VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

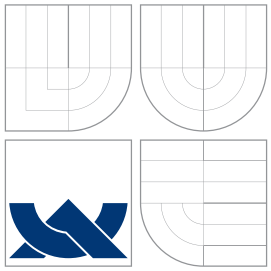
ANALYZÁTOR SÍŤOVÝCH PROTOKOLŮ

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

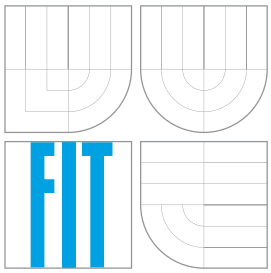
AUTOR PRÁCE
AUTHOR

DANIEL PŠORN

BRNO 2009



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 — DANIEL PŠORN
AUTHOR

VEDOUCÍ PRÁCE — Ing. JIŘÍ TOBOLA
SUPERVISOR

BRNO 2009

Abstrakt

Cílem této práce je nalézt způsob, jak naprogramovat analyzátor síťového provozu na nejvyšší vrstvě modelu ISO/OSI. K tomu je nutné použít některou z metod detekce aplikačních protokolů. Program popsaný v této práci používá metodu založenou na číslech portů a metodu založenou na signaturách. Cílová platforma je operační systém FreeBSD. Implementačním jazykem je jazyk C s použitím knihovny libpcap.

Klíčová slova

Síťová komunikace, Model ISO/OSI, FreeBSD, BPF, libpcap

Abstract

Object of this thesis is to find the way how to program network protocol analyzer on the highest level of ISO/OSI model. We need to use some of the methods for detection of application protocols. The software described in this thesis uses traditional application-level traffic identification method and signature-mapping-based method. Basic platform is FreeBSD operating system. Programming language is C using libpcap library.

Keywords

Network communication, Model ISO/OSI, FreeBSD, BPF, libpcap

Citace

Daniel Pšorn: Analyzátor síťových protokolů, bakalářská práce, Brno, FIT VUT v Brně, 2009

Analyzátor síťových protokolů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jiřího Toboly. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Daniel Pšorn
23. ledna 2009

Obsah

Kapitola 1

Úvod

Počítačové sítě a telekomunikační technologie zažívají v posledních několika desetiletích bouřlivý rozvoj. Především úspěch Internetu přinesl síťové technologie do komerční sféry a do velkého množství domácností. Díky připojení do Internetu mohou uživatelé využívat mnoho služeb, od prohlížení webových stránek přes používání elektronické pošty, telefonování až po náročné multimediální aplikace jako je sledování videa. Abychom mohli všechny tyto služby využívat, potřebujeme spolehlivý chod počítačové sítě jako celku. Z toho důvodu vzniklo velké množství nástrojů na monitorování sítí. Popisem jednoho způsobu monitorování, analýzou paketů, se zabíva tato práce.

Při návrhu analyzátoru síťových protokolů je nůtné důkladně se seznámit s principy fungování počítačové sítě. Pro komunikaci mezi sebou používají zařízení síťové protokoly. Síťový protokol je norma definovaná na papíře. Jako referenční norma je používána norma ISO/OSI, která je popsána v druhé kapitole. Síť Internet normu ISO/OSI nepoužívá, používá síťové protokoly rodiny TCP/IP.

Samotná technologie odchytávání paketů je představena ve třetí kapitole. Nejprve jsou rozebrány principy, jaké je možné použít k odchytávání dat. Poté následuje popis zařízení bpf (Berkeley Packet Filter), jejž bude využt k odchytávání paketů. Nakonec je popsáno programové vybavení (API), které je možné použít na realizaci síťového analyzátoru. Na operačních systémech unixového typu je to knihovna libpcap a na operačních systémech rodiny MS Windows je to knihovna winpcap.

Čtvrtá kapitola popisuje samotný návrh a implementaci analyzátoru síťových paketů na aplikační vrstvě. Je zde popsána jak metoda analyzování na základě čísel portů, tak i metoda založená na signaturách.

Následující kapitola shrnuje výsledky práce a testování, případně porovnává s jinými podobnými aplikacemi.

Poslední kapitola je závěr, ve kterém je shrnuto hodnocení implementovaného systému. Dále jsou zde navržená různá zlepšení, zejména co se týče rychlosti detekování aplikačních protokolů.

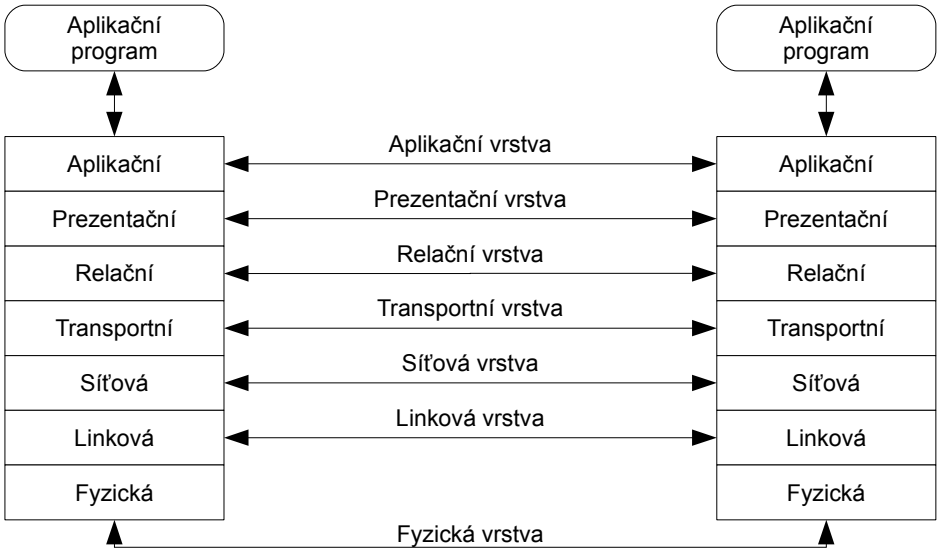
Kapitola 2

Síťové protokoly

Komunikace mezi počítači je složitá věc. Abychom tedy mohli porozumět analýze paketů, je důležité pochopit principy komunikace počítačů navzájem. V této kapitole se zaměřím na popis referenčního modelu ISO/OSI, architekturu TCP/IP a přenos dat.

2.1 Model ISO/OSI

Referenční model ISO/OSI (Open Systems Interconnections) [7] byl vypracován organizací ISO (International Organization for Standardization) a v roce 1984 byl schválen jako mezinárodní norma ISO 7489. Model OSI nespecifikuje implementaci konkrétního protokolu, ale je abstraktním popisem sedmivrstvé síťové architektury. Každá vrstva má funkce, které využívají služeb nejbližší nižší vrstvy a poskytují své služby nejbližší vyšší vrstvě. Sousední vrstvy mezi sebou komunikují přes přístupový bod služby (Service Access Point). Komunikace mezi dvěma počítači je znázorněna na obrázku 2.1.



Obrázek 2.1: Komunikace na sedmivrstvé architektuře ISO/OSI

Následuje popis jednotlivých vrstev a jejich funkcí.

Aplikační vrstva
Aplikační vrstva, nejvyšší vrstva OSI modelu, poskytuje rozhraní aplikacím, pomocí kterého mohou přistupovat k prostředkům komunikačního systému. Jedná se o různé služby typu elektronické pošty, přenos souborů pomocí protokolu FTP nebo bezpečné přihlášení na vzdálený terminál (SSH).

Prezentační vrstva
Prezentační vrstva má za úkol transformovat příchozí data do vhodné podoby, které používá aplikační vrstva. Jedná se především o kódování dat (problém nejvyššího bitu v bajtu) nebo zabezpečení přenášené informace šifrováním, zabezpečení integrity dat aj.

Relační vrstva
Relační vrstva řídí sezení (dialog) mezi dvěma zařízeními, včetně zahájení a ukončení spojení.

Transportní vrstva
Hlavním cílem transportní vrstvy je poskytování spolehlivého spojení mezi procesy. Transportní vrstva se tedy nezabývá spojením mezi vzdálenými počítači, ale zajišťuje komunikaci mezi aplikacemi. Mezi dvěma počítači je tedy možné mít několik transportních spojení současně. Transportní vrstva zajišťuje jak spolehlivý přenos dat (řízení toku, potvrzování přijetí, pořadí paketů), tak i nespolehlivé spojení (bez přímého navázání spojení, nezajišťuje potvrzování). Adresování aplikací je jednoznačné v rámci jednoho zařízení.

Síťová vrstva
Síťová vrstva je zodpovědná za přenos dat mezi vzdálenými počítači. Stará se o vytváření síťových paketů a o jejich směrování při průchodu sítí. Síťová vrstva je také zodpovědná za jednoznačné adresování síťového rozhraní v celé síti.

Linková vrstva
Linková vrstva zajišťuje výměnu dat v rámci lokální sítě nebo mezi dvěma zařízeními. Základní jednotkou pro přenos dat je datový rámec, který se skládá ze záhlaví, přenášených dat a zápatí. V záhlaví se nachází linková adresa odesílatele a příjemce (MAC adresa) a řídící informace. V přenášených datech jsou uloženy informace pro vyšší vrstvy. Záhlaví obsahuje kontrolní součet, pomocí něhož lze zjistit, zda došlo k poškození dat při přenosu.

Fyzická vrstva
Fyzická vrstva je nejnižší vrstva OSI modelu. Definuje fyzikální a elektrické vlastnosti používané při komunikaci mezi zařízeními. Fyzická vrstva zahajuje a ukončuje spojení, popisuje signály na konektorech, určuje napěťové úrovně a specifikuje vlastnosti kabelů.

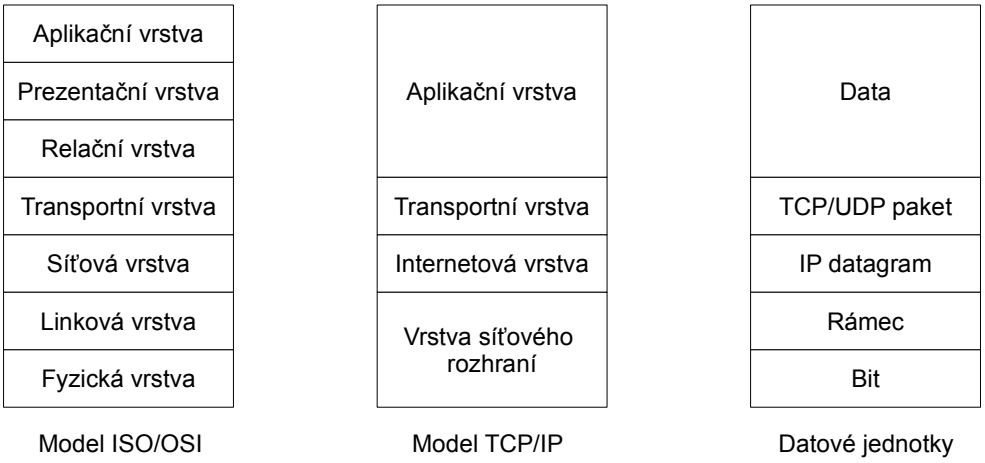
2.2 TCP/IP

Rodina protokolů TCP/IP je pojmenována po dvou důležitých protokolech, kterými jsou Transmission Control Protocol (TCP) a Internet Protocol (IP). V odborné literatuře je možne se setkat i s názvy Internet Protocol Suite (IPS).

Soubor protokolů TCP/IP vznikl na žádost agentury DARPA (Defense Advanced Research Projects Agency) pro potřeby válečného ohrožení. Jedná se o decentralizovaný, robustný systém, který je nezávislý na typu fyzického média - metalické vedení, optická

vlákna, bezdrátové technologie. Postupem času se z rodiny protokolů TCP/IP stal standard současného Internetu.

Protokoly TCP/IP, jako vetšina protokolů v síti Internet, jsou rozděleny do vrstev. Vrstvy jsou čtyři, nejnižší je vrstva síťového rozhraní, nasleduje internetová vrstva, transportní vrstva a nejvýš se nachází vrstva aplikační.



Obrázek 2.2: Porovnání modelů OSI a TCP

Aplikační vrstva
Aplikační vrstvu využívají programy, které potřebují k přenosu dat po síti protokoly TCP/IP. Tato vrstva obsahuje aplikace jako je HTTP, FTP, Telnet a další.

Transportní vrstva
Transportní vrstva poskytuje přenos dat mezi aplikacemi běžícími na vzdálených počítačích (vytváří tzv. logické spojení mezi procesy). Hlavními protokoly transportní vrstvy jsou TCP (Transmission Control Protocol) a UDP (User Datagram Protocol). Protokol TCP zajišťuje spolehlivý přenos dat, tzn. vytvoří na dobu spojení virtuální okruh mezi aplikacemi. Protokol UDP je na rozdíl od protokolu TCP nespojovaná služba, tj. nezajišťuje spolehlivé spojeni. Odesílatel pouze odešle UDP datagram a už se vůbec nestará, zda došel příjemci. Oba typy protokolů mají své výhodý (UPD rychlost, TCP spolehlivost) i nevýhody (UDP nespolehlivé spojení, TCP protokol je pomalejší než UDP).

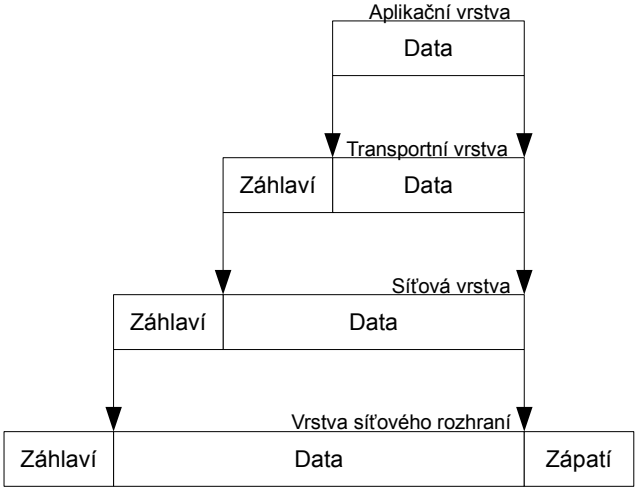
Internetová vrstva
Internetová vrstva (též nazývaná síťová vrstva) má za úkol dopravovat data mezi různými počítači v Internetu, tj. i přes mnohé LAN. Na této vrstvě je používaný protokol IP (Internet Protocol). Díky protokolu IP jsou data od odesílatele směrována na místo určení přes směrovače. Na cestě od odesílatele k příjemci se může vyskytovat několik směrovačů.

Vrstva síťového rozhraní
Nejnižší vrstva poskytuje přístup na přenosové médium. Rodina protokolů TCP/IP přesně nespecifikovala přístup na fyzické médium, je schopná používat různé přenosové technologie (Ethernet, Token Ring, FDDI, Frame Relay, RS-232 aj.).

2.3 Přenos dat

Při přenosu dat po síti se využívá datových jednotek (PDU - Protocol Data Unit), které mají na různých vrstvách jiné názvy. Nejmenší jednotkou informace je jeden bit, jenž slouží pro přenos dat na nejnižší, fyzické vrstě. Na druhé vrstě je rámec, následuje IP datagram, TCP/UPD paket a poslední jsou data (obr. 2.2) [7].

Přenos informace z jednoho zařízení na druhé využívá zapouzdření. Zapouzdření je proces, kdy dochází k přenosu dat z vyšší vrstvy do nižší. Tedy nějaký program chce poslat data na vzdálený počítač. K přenosu použije model TCP/IP. Pomocí sady pravidel aplikačního protokolu vytvoří zprávu a předá ji transportní vrstě. Transportní vrstva zprávu přijme a přidá k ní záhlaví. Zprávu zapouzdří (vznikne paket) a předá síťové vrstvě. Síťová vrstva paket přijme, přidá záhlaví, zprávu zapouzdří (vznikne datagram) a předá linkové vrstvě. Linková vrstva přidá záhlaví a zápatí, kde je umístěn kontrolní součet. Data zapouzdří a vznikne rámec, který předá fyzické vrstvě. Fyzická vrstva přemění rámec na posloupnost bitů a odešle je. Celý proces je znázorněn na obr. 2.3. Nyní následuje podrobný rozbor záhlaví jednotlivých vrstev.



Obrázek 2.3: Zapouzdření dat během přenosu

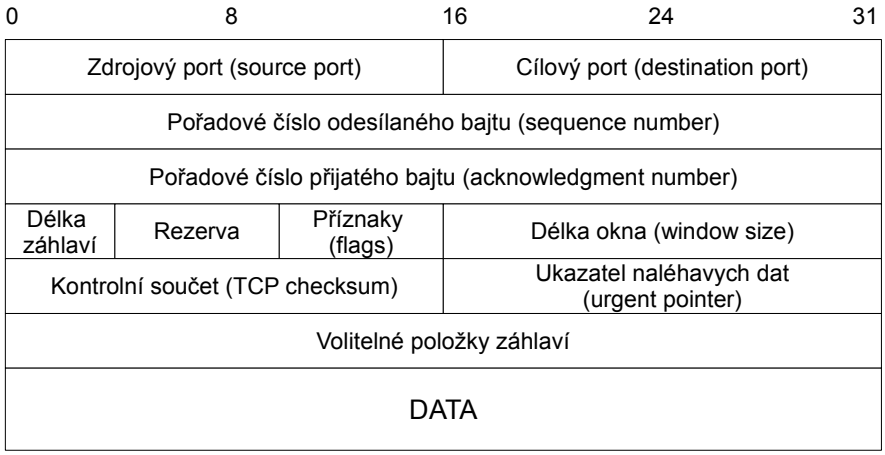
2.3.1 Aplikační vrstva

Aplikační vrstva poskytuje přístup procesům ke komunikačnímu systému. Poskytuje velké množství protokolů (ftp, http, xmpp aj.), které definují pravidla síťové komunikace a formáty datových struktur. Některé služby vyžadují buď protokol TCP nebo protokol UDP. Jiné služby jsou schopné využívat oba protokoly (záleží na implementaci či konfiguraci). Data mohou být přenášena jak ve formě binární tak i textové.

