VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

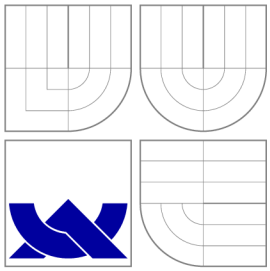
SLEDOVÁNÍ A ÚČTOVÁNÍ PROVOZU IP TELEFONIE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

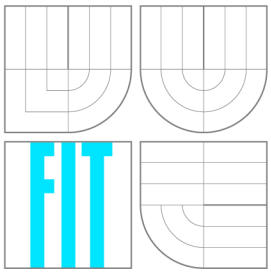
AUTOR PRÁCE
AUTHOR

Bc. VLADIMÍR SIVÁK

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SLEDOVÁNÍ A ÚČTOVÁNÍ PROVOZU IP TELEFONIE

ACCOUNTING AND INSPