doba dekódovania hlavičiek, práca s bufferom a prípadná zmena stavu spojenia.

Namerané časy ukazujú, že čísla portov podstatne urýchľujú analýzu. Ak sa podarí protokol detekovať na prvý pokus, čas závisí od veľkosti uložených dát v buffery, zložitosti

počet neúspechov	detekovaný protokol	čas analýzy [μs]
0	HTTP	77,048
0	HTTP	80,458
0	HTTP	87,162
0	FTP	31,009
0	FTP	64,254
0	FTP	60,622
0	SSH	36,317
0	SSH	31,288
0	SSH	33,803

Tabuľka 4.4: Analýza vybraných protokolov využívajúca čísla portov

regulárneho výrazu a vyt'aženosti procesora inými procesmi počas analýzy. Ak je analýza na základe čísla portu neúspešná, pokračuje sa sekvenčným prehľadávaním bufferu ostatnými regulárnymi výrazmi, tak ako je popísané v predchádzajúcom teste. S väčším počtom regulárnych výrazov rastie doba analýzy lineárne.

4.3 Priepustnost' systému

Priepustnost' systému udáva aké množstvo dát je schopný analyzátor spracovat' za jednu sekundu. Toto číslo sa nedá jednoznačne určit', závisí na mnohých faktoroch, akými sú rýchlost' procesora, operačný systém a jeho nadstavenie, vyt'aženost' procesora počas analýzy inými procesmi, počet analyzovaných protokolov, atd'. Pri analýze je dôležité, aby nedochádzalo k stratám pri odchyte komunikácie z dôvodu pret'aženia analyzátora.

Pri odhade priepustnosti systému je potrebné určit' časy medzi odchytom dvoch za sebou idúcich rámcov. Medzi rámcami sa nachádzajú určité medzery tzv. *interFrameGap* [8], ktoré určujú začiatok nového rámca. Ich veľkost' je minimálne 96 bitov. U 100 Mb Ethernetu je doba prenosu jedného bitu $10\,ns$ ($\frac{1}{10^8}\,s$). Minimálna veľkost' Ethernetového rámca je 576 bitov a čas jeho prenosu sa dá vypočítat' nasledovne:

$$(576 + 96) * 10\,ns = 6\,720\,ns = 6,72\,\mu s$$

Maximálna veľkost' rámca je 12 208 bitov a pre výpočet času prenosu takéhoto objemu dát platí:

$$(12\,208 + 96) * 10\,ns = 123\,040\,ns = 123,040\,\mu s$$

Pre odhad priepustnosti systému je však treba poznat' jeho priemernú dĺžku, ktorá bola vypočítaná zo 100 000 odchytených rámcov. Priemerná hodnota dĺžky rámca je 6 101 bitov, pri pripočítaní 96 bitov je výsledok 6 197 bitov. Priemerný čas medzi odchytami rámcov je teda $61,97\,\mu s$. Filter, ktorý sa pri odchyte komunikácie používa, posiela do analyzátoru iba TCP a UDP pakety za predpokladu, že je nadstavený výrazom `tcp or udp`. To znamená, že doba medzi odchytenými rámcami sa môže ešte výrazne zväčšit' a tým zostáva viacej času na analýzu.

Pri testoch sa ukázalo, že analýza payloadu prebieha iba na dvanástich percentách z odchytenej komunikácie čo predstavuje asi každý ôsmy odchytený rámec. Pokiaľ je čas na analýzu okolo $61,97\,\mu s$ a podarí sa detekovat' aplikačný protokol použitím čísla portu na prvý pokus, potom nedochádza k stratám pri odchyte komunikácie. V prípade, že protokol

nie je detekovaný na základe čísla portu, čas analýzy sa môže dostať do rádov stoviek až tisícok mikrosekúnd. V takomto prípade však ešte nemusí dochádzať k stratám, pretože operačný systém poskytuje paketový buffer a v ňom sa hromadia odchytené rámce, ktoré by inak boli stratené. Veľkosť tohto buffera je v Unixových systémoch väčšinou 256 KB, ale dá sa zväčšiť až na niekoľko MB. Nahromadené rámce sa z buffera potom dostávajú do analyzátora, v tomto momente je podstatné, aby sa znova neopakovala situácia, pri ktorej bude vykonaná neúspešná analýza na základe čísla portu, čo by už mohlo viesť k stratám pri odchyte komunikácie.