2.3.2 Transportní vrstva

Úkolem transportní vrstvy je vytvářet tzv. logické spojení mezi procesy. Mezi nejdůležitější protokoly transportní vrstvy patří TCP [17] a UDP [18]. Protokol TCP poskytuje spolehlivý přenos dat (doručí adresátovy data tak, jak byla odeslána). Navázání spojení je realizováno

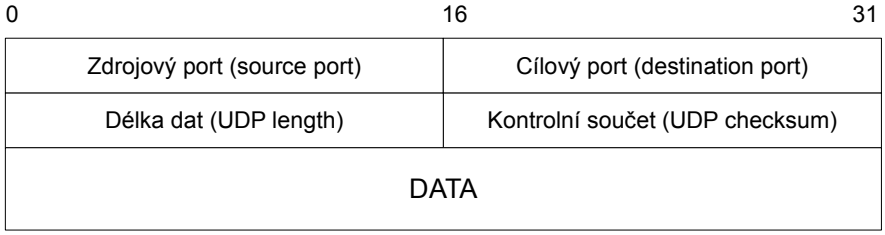
pomocí mechanizmu three-way handshake. Protokol TCP dokonce umožňuje, aby obě strany navazovaly spojení současně. TCP segment má strukturu znázorněnou na obr. 2.4.



Obrázek 2.4: Záhlaví TCP paketu

Protokol UDP je narozdíl od protokolu TCP nespolehlivou nespojovanou službou, tzn. nenavazuje spojení. Odesílatel odešle UDP datagram příjemci a už se nestará, zda příjemci skutečně přišel (o to se stará vyšší vrstva). Typicky se používá v aplikacích, které požadují jednoduchost a malou režii spojenou s přenosem. Záhlaví UDP datagramu je znázorněno na obr. 2.5

Adresování na transportní vrstvě je řešeno tzv. číslem portu, který jednoznačně identifikuje službu na daném zařízení. Číslo portu je dvoubajtové, takže může nabývat hodnot 0 až 65535. U čísel portů se též udává typ protokolu zapisovaný za lomítko. Pro protokol UDP je jiná sada protokolů než pro protokol TCP, tzn. port 53/tcp nemá nic společného s portem 53/udp. Seznam přidělených portů se na operačních systémech unixového typu nachází v souboru /etc/services.

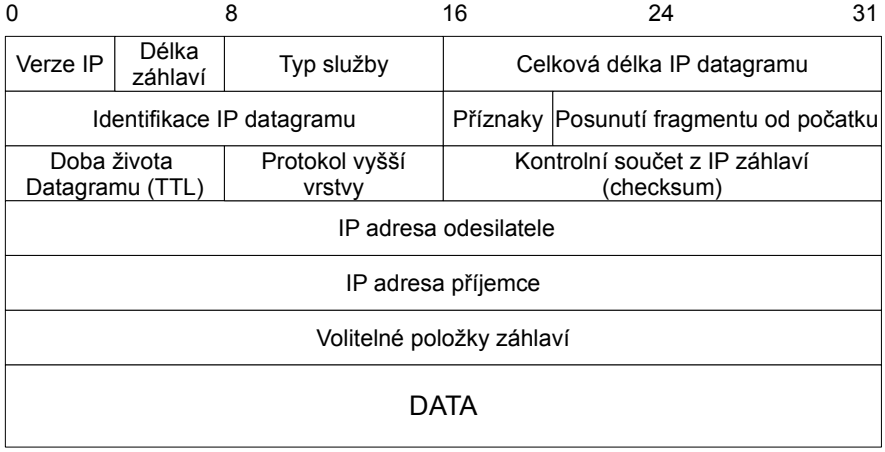


Obrázek 2.5: Záhlaví UDP paketu

2.3.3 Internetová vrstva

Internetová vrstva umožňuje propojit jednotlivé lokální sítě do celosvětového Internetu. Obsahuje jeden ze základních komunikačních protokolů, protokol IP (Internet Protocol [16]). IP protokol je tvořen několika dílčími protokoly a to jsou samotný protokol IP, služební

protokoly ICMP a IGMP a služební protokoly ARP a RARP, které jsou často vyčleňovány jako samostatné, protože jejich rámce neobsahují IP záhlaví. Struktura IP datagramu je na obr. 2.6.



Obrázek 2.6: Záhlaví IP datagramu

Adresování na internetové vrstvě je reprezentováno IP adresou, což je 32-bitové číslo oddělené tečkami. Např. 147.229.201.5 je platná IP adresa v síti Internet. IP adresa jednoznačně identifikuje síťové rozhraní systému (anglicky se jednoznačná adresa nazývá *unicast*). Pokud má systém (počítač) více síťových rozhraní, pak každé rozhraní má svojí IP adresu. Je také možné používat více IP adres na jednom síťovém rozhraní. Struktura adresy se skládá z části, která identifikuje síť, ve které se počítač nachází a z části identifikující počítač. IP adresy v Internetu přiděluje organizace Internet Assign Numbers Authority (IANA). V současné době se používá IP protokol verze 4 (IPv4) a verze 6 (IPv6). Nicméně IPv6 se zatím moc nerozšířil. Existence dvou verzí je z důvodu nedostatku adresního prostoru u IP verze 4.

Problematika adresace je velice složitá a vydala by na napsání samotné práce. Proto zájemce odkazuji na příslušnou literaturu [7].

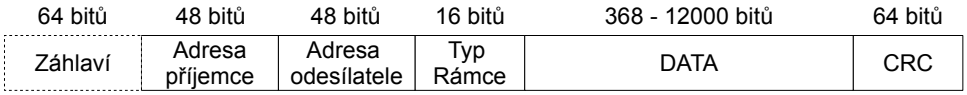
2.3.4 Vrstva síťového rozhraní

Rodina protokolů TCP/IP se nezabývá linkovou a fyzickou vrstvou. Proto je možné tyto protokoly provozovat na různých sítích typu WAN (Wide Area Network - rozsáhlé sítě) a LAN (Local Area Network - lokální sítě). Jelikož analyzátor síťového provozu popisovaný v této práci bude umístěn v lokální síti, je zde popsána pouze technologie Ethernet.

Počítačová síť Ethernet byla vyvinuta firmami DEC, Intel a Xerox. Původní Ethernet pracoval s přenosovou rychlostí 10Mb/s, pozdější varianta uvedena na trh pod názvem Fast Ethernet umožňovala používat rychlost 100 Mb/s. V současné době je běžně používán tzv. Gigabit Ethernet pracující rychlostí 1000 Mb/s.

K přenosu dat se používá ethernetový rámec, který má proměnlivou délku. Minimální délka je 68 bytů (544 bitů), maximální 1522 bytů (12176 bitů) [7]. Struktura rámce je zobrazena na obr. 2.7.

Ethernet má na začátku synchronizační preambuli, která je součástí fyzické vrstvy. Následuje MAC (Media Access Control) adresa příjemce a odesílatele, což je 48 bitové



Obrázek 2.7: Formát Ethernet rámce

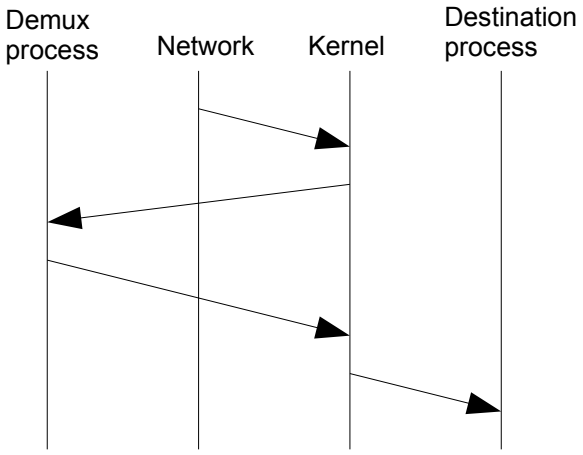
číslo, které jednoznačně identifikuje síťové rozhraní. MAC adresa je rozělena na dvě části. První část identifikuje výrobce daného zařízeni a druhou část určuje sám výrobce, aby definoval celosvětovou jedinečnost zařízení. Typ rámce pro normu IEEE 802.3 vyjadřuje délku datového pole v bajtech a pro normu Ethernet II (používáno protokolem TCP/IP) určuje typ paketu. Následuje proměnné pole, které obsahuje data vyšší (internetové vrstvy) a kontrolní součet (CRC).

Kapitola 3

Technologie odchytávání síťových paketů

V současné době existuje velké množství nástrojů na odchytávání paketů jak na operačních systémech typu Unix, tak i na systémech MS Windows. Rozsah této práce neumožňuje popsat jednotlivé nástroje na různých operačních systémech, proto se zaměřím na systémy unixového typu (konkrétně FreeBSD [13]). Nebudu zde popisovat jednotlivé aplikace, ale nástroje a techniky, pomocí kterých je možné aplikace naprogramovat. Existují ještě hardwarová řešení, nicméně těm se věnovat také nebudu.

Hlavní výhodou softwarových nástrojů na odchytávání paketů je jejich univerzálnost. Naopak velká nevýhoda spočívá v malé vykonnosti u vysokorychlostních sítí. Díky těmto problémům vznikly dva přístupy návrhu a implementace síťových analyzátorů [15].

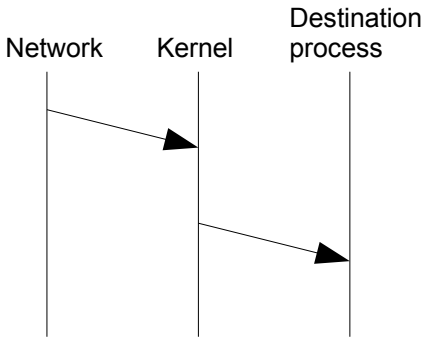


Obrázek 3.1: Přepínání na úrovni uživatelského procesu

První přístup využívá dělení toku dat na úrovni uživatelského procesu (obrázek č. 3.1). Jak je z obrázku patrné, doručení dat ze síťového toku cílovému procesu je celkem náročné, což je velká nevýhoda. Naopak výhodou použití pomocného procesu na řízení toku dat je v jednoduché implementaci analyzátoru.

Druhá možnost vychází z faktu, že téměř jedna třetina zdrojových kódů jádra operačního

systému se týká síťových služeb. Proto i samotný nástroj pro odchytávání paketů je umístěn v jádře (obrázek č. 3.2). Z obrázku proti předchozímu řešení je vidět, že pokud je analyzátor součástí jádra, dokáže stejnou službu vykonávat mnohem rychleji. Dalším nezanedbatelným pozitivem je i možnost aplikovat jednoduchý filtr, který dokáže propustit konkrétní pakety. Tím se nám propustnost dat ještě zvýší.



Obrázek 3.2: Přepínání na úrovni jádra

Mezi nevýhody patří především složitost realizace analyzátoru jako součást jádra. Při ladění a hledání chyb v analyzátoru často dochází k nenadálým pádům nebo k zatuhnutí operačního systému. Na základě těchto problémů vzniklo několik více či méně úspěšných projektů. Mezi známé patří např. systém NIT (Network Interface Tap), který byl uvolněn společně s operačním systémem SunOS 4.1, nebo systém BPF (Berkeley Packet Filter).

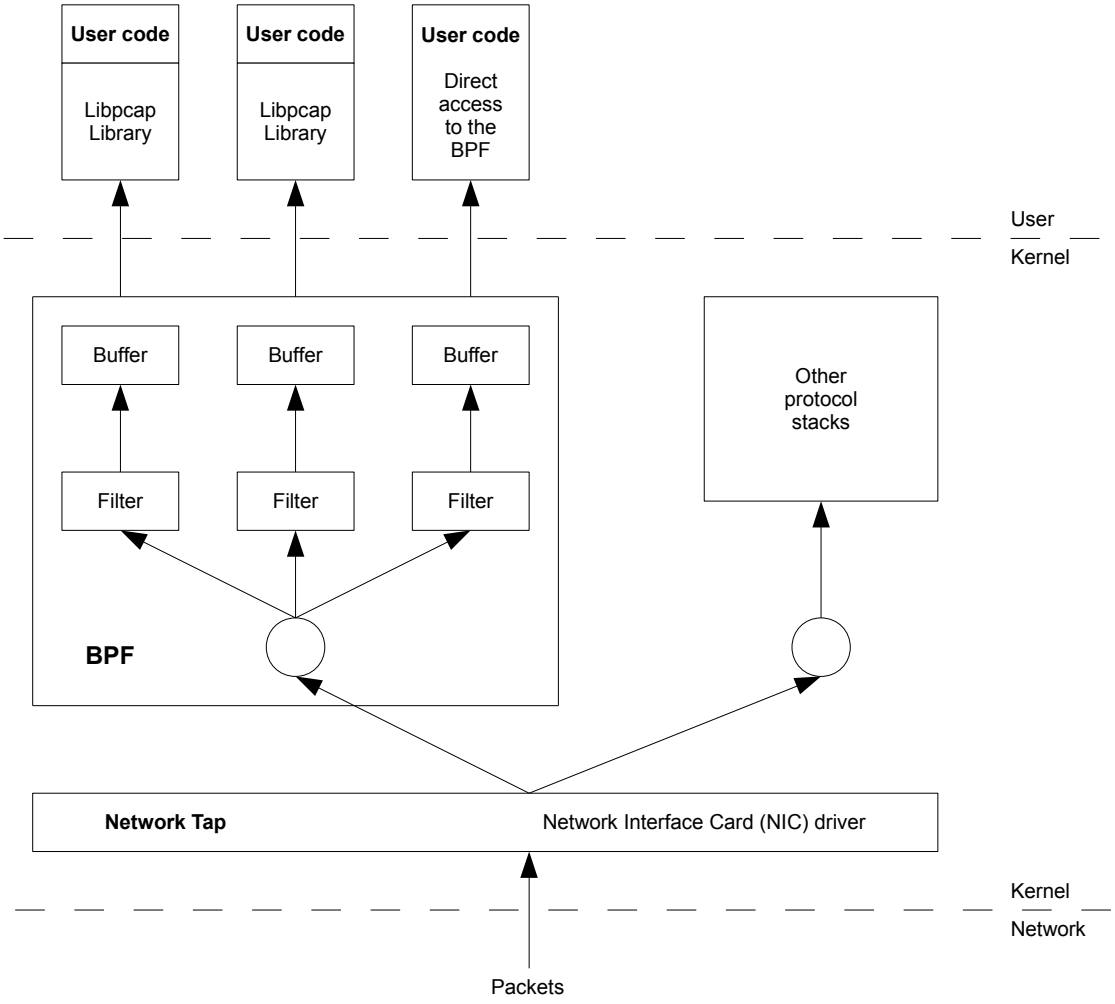
3.1 Berkeley Packet Filter

Berkeley Packet Filter (BPF) [14] [20] [11] je součást jádra operačního systému, který umožňuje plný („raw“) přístup na linkovou vrstvu síťového rozhraní. BPF používá rozhodovací filtr pracující na registrech, díky čemuž dosahuje vysoké propustnosti dat. Původní unixový paketový filtr používal rozhodovací mechanismus založený na principu zásobníku, a proto byl až 20 krát pomalejší.

BPF má dlouhou, téměř 30ti letou historii. Předchůdce BPF je paketový filtr Enet, který byl vytvořen v roce 1980 pány Mike Accetta a Rick Rashid na Carnegie Mellon University. Filtr Enet byl úspěšný, a proto ho Jeffrey Mogul [15] přenesl na operační systém BSD (Berkeley Software Distribution), kde ho dále do roku 1983 vylepšoval. Díky desetiletému vývoji filtru Enet vzniká v roce 1990 BPF. Autory jsou pánové Steven McCanne a Van Jacobson.

BPF se skládá z následujících dvou součástí. První část je síťový odposlech (Network Tap) a druhá část je paketový filtr. Na obrázku 3.3 je znázorněn celý systém odchytávání paketů.

Síťové funkce operačního systému normálně (bez spuštěného BPF) fungují tak, že pakety přicházejí na rozhraní síťové karty, které jsou dále předávány do zásobníku síťových protokolů. Jakmile je BPF aktivován (začne poslouchat na rozhraní síťové karty), činnost síťového podsystému jádra se změní. Při doručení paketu na rozhraní síťové karty jsou data nejdříve předána BPF, a poté i do zásobníku síťových protokolů (kde jsou dále zpra-



Obrázek 3.3: Berkeley Packet Filter

cována příslušnou aplikací). BPF převezme data a předá je do filtru ke zpracování. Ve filtru jsou uživatelem definována pravidla, která určují, zda budou či nebudou moci data putovat do vyrovnávací paměti (buffer) a pak dále k příslušnému procesu. Jakmile jsou data ve vyrovnávací paměti, BPF předává řízení síťové kartě a čeká na další příchozí paket.

Jelikož BPF filtruje každý paket, který mu je předán a čas mezi doručenými pakety je velmi malý (řádově mikrosekundy), není možné použít pro každý paket systémové volání **read**. Z toho důvodu je použita vyrovnávací paměť, která je rozdělena na dvě části (*store* a *hold*). Do části store se ukládají příchozí data. Jakmile dojde k naplnění, obě části BPF prohodí. Nyní se z části store stává část hold, která uvolní data uživateskému procesu. Tím je vyprázdněna a čeká na zaplnění store části a na následnou výměnu. Pomocí tohoto mechanismu má BPF vysokou datovou propustnost.

3.2 Libpcap

Libpcap [2] [9] je programátorské rozhraní (API), které poskytuje nástroje k odchytávání a filtrování paketů. Vytváří abstraktní rozhraní mezi samotnou aplikací a jádrem operačního systému, díky čemuž je návrh a realizace programu jednodušší.

Knihovna pracuje na uživatelské úrovni, není tolik závislá na operačním systému, a proto ji bylo možné úspěšně přenést na několik druhů operačního systému Unix. Nevýhodou některých Unixů je absence BPF (např. Solaris), díky čemuž musí knihovna libpcap emulovat BPF a její výkonnost klesá. Na operačních systémech rodiny Windows existuje knihovna WinPcap [3], která má podobné vlastnosti jako knihovna libpcap. Pomocí knihovny libpcap je vytvořen i tento analyzátor, více o implementaci je v následující kapitole.