Z vykonaných testov sa dá odhadnúť priepustnosť navrhnutého systému, ktorá je okolo 100 Mb/s. Tá závisí hlavne na použitom hardware, operačnom systéme a jeho konfigurácii, počte detekovaných protokolov a vyťaženosti systému pri analýze inými procesmi.

Kapitola 5

Záver

Bol vytvorený systém pre detekciu aplikačných protokolov na základe obsahu dátovej časti paketu. Systém bol implementovaný v jazyku C pre dosiahnutie čo možno najväčšej rýchlosti analýzy. Analyzátor je okrem rozpoznávania protokolov z payloadu paketu schopný dekódovať hlavičky odchytených dát na jednotlivých vrstvách sieťového modelu TCP/IP, konkrétne sa jedná o Ethernet, protokol IPv4 a na transportnej vrstve protokoly TCP a UDP.

Pre popis aplikačných protokolov systém používa regulárne výrazy, ktoré je možné čerpať z *L7-filtra* [22]. Analýzou payloadu je možné rozpoznať aj protokoly využívajúce dynamické prideľovanie čísel portov (napríklad P2P aplikácie). Štandardné porty môžu byť využívané aj inými aplikáciami, ako je očakávané. Je to snaha o znemožnenie odhalenia takejto komunikácie a jej prípadného blokovania. Zariadenia ako firewally alebo analyzátory, ktoré rozpoznávajú aplikačné protokoly na základe čísel portov nedokážu takúto komunikáciu identifikovať. Navrhnutý systém ale rozpoznáva, že na štandardnom porte sa nemusí nachádzať očakávaný protokol a je schopný detekovať, o aký protokol sa jedná.

Výsledky testov ukázali, že priepustnosť systému je okolo 100 Mb/s. Limitujúcou časťou je podľa nameraných časov vyhľadávanie vzorov v payloade paketu pomocou regulárnych výrazov. Pre urýchlenie vyhľadávania boli použité čísla portov, ktoré určujú protokoly, pre ktoré sa vykoná analýza ako prvá. Tým sa celková priepustnosť systému značne zvyšuje, ale nastáva problém pri neúspešnej detekcii protokolu pomocou čísla portu, ako je popísané v predchádzajúcej kapitole. Z pohľadu ďalšieho vývoja je teda potrebné skrátiť dobu vyhľadávania vzorov v payloade paketu čím sa dosiahne aj väčšia priepustnosť. Riešením by mohlo byť vytvorenie jedného deterministického konečného automatu, ktorý by obsahoval regulárne výrazy všetkých detekovaných protokolov. Pri analýze payloadu by bol potrebný iba jeden prechod takýmto konečným automatom čo by zjednodušilo analýzu a nebolo by potrebné spoliehať sa na číslo portu. Rovnako je potrebné pridať podporu pre ďalšie protokoly na prvej až tretej vrstve sieťového modelu TCP/IP, jedná sa hlavne o protokoly PPP, ARP, IPv6 a ICMP, čím sa zvýši miera použiteľnosti vytvorenej aplikácie.

Literatúra

[1] Aho, A.V., Corasick, M.J.: Efficient String Matching: An Aid to Bibliographic Search. In: Communications of the ACM, vol. 18, no 6., pp. 333-340, 1975.

[2] Boyer, R.S., Moore, J.S.: A fast string searching algorithm. In: Communications of the ACM, vol. 20, no. 10, pp. 762-772, 1977.

[3] Cormen, T.H., aj.: Introduction to algorithms. MIT Press and McGraw-Hill. ISBN 0-262-03141-8, 2001

[4] Deering, S., Hinden, R.: Internet protocol version 6 (IPv6) specification, RFC 2460, 1998. Dokument dostupný na `http://www.ietf.org/rfc/rfc2460.txt`, (máj 2008).

[5] Dreger, H., Feldmann, A., Mai, M., Paxson, V., Sommer, R.: Dynamic application-layer protocol analysis for network intrusion detection. Dokument dostupný na `http://www.icir.org/robin/papers/usenix06/`, (máj 2008).

[6] IANA. Port numbers. Dokument dostupný na `http://www.iana.org/assignments/port-numbers`, (máj 2008).

[7] IEEE, 3 Park Avenue, New York, NY 10016-5997, USA. Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications. Std. 802.3 edition, 2005. ISBN 0-7381-4741-9.

[8] Karlin, S., Peterson, L.: Maximum packet rates for full-duplex ethernet, technical report. Dokument dostupný na `ftp://ftp.cs.princeton.edu/techreports/2002/645.pdf`, (máj 2008).

[9] Karp, R.M., Rabin, M.O.: Efficient randomized pattern-matching algorithms. In: IBM Journal of Research and Development, vol. 31, no. 2, pp. 249-260, 1987.

[10] Knuth, D.E., Morris, J., Pratt, V.R.: Fast pattern matching in strings. In: SIAM Journal of Computing, vol. 6, no. 2, pp. 323-350, 1977.

[11] McCanne, S., Jacobson, V.: The bsd packet filter: A new architecture for user-level packet capture, 1992. Dokument dostupný na `http://www.tcpdump.org/papers/bpf-usenix93.pdf`, (máj 2008).

[12] Plummer, W.: Sequence number arithmetics, 1978. Dokument dostupný na `http://www.networksorcery.com/enp/ien/ien74.txt`, (máj 2008).

[13] Postel, J.: Internet protocol, RFC 791, 1981. Dokument dostupný na `http://www.ietf.org/rfc/rfc0791.txt`, (máj 2008).

[14] Postel, J.: Transmission control protocol, RFC 793, 1981. Dokument dostupný na http://www.ietf.org/rfc/rfc0793.txt, (máj 2008).

[15] Postel, J.: User datagram protocol, RFC 768, 1980. Dokument dostupný na http://www.ietf.org/rfc/rfc0768.txt, (máj 2008).

[16] Sourcefire: Snort: The open source network intrusion detection system. http://www.snort.org, (máj 2008).

[17] WWW stránky. Finite State Automata. http://www.cs.princeton.edu/introcs/73fsa/, (máj 2008)

[18] WWW stránky. Nondeterministic Finite Automata. http://planning.cs.uiuc.edu/node558.html, (máj 2008)

[19] WWW stránky. Manuálová stránka pre regex.h. http://linux.die.net/man/3/regex, (máj 2008).

[20] WWW stránky. Manuálová stránka pre time.h. http://www.opengroup.org/onlinepubs/007908799/xsh/time.h.html, (máj 2008).

[21] WWW stránky projektu Bro. http://www.bro-ids.org, (máj 2008).

[22] WWW stránky projektu L7-filter. http://www.l7-filter.sourceforge.net, (máj 2008).

[23] WWW stránky projektu Libpcap. http://www.tcpdump.org, (máj 2008).

[24] WWW stránky projektu Wireshark. http://www.wireshark.org, (máj 2008).

Dodatok A

Manuál k programu

Program sa preloží príkazom `make` v adresári `/src`, kde sa nachádza súbor `Makefile`. Po preklade sa vytvorí program s názvom `sniffer` v nadradenom adresári. Pri preklade je možné použit' parameter `DEBUG`, ktorý spôsobí, že výsledný program bude vypisovat' pri odchyte komunikácie ladiace informácie.

Parametre

V súbore `sniffer.conf` sa nachádzajú parametre programu, s ktorými je spustený. Popis jednotlivých parametrov:

`interface` – názov zariadenia, na ktorom bude prebiehat' analýza odchytenej komunikácie. Ak je tento parameter nadstavený na `auto`, program sa sám pokúsi vyhl'adat' správne zariadenie.

`bpf_filter` – určuje výraz, podl'a ktorého bude prebiehat' filtrácia odchytenej komunikácie pomocou BSD packet filtra.

`regex_file` – názov súboru, v ktorom sa nachádzajú názvy protokolov a regulárne výrazy

`checksum` – tento parameter zapína/vypína kontrolu CRC checksum na tretej a štvrtej vrstve modelu ISO/OSI. Jeho hodnoty sú `yes` alebo `no`.

`hash_table_size` – udáva veľkost' hashovacej tabuľky (veľkost' by mala byt' prvočíslom).

`timer_lim` – čas v sekundách, po ktorom sa stáva spojenie neaktívnym.

`timer_interval` – čas v sekundách, ktorý udáva, ako často sa má vykonávat' kontrola na neaktívne spojenia.

`print_l2`, `print_l3`, `print_l4` – spôsob výpisu informácii o odchytených dátach. Číslo udáva, o akú vrstvu sa jedná vzhl'adom na model ISO/OSI. Parameter môže mat' hodnoty `no`, `yes` alebo `detail`.