3.3 Analýza protokolů aplikační vrstvy

Analýza protokolů na síťové a transportní vrstvě není složitá, protože všechny důležité informace jsou přenášené v záhlaví příslušné vrstvy. Náročnější na analýzu je ale vrstva nejvyšší, aplikační. Při zkoumání payload (samostatná přenášená data, bez TCP/IP záhlaví) můžeme využít číslo portu [1] z TCP/UDP záhlaví. Toto řešení není úplně ideální, protože většina portů nemá přidělené konkrétní číslo. Situaci komplikují i nově vznikající služby a programy, které si čísla portů náhodně vybírají. Proto vzniklo několik metod na identifikování a analýzu dat na aplikační úrovni [12] [8].

- **Běžná metoda určování aplikačních protokolů**
  Běžná metoda je založená na dobře známých číslech portů. Známá čísla musí být menší než 1024 a nebo musí být uvedena na seznamu orgranizace IANA [1]. Pomocí čísel portů můžeme snadno rozpoznat použitý protokol. Dříve tato metoda dostačovala k identifikaci téměř veškerého síťového provozu. Výhoda běžné metody je v jednoduché implementaci. Hlavní nevýhoda je neschopnost identifikace dynamicky generovaných čísel portů (např. stremování multimédií).

- **Metoda založená na vyšetřování payload**
  Metoda založená na vyšetřování dat nejvyšší vrstvy je schopná odhalit aplikace, které využívají dynamické generování čísel portů. Často se s touto metodou setkáme u streamování hudby nebo videa a taky u Peer-to-Peer (P2P) programů, kde je nutné vytvořit kontrolní a datový kanál mezi komunikujícími stranami. Metoda je velice přesná, což je její výhoda. Naopak nepříjemné je, že více zatěžuje systém, a proto může mít nižší propustnost.

- **Metoda založená na signaturách**
  Metoda využívající signatury (signatura je textový řetězec) je založená na faktu, že část aplikačního protokolu je jednoznačně určena. Signatura přesně odpovídá charakteristické části protokolu. Při analýze je každý payload porovnán se signaturou. Díky porovnávání všech paketů je přesnost metody vysoká. Naopak nepříjemná je velká náročnost na systémové prostředky.

- **Metoda používaná k identifikaci P2P provozu**
  V poslední době se množství přenesených dat posouvá od klasických služeb (web, e-mail) ke sdílení souborů a přenášení velkého objemu dat. Existuje spousty programů

nabízejících služby typu P2P, DirectConnect nebo KaZaA. Často jsou přenášený nelegální soubory a je nutné tyto služby omezit. Proto byly vytvořeny speciální analyzátory na konkrétní typ služby, např. DirectConnect. Účinnost těchto analyzátoru je vysoká, pouze pokuď jsou použity na konkrétní protokol, na který byly vytvořeny.

Kapitola 4

Návrh a implementace

V této kapitole nejprve na základě analýzy daného problému zvolím nejvhodnější řešení a poté popíšu implementaci konečné aplikace. Cílem návrhu a implementace je co nejrychlejší činnost analyzátoru síťového provozu, a také schopnost běhu na různých operačních systémech unixového typu (FreeBSD, Linux, Solaris aj.).

4.1 Návrh

Kvalitní návrh aplikace je pro vývoj klíčový, protože se od něj odvíjejí veškeré vlastnosti konečného programu. Je tedy důležité věnovat návrhu patřičnou pozornost. Jako nejvýhodnější se mi jeví prozkoumat podobné nástroje na analyzování síťového provozu (tcpdump, wireshark, snort atd.) a poté implementovat použité techniky z těchto nástrojů.

4.1.1 Analýza problému

Aby bylo možné navrhnout co možná nejefektivnější řešení daného problému, je nutné ho důkladně analyzovat. Tato analýza by měla odpovědět na dvě hlavní otázky, kterými jsou:

- Jaký použít systém na odchytávání paketů síťové komunikace?
- Jak efektivně analyzovat provoz na aplikační vrstvě?

Odpověď na první otázku není tolik náročná, protože v současné době existuje velké množství řešení. Na odchytávání paketů je možné použít jak speciální hardwarová zařízení, tak i optimalizované softwarové nástroje. Výhodou hardwarových zařízení je vysoká efektivita odchytávání síťového provozu. Nopak nepříjemná je horší dostupnost těchto zařízení, a také cena je dost vysoká. U softwarových nástrojů je situace opačná. Jsou snadno dostupné, některé se distribuují pod svobodnou licencí. Mezi další výhodu patří univerzálnost řešení. V této práci byl použit softwarový nástroj na odchytávání paketů, konkrétně knihovna libpcap [2].

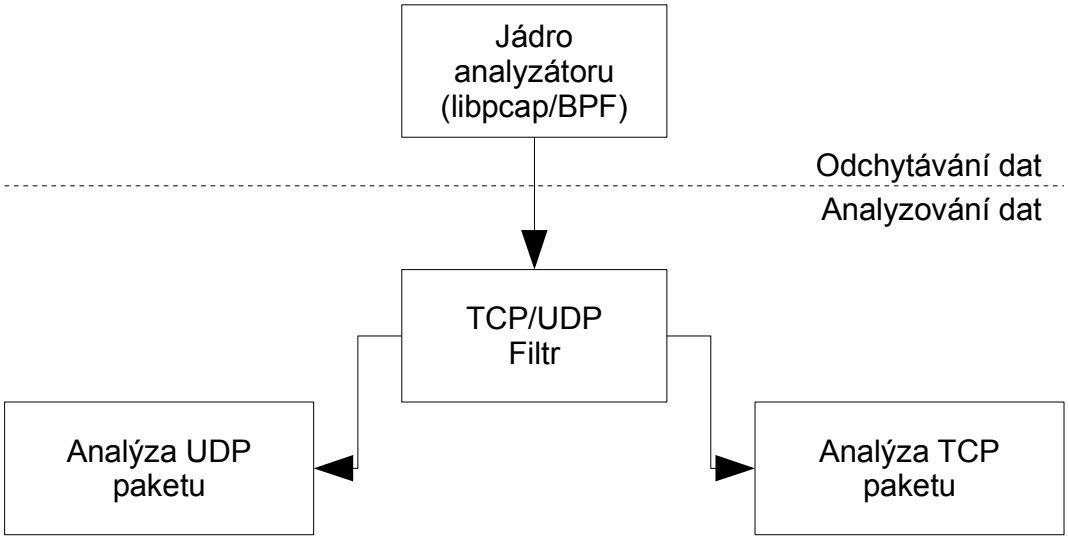
Zodpovězení druhé otázky je o něco náročnější. Ještě v nedávné minulosti se na rozpoznávání protokolů aplikační vrstvy výhradně používaly čísla portů [1]. Nicméně v současné době není tento mechanizmus tak spolehlivý. Důvodem je používání různých aplikací, které využívají ke komunikaci s protistranou dynamicky generovaná čísla portů. Z tohoto důvodu vzniklo několik mechanizmů, které analyzují pakety jinou metodou než jenom zkoumáním čísel portů. Mezi tyto metody patří i detekce protokolů založená na signaturách, což jsou

řetězce specifické pro konkrétní aplikační protokol. V této práci je nejen použita metoda založená na detekci čísel portů, ale i metoda založená na signaturách.

Neméně důležité je i použití vhodného operačního systému. Jak jsem se zmínil dříve, popisovaná aplikace využívá knihovnu libpcap a je ji možné používat na operačním systému FreeBSD. Je možné ale použít jiné kombinace, jako např. OS Linux. Take existují nástroje vhodné pro operační systémy rodiny Microsoft Windows, konkrétně knihovna winpcap.

4.2 Implementace

Během vývoje byl jako referenční použit osobní počítač s procesorem architektury i386, konkrétně AMD Athlon 1800+. Paměti bylo možné využít 1024 MB. Síťové rozhraní počítače je typu Ethernet. Jako implementační jazyk byl použit jazyk C. Síťový analyzátor je rozdělen na dvě části. Jedna se stará o odchytávání síťového provozu a druhá o samotné dekódování komunikace. Celý systém je znázorněn na obrázku č. 4.1.



Obrázek 4.1: Schéma analyzátoru

4.2.1 Odchytávání dat

Jádrem síťového analyzátoru je systém na odchytávání dat, který je implementován pomocí knihovny libpcap. Díky této knihovně je vytvoření jádra analyzátoru jednodušší, protože není nutné přistupovat k nízkoúrovňovým funkcím operačního systému.

První věc, jenž program musí vykonat, je zjištění jména síťového zařízení, na kterém bude odposlouchávat síťovou komunikaci. Jelikož je velmi časté, že počítač obsahuje více síťových karet, je dobré vyhledat všechny názvy (zajistí funkce `pcap_findalldevs()`), a poté si správně ze seznamu vybrat. Následuje nastavení síťové karty na odposlouchávání a

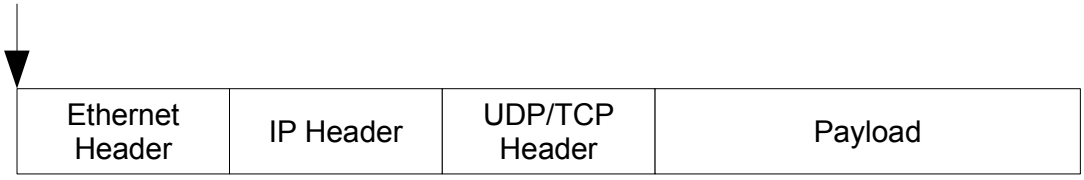
přepnutí do tzv. *promiskuitního režimu*. Promiskuitní režim způsobí, že síťová karta přijímá veškeré rámce linkové vrstvy.

Nevýhodou příjímaní všech linkových rámců je v zahlcení systému. Jakmile čas potřebný k analýze jednoho rámce je vyšší než rozdíl časů po sobě jdoucích rámců, dochází ke ztrátě přijatých rámců a tím i nemožnost určit aplikační protokol. Proto je vhodné využít paketový filtr BPF [14], který umožňuje filtrovat data na síťové a transportní vrstvě. Zařízení BPF odposlouchává data na úrovni jádra operačního systému a je díky tomu velmi efektivní. Konfigurace filtru je velice rozsáhlá a ani není předmětem této práce. Úplný popis jednotlivých výrazů je možné najít v manuálových stránkách programu `tcpdump` [10].

Pokud je síťové rozhraní správně nastaveno a BFP zařízení vhodně nakonfigurováno, může začít odposlouchávání síťové komunikace a předávání dat analyzátoru aplikačních protokolů. Knihovna libpcap při každém zachyceném rámci předává ukazatel na jeho začátek a zároveň strukturu `pcap_pkthdr`, ve které je uložen čas odchycení rámce, velikost odchycených dat a skutečnou velikost dat. Přijatá data odebírá druhá část analyzátoru, která má za úkol provést rozbor každého rámce a zjistit protokol aplikační vrstvy případně jiné informace.

4.2.2 Zpracování přijatých dat

Jelikož knihovna libpcap předává ukazatel na začátek odchyceného rámce (obrázek č. 4.2), není možné hned analyzovat data na aplikační vrstvě, ale je potřeba se k nim nejdříve dostat, tzn. vhodně posunout ukazatel na začátek aplikační vrstvy. Proto je velmi důležité znát přesné uspořádání hlavičky každé vrstvy.



Obrázek 4.2: Odchycený rámec

První záhlaví na obrázku 4.2 je záhlaví vrstvy síťového rozhraní ethernet. Skládá se z MAC (Media Access Control) adresy příjemce a z MAC adresy odesílatele. Poslední položka má dvojí použití. Pokud struktura rámce odpovídá normě Ethernet II, poté tato položka nese typ protokolu. Pokud struktura rámce odpovídá normě ISO 8802-3, tak v tého části je zaznamenána délka přenášených dat. V současné době se v naprosté většině případů v Internetu setkáváme s protokolem Etherent II. Velikost hlavičky je 14 bajtů. Abychom mohli získat jednotlivé udáje z hlavičky, je nutné v jazyce C definovat následující strukturu:

```
struct ether_header {
    u_char  ether_dhost[ETHER_ADDR_LEN];    // destination host address
    u_char  ether_shost[ETHER_ADDR_LEN];    // source host address
    u_short ether_type;                     // type
};
```

Pomocí přetypování přijatých dat touto strukturou je možné získat MAC adresy a typ protokolu vyšší vrstvy. Tento princip přetypování je použit i u síťové a transportní vrstvy. Důležité je dát si pozor na ukazatele, aby neukazovaly na špatné místo v paměti. Tím bychom dostaly nepřesné údaje nebo by byl možný pád celé aplikace.

Posunutím ukazatele o velikost záhlaví linkové vrstvy (v našem případě 14 bajtů) je možné přistupovat k jednotlivým položkám záhlaví síťové vrstvy. Opět je nutné použít strukturu kvůli přetypování, která vypadá takto:

```
struct ip_header {
    u_char  ip_vhl;         // version (4 bits) + IP header length
    u_char  ip_tos;         // type of service
    u_short ip_len;         // total length IP datagram
    u_short ip_id;          // identification
    u_short ip_flags_fo;    // flags (3 bits) + fragment offset
    u_char  ip_ttl;         // time to live
    u_char  ip_proto;       // protocol transport layer
    u_short ip_crc;         // checksum
    struct  in_addr ip_src, ip_dst; // source and destination address
};
```

Jedná se o protokol IP verze 4. Z pohledu analyzátoru protokolů aplikační vrstvy jsou nejdůležitější položky délka záhlaví IP datagramu a typ protokolu vyšší vrstvy. Délka záhlaví není uváděna v bajtech, ale ve čtyřbajtech. Proto musí být délka záhlaví násobena čtyřmi, abychom dostali správnou hodnotu. Položka protokol vyšší vrstvy bude nabývat hodnot pouze TCP nebo UDP, ostatní protokoly budou ignorovány.

Dekódování na transportní vrstvě je podobné jako v předchozím případě. Rozdíl je v použití dvou transportních protokolů – TCP a UDP. TCP je složitější protokol. Definice TCP hlavičky je následující:

```
struct tcp_header {
  u_short th_sport;         // source port
  u_short th_dport;         // destination port
  u_int   th_seq;           // sequence number
  u_int   th_ack;           // acknowldgement number
  u_char  th_hl;            // header length (4 bits) + reserve (4 bits)
  u_char  th_flags;         // flags (8 bits)
  u_short th_win;           // window size
  u_short th_crc;           // checksum
  u_short th_urp;           // urgent pointer
};
```

Důležité položky TCP protokolu jsou zdrojový a cílovy port, některé příznaky a délká záhlaví. Délka záhlaví je opět uváděna ve čtyřbajtech. V současné době se používá osm příznaků (RFC 3168 [19]) místo dřívějších šesti.

Záhlaví UDP protokolu je jednodušší než u TCP. Skládá se pouze ze čtyř položek, mezi které patří číslo zdrojového a cílového portu, kontrolní součet a délka dat. Délka dat je v tomto případě velikost záhlaví a velikost dat, které protokol nese. Minimální délka záhlaví je tedy 8 bajtů, tj. UDP paket obsahující pouze záhlaví a žádná data.

4.2.3 Analýza payload

Jakmile jdou zjištěny důležité hodnoty ze síťové a transportní vrstvý a ukazatel je nastaven na začátek aplikační vrstvy, je možné přistoupit k samotné analýze aplikační vrstvy. Program používá k rozboru dat dva různé přístupy. První přístup určuje typ aplikačního protokolu na základě čísel portů, naopak druhý přístup je založen na signaturách.

Metoda využívající k analýze aplikační vrstvy čísla portů je nejstarší metoda, a proto je velice rozšířená. Její výhoda je v jednoduchém použití. Všechny známé čísla portů přiřazené konkrétním aplikačním protokolům je možné najít na webových stránkách organizace IANA [1], případně na operačních systémech typu Unix je možné čísla najít v souboru /etc/services. Struktura souboru je pevně daná, každý řádek obsahuje jeden záznam a vypadá následovně:

```
jméno protokolu   číslo portu/[udp|tcp]   komentář
```

Po spuštění analyzátor načte všechny záznamy ze souboru do pole struktur. Pole obsahuje 65 536 struktur a indexem je číslo portu. Každá struktura obsahuje jméno aplikačního protokolu, číslo portu, typ transportního protokolu a komentář. Definice struktury je takováto:

```
struct TPortInfo {
  const char *m_pcszName;
  const char *m_pcszComment;
  const u_char  m_cucSupportedProtocols;
  unsigned m_uCountTCPsrc, m_uCountTCPdst;
  unsigned m_uCountUDPsrc, m_uCountUDPdst;
  unsigned m_uCountDDPsrc, m_uCountDDPdst;
};
```

Poslední tři řádky slouží k uchovávání počtu použití jednotlivých portů a následně k výpisu statistik. Při odchytnutí linkového rámce a následném analyzování dojde pouze k porovnání čísel portů. Pokud číslo portu odpovídá nějakému záznamu, je příslušné jméno protokolu vypsáno na standardní výstup. V opačném případě program oznámí, že aplikační protokol je neznámý. Díky tomuto jednoduchému mechanizmu je možné získat celkem věrohodný přehled o použití různých aplikací ve sledované síti.

Nepříjemná vlastnost analyzátoru, který rozpoznává aplikační protokoly pouze na základě čísel portů, je v neschopnosti vypátrat různé programy, jejž používají ke komunikaci náhodná čísla portů. Pro analýzu protokolů, které využívají dynamicky generované čísla portů, je tedy nutné použít důmyslnější nástroje. Jedním z nich je zkoumání signatur jednotlivých aplikačních protokolů (na podobné metodě je založen systém IDS Bro [4], nebo projekt l7-filter [5] [6]).

Program využívá pro popis signatur aplikačních protokolů regulární výrazy. Reglární výrazy může uživatel vymyslet sám, nebo je možné některé regulární výrazy převzít z projektu l7-filtr [5]. Regulární výraz je řetězec popisující celou množinu řetězců. Regulární výraz se skládá z literálů textu, které se mají shodovat, a speciálních znaků, které nejsou součástí hledaného textu.

Programové vybavení, tedy funkce a struktury pro práci s regulárními výrazy je možné najít v souboru `regex.h`, který se nachází ve zdrojových kódech operačního systému (např. FreeBSD má tento soubor v adresáři `/usr/include`). Pokud má program použít regulární

výrazy na analýzu síťového provozu, je nejdříve nutné přeložit daný regulární výraz do tzv. vnitřího stavu (deterministický konečný automat). Toho dosáhneme pomocí funkce `regcomp()`. Jakmile je regulární výraz ve formě struktury, která popisuje deterministický konečný automat, je možné použít funkci `regexec()` na vyhledávání retězce v odchytávaných datech.

Nevýhodou v použití regulárních výrazů je náročnost na výkon počítače. Není většinou možné, aby analýza síťového provozu pomocí signatur probíhala na veškerých odchycených datech, zvláště pak na vysokorychlostních sítích. Docházelo by k úniku dat a tím i k nemožnosti analyzovat každý paket. Proto je nutné vhodně nastavit nízkoúrovňový filter zařízení BPF (např. na `tcp or udp` nebo na `port 80`). Díky tomu nebude tolik docházet k zahlcení systému.

Kapitola 5

Testování a výsledky

V této kapitole jsou popsány různé testy, které byly prováděné na sledovaném systému. Při realizaci analyzátoru síťového provozu je nutné obsloužit všechny příchozí linkové rámce a to v co nejkratší době. Díky testům je možné odhalit slabá místa systému a navrhnou optimalizace. Testy byly prováděny na síti typu Ethernet. Použitý počítač měl procesor AMD Athlon 64 o frekvenci 2400 MHz a 1024 MB RAM. Operační systém byl použit FreeBSD 7.1.

5.1 Metodika testování

Základem provedených testů bylo měření času mezi různými dvěma částmi programu. Na tyto účely se hodí funkce a struktury deklarované v hlavičkovém souboru `time.h`. Konkrétní použitá struktura se jmenuje `timespec` definovaná následovně:

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

Ve zdrojovém kódu programu jsou použity dvě tyto struktury – na začátku a na konci měřeného úseku. Pomocí funkce `clock_gettime()` jsou inicializovány. Rozdílem hodnot ze struktury timespec je možné získat dobu analýzy jednoho rámce.

Všechna měření byla prováděna v šesti pokusech. Rychlost přenášení dat byla na třech úrovních – 10 Mb/s, 100 Mb/s a 1000 Mb/s. Při každém testu je možné nastavit filtrovaní na úrovni BPF. Všechny testy mají stejnou úroveň nastavení filtru a to na hodnotu „ip“. V reálném provozu je možné různě optimalizovat tento filtr a dosáhnout tak lepších výsledků.

Testy jsou rozděleny na dvě části. První část využíva k analýze čísla portů, druhá část signatury. Nejprve je otestována rychlost detekce běžně používaných protokolů. Poté následuje test vytížení počítače a počet odchycených, zpracovaných a odmítnutých rámců. Zejména poměr počtu odchycených paketů ku počtu zpracovaných paketů vypovídá o efektivnosti navrhnutého systému. Poslední test byl zaměřen na zjištění čísel portů, které jsou ve sledované síti používány.

5.2 Testy a výsledky

Cílem prvního měření bylo zjistit rychlost určování aplikačního protokolu na základě čísla portu. Jak je vidět z tabulky 5.1, detekce probíhala celkém rychle a v podobných časech u všech zkoumaných protokolů. Je to z toho důvodu, že určování aplikačních protokolů na základě všech známých portů je pro všechny aplikační protokoly stejná. Analyzátor obsloužil všechny odchycené rámce, protože připojení do Internetu je řádove v jednotkách Mb/s.

Měření:	1. [μs]	2. [μs]	3. [μs]	4. [μs]	5. [μs]	6. [μs]	Průměr
HTTP	15,37	17,32	15,64	15,37	15,65	15,95	15,88
FTP	15,64	28,78	15,66	15,92	15,65	25,98	19,61
SMTP	20,31	16,23	15,39	15,33	15,92	15,37	16,43
POP3	17,04	15,64	15,92	21,51	15,64	15,62	16,90
DNS	15,65	15,92	16,48	15,37	40,23	15,62	19,88
SSH	16,20	15,92	15,36	15,64	15,92	16,48	15,92

Tabulka 5.1: Čas potřebný k analýze jednoho rámce (čísla portů)

Následující test byl provedem s ohledem na celkovou datovou propust analyzátoru, zejména pak co se týče rychlosti analýzy paketů. Za tímto účelem byl vytvořen soubor o velikosti 200 Mb, který byl kopírován z jednoho počítače na druhý. Oba počítače jsem zapojil přímo křížovým kabelem, aby docházelo co možná nejméně ke kolizím. Z tabulky 5.2 je vidět, ze v případě rychlosti přenosu dat 10 Mb/s je úspěšnost analyzování paketů vysoká. I vytížení procesoru je na přijatelný úrovni.

	Využití CPU	Odchycené	Zpracované	Nezpracované	Úspěšnost
Rychlost sítě 10 Mb/s					
HTTP	58 %	202 350	195 537	6 709	96 %
FTP	66 %	205 422	202 114	3 216	98 %
Rychlost sítě 100 Mb/s					
HTTP	88 %	201 363	28 387	172 965	14 %
FTP	97 %	211 618	23 274	188 331	11 %
Rychlost sítě 1000 Mb/s					
HTTP	98 %	220 523	3 876	216 647	1,75 %
FTP	98 %	213 711	3 625	210 086	1,69 %

Tabulka 5.2: Úspěšnost analýzy odchycených rámců

Naopak u sítí o rychlostech 1 000 Mb/s nebo vyšších je použití tohoto analyzátoru téměř nemožné. Procesor je zatížen úplně na maximum a většina přijatých paketů musí být zahozena. Zpracovaných rámců jsou necelá dvě procenta, což je každý 56tý.

Zatímco určování aplikačních protokolů využívající čísla portů je na pomalejších sítích celkem úspěšné, při použití signatur k detekci je situace komplikovanější. Záleží totiž na složitosti regulárního výrazu, který daný aplikační protokol popisuje. Z tohoto důvodu je doba potřebná k analyzování konkrétního protokolu různě dlouhá.

Tabulka 5.3 shrnuje dobu analýzy jednoho rámce konkrétního protokolu. Na první pohled je vidět, že průměrné časy se zvýšily, některé dokonce několikrát. Zejména u protokolu

Měření:	1. [μs]	2. [μs]	3. [μs]	4. [μs]	5. [μs]	6. [μs]	Průměr
HTTP	75,37	87,37	89,14	95,37	98,65	91,92	88,63
FTP	44,64	47,87	55,63	49,92	53,65	60,98	52,27
SMTP	30,32	36,23	29,39	31,33	35,25	33,37	32,65
POP3	37,24	31,64	45,29	35,73	32,46	39,42	36,97
DNS	65,45	55,32	59,48	69,37	54,83	65,65	61,68
SSH	26,23	22,25	31,36	25,47	29,92	31,65	27,81

Tabulka 5.3: Čas potřebný k analýze jednoho rámce (signatury)

http je nárůst času detekce vysoký. Nicméně zvýšení času se dalo očekávat, protože regulární výraz popisující http protokol je poměrně komplikovaný. Navíc vysoký čas detekce je také nepříjemný z dalšího důvodu, kterým je časté použití protokolu http. A jak je vidět z tabulky 5.5, téměř 72% komunikace je uskutečněná pomocí protokolu http.

	Využití CPU	Odchycené	Zpracované	Nezpracované	Úspěšnost
Rychlost sítě 10 Mb/s					
HTTP	98 %	203 012	35 557	166 943	18 %
FTP	98 %	206 131	71 736	134 394	34 %
Rychlost sítě 100 Mb/s					
HTTP	98 %	201 954	5 161	196 634	2,6 %
FTP	98 %	212 147	8 951	203 195	4,2 %
Rychlost sítě 1000 Mb/s					
HTTP	98 %	218 936	876	217 839	0,5 %
FTP	98 %	211 375	1 394	209 980	0,7 %

Tabulka 5.4: Úspěšnost analýzy odchycených rámců (signatury)

Pro zajímavost je v tabulce 5.4 uvedena úspěšnost detekce aplikačních protokolů v závislosti na různě použité přenosové rychlosti. U rychlostí kolem 1 Gb/s a výše je prakticky nemožné rozumným způsobem analyzovat síťový provoz tímto analyzátorem. Je potřeba použít jiné techniky, aby výsledek nebyl tak žalostný jako v tomto případě.

Poslední test byl zaměřen na zjištění většiny čísel portů, které jsou na sledované síti používány. Díky tomuto testu je možné získat celkový přehled používaných aplikací. Sledovaná síť se skládá ze čtyř počítačů a jednoho serveru, který slouží jako datové uložiště. Tato síť je připojena do Internetu prostřednictvím brány (router), na které běžel analyzátor 24 hodin. Rychlost připojení do Internetu je 2 Mb/s. Jednalo se o malou firemní síť. Po ukončení progrma vytiskl statistiku použitých portů. Seznam čísel portů je zobrazen v tabulce 5.5.

Provedený test odhalil některé zajímavosti. Většina síťového provozu uskutečněného směrem do Internetu používá protokol http, konkrétně 71 %. Na druhém místě je zabezpečený protokol https. Následují další protokoly aplikační vrstvy, které jsou ale použity v menší míře. Druhé zajimavé zjištění je, že většina (téměř 90 %) čísel portů je menší než 1024. Je to způsobené firemní politikou, kdy je zakázané sdílení souborů. U jiných sítí je použití síťových aplikací různé a tím i seznam čísel portů se bude jistě lišit. Zejména pokud se bude jednat o různé komunitní sítě, případně pirátské sítě, tak bude výskyt dynamicky

Protokol	Port	Počet rámců	% Rámců
HTTP	80	963 551	71,23 %
HTTPS	443	85 898	6,35 %
FTP	21	38 147	2,82 %
SSH	22	16 773	1,24 %
SMTP	25	19 885	1,47 %
POP	110	25 160	1,86 %
DNS	53	43 963	3,25 %
ICQ	5190	74 129	5,48 %
XMPP	5222	38 552	2,85 %
Zbytek		46 669	3,45 %
	< 1024	1 193 380	88,22 %
	≥ 1024	159 351	11,78 %
celkem		1 352 731	100 %

Tabulka 5.5: Často používaná čísla portů

generovaných čísel portů mnohonásobně více díky používání různých aplikací typu torrent aj.

Kapitola 6

Závěr

Hlavním cílem této práce bylo nalézt způsob, jak naprogramovat aplikaci typu síťový analyzátor, který je schopen efektivně odchytávat síťový provoz a analyzovat aplikační vrstvu. Tento problém nespočívá v prostém naprogramování dané aplikace, ale je nutné pochopit princip fungování počítačových sítí a operačních systémů. Také je nutné nastudovat problematiku kolem regulárních výrazů.

Ve druhé a třetí kapitole je probrána teorie ohledně technologií počítačových sítí a mechanizmů odchytávání dat. Díky těmto teoretickým poznatkům jsem byl schopen implementovat síťový analyzátor. Konkrétní popis implementace je rozebrán ve čtvrté kapitole. Nejdříve je popsána metoda odchytávání samotných dat za použití knihovny libpcap. V druhé části čtvrté kapitoly je rozebráno analyzování dat na aplikační úrovni.

Předposlední, pátá, kapitola obsahuje testování programu. Byly použity různé testy na ověření funkčnosti aplikace. Díky těmto testům byly odhaleny nedostatky, kterými analyzátor trpí. Zejména neschopnost využití programu na sítích s rychlostmi vyššími než 100 Mb/s, kdy je téměř nemožné analyzátor použít. Důvodem takto špatných výsledků je návrh aplikace, protože používá pouze jedno vlákno na vykonávání instrukcí. Jestliže tedy chceme použít softwarový analyzátor k detekování aplikačních protokolů na moderních vysokorychlostních sítích, je nezbytné, aby návrh a následná implementace používala více vláken. Zejména v dnešní době, kdy je na trhu velké množství vícejádrových procesorů, a proto se hodí využít více vláken, případně procesů.

Pokud ani vícevláknové aplikace nestačí na analyzování síťového provozu, potom nezbývá nic jiného než použít specializovaná hardwarová zařízení, která jsou velice rychlá a efektivní.

Literatura

[1] WWW stránky organizace IANA (čísla portů), Naposledy navštíveno 19. 12. 2008. `http://www.iana.org/assignments/port-numbers`.

[2] WWW stránky projektu Libpcap/Tcpdump, Naposledy navštíveno 26. 12. 2008. `http://www.tcpdump.org`.

[3] WWW stránky projektu WinPcap, Naposledy navštíveno 26. 12. 2008. `http://www.winpcap.org`.

[4] WWW stránky projektu IDS Bro, Naposledy navštíveno 5. 1. 2009. `http://www.bro-ids.org`.

[5] WWW stránky projektu L7-filter, Naposledy navštíveno 5. 1. 2009. `http://l7-filter.sourceforge.net`.

[6] WWW stránky, Naposledy navštíveno 5. 1. 2009. `http://www.protocolinfo.org`.

[7] Libor DOSTÁLEK and Alena KABELOVÁ. *Velký průvodce protokoly TCP/IP a systémem DNS*. Computer Press, Praha, 1999.

[8] Holger DREGER, Anja FELDMANN, Michael MAI, Vern PAXSON, and Robin SOMMER. Dynamic application-layer protocol analysis for network intrusion detection. [online], Naposledy navštíveno 3. 1. 2009. `http://www.icir.org/robin/papers/usenix06.pdf`.

[9] Van JACOBSON, Craig LERES, and Steven MCCANNE. pcap – packet capture library. Manuálové stránky FreeBSD.

[10] Van JACOBSON, Craig LERES, and Steven MCCANNE. tcpdump – dump traffic on a network. Manuálové stránky FreeBSD.

[11] Van JACOBSON and Steven MCCANNE. bpf – berkeley packet filter. Manuálové stránky FreeBSD.

[12] Myung-Sup KIM, Young J. WON, and James Won-Ki HONG. Application-level traffic monitoring and an analysis on ip networks. [online], Naposledy navštíveno 3. 1. 2009. `http://dpnm.postech.ac.kr/papers/ETRI-Journal/04/application-layer-analysis/application-layer-analysis.pdf`.

[13] Michael LUCAS. *Síťový operační systém FreeBSD*. Computer Press, Brno, 2003. ISBN 80-7226-795-7.

[14] Steven MCCANE and Van JACOBSON. The bsd packet filter: A new architecture for user-level packet capture. [online], Naposledy navštíveno 28. 12. 2008. http://www.tcpdump.org/papers/bpf-usenix93.pdf.

[15] Jeffrey C. MOGUL. The packet filter: An efficient mechanism for user-level network code. [online], Naposledy navštíveno 28. 12. 2008. http://www.hpl.hp.com/techreports/Compaq-DEC/WRL-87-2.pdf.

[16] Jon POSTEL. Internet protocol, rfc 791. [online], Naposledy navštíveno 6. 12. 2008. http://www.ietf.org/rfc/rfc0791.txt.

[17] Jon POSTEL. Transmission control protocol, rfc 739. [online], Naposledy navštíveno 6. 12. 2008. http://www.ietf.org/rfc/rfc0793.txt.

[18] Jon POSTEL. User datagram protocol, rfc 768. [online], Naposledy navštíveno 6. 12. 2008. http://www.ietf.org/rfc/rfc0768.txt.

[19] R. RAMAKRISHNAN, S. FLOYD, and D. BLACK. The addition of explicit congestion notification to ip, rfc 3168. [online], Naposledy navštíveno 10. 1. 2009. http://tools.ietf.org/html/rfc3168.

[20] Fulvio RISSO and Loris DEGIOANNI. An architecture for high performance network analysis. [online], Naposledy navštíveno 19. 12. 2008. http://www.winpcap.org/docs/iscc01-wpcap.pdf.

Dodatek A

Použití programu

Program je před použitím nutné přeložit. K tomu slouží soubor `Makefile`, který je dodáván spolu se zdrojovými kódy. Dále jsou přiloženy ještě soubory `etc.service` a `regexp`. V souboru etc.services jsou popsány všechny známé čísla portů a v souboru regexp jsou regulární výrazy popisující protokoly aplikační vrstvy. Pro správný chod analyzátoru je nutné mít tyto soubory ve stejném adresáři jako samotný program.

Aplikace je navržena na použití z příkazové řádky. Chování programu je možné ovlivnit parametry při spouštění.

Parametry programu:

- `-f (filtr)` podmínka pro zařízení BPF
- `-p (počet)` počet zobrazovaných znaků aplikační vrstvy (implicitně = 0)
- `-d (zařízení)` potlačí dotaz na číslo odposlouchávaného zařízení a zvolí ho přímo
- `-r` místo čísel portů se použijí k detekci regulární výrazy
- `-R (výraz)` vlastní regulární výraz
- `-n (počet)` počet znaků aplikační vrstvy zkoumaných regulárním výrazem
- `-h` tisk nápovědy

V průběhu činnosti analyzátoru jsou o každém odchyceném rámci vypisovány tyto údaje:

- Čas odchycení rámce
- Počet odchycených rámců
- IP adresa příjemce a odesílatele
- Číslo portu příjemce a odesílatele
- Velikost celkem přenesených dat v bajtech
- Typ protokolu transportní vrstvy
- V případě protokolu TCP typ příznaku
- Velikost dat aplikační vrstvy

- Tisk aplikačních dat v hexa a ascii formátu
- Počet všech přijatých rámců (knihovna libpcap)
- Počet všech nezpracovaných rámců (knihovna libpcap)

Ukončení programu je možné zasláním signálu SIGINT (Ctrl + c). Jakmile je signál SIGINT odchycen analyzátorem, tak jsou vytištěny statistiky a program je ukončen. Statistiky jsou následující:

- Seznam detekovaných aplikačních protokolů a počet rámců
- Počet TCP paketů menších než 1024
- Počet TCP paketů větších než 1024
- Počet UDP paketů menších než 1024
- Počet UDP paketů větších než 1024
- Celkový počet TCP paketů
- Celkový počet UDP paketů