`print_l7_length` – určuje, či sa má vypisovat' veľkost' payloadu. Hodnota parametra je `no` alebo `yes`.

`print_l7` – môže mat' hodnoty `yes` alebo `no` a určuje, či sa bude vypisovat' payload paketu.

`print_status` – určuje, či sa má vypisovať stav spojenia. Parameter môže mať hodnotu `no` alebo `yes`.

TCP spojenie sa môže dostať do rôznych stavov, podľa detekcie TCP flagov a prenášaného payloadu:

`inactive` – žiadny paket obsahujúci payload, SYN, FIN alebo RST flag

`syn` – iba SYN paket od klienta

`syn_ack` – iba paket so SYN a zároveň ACK flagom od servera bez SYN paketu od klienta

`partial` – iba paket s payloadom

`parial_s` – SYN paket a paketom s payloadom

`partial_sa` – SYN, ACK paket od servera a paket s payloadom, bez SYN paketu od klienta

`establish` – SYN a SYN, ACK paket, bez paketu s payloadom

`active` – SYN paket spolu so SYN, ACK paketom a paketom s payloadom

`client_f` – FIN paket od klienta

`server_f` – FIN paket od servera

`last_a` – odpoveď na FIN flag, čakanie na posledný ACK paket

`closed` – spojenie ukončené na základe FIN paketu

`reset` – RST paket

`time_term` – spojenie ukončené po určitej dobe neaktivity

V UDP spojení sú definované stavy:

`inactive` – v spojení nebol detekovaný žiadny paket

`request_s` – klient poslal požiadavku na server

`request_r` – server odchytil požiadavku od klienta

`active` – server odpovedal na požiadavku

`time_term` – spojenie ukončené po určitej dobe neaktivity

Súbor s protokolmi

V tomto súbore sa nachádzajú informácie o jednotlivých protokoloch, ktoré je analyzátor schopný identifikovať. Tento súbor má nasledovný formát:

```
protocol_name; [t|u]; port_numbers
regex
```

Popis jednotlivých položiek:

`protocol_name` – je názov protokolu aplikačnej vrstvy

`t` alebo `u` – určujú, či sa jedná o TCP alebo UDP spojenie

`port_numbers` – číslo alebo čísla portov

`regex` – regulárny výraz popisujúci aplikačný protokol

Všetky položky sú povinné okrem `port_numbers`.

Výstup programu

Počas odchytávania komunikácie program vypisuje informácie o každom odchytenom rámci. Formát výpisu je závislý od parametrov nachádzajúcich sa v súbore `sniffer.conf`. Ak je detekovaný aplikačný protokol, program vždy túto informáciu vypíše spolu s názvom daného protokolu.

Ak je analýza ukončená, program vypíše informácie o všetkých spojeniach, ktoré boli počas analýzy identifikované. Takýto výpis má formát:

```
IP_addresses  ports  transport_layer  status  pkt_count  protocol
```

Popis jednotlivých položiek:

`IP_addresses` – IP adresy koncových zariadení, medzi ktorými prebiehala komunikácia

`ports` – čísla portov aplikácií, medzi ktorými prebiehala komunikácia

`transport_layer` – protokol na transportnej vrstve (TCP alebo UDP)

`status` – stav, v ktorom sa spojenie nachádzalo v čase ukončenia analýzy

`pkt_count` – celkový počet odchytených paketov v spojení (od klienta aj servera)

`protocol` – názov aplikačného protokolu

Ďalej program vypisuje štatistiky týkajúce sa počtov spojení a odchytených paketov. Výpis ma formát:

`UDP connections:` – počet UDP spojení

`TCP connections:` – počet TCP spojení

`Connections total:` – celkový počet spojení (TCP + UDP)

`UDP packets:` – počet odchytených UDP paketov

`TCP packets:` – počet odchytených TCP paketov

`Packets total:` – celkový počet odchytených paketov (TCP + UDP)

Libpcap poskytuje štatistiky týkajúce sa počtu paketov od spustenia po ukončenie analýzy. Položka `Dropped:` predstavuje počet paketov, ktoré boli analyzátorom odmietnuté, najčastejšie z dôvodu jeho pret'aženia.

Ukončenie analýzy

Program analýzu ukončí pri odchytením signálu `SIGINT` (`Ctrl + C`) alebo `SIGTERM` (`kill PID`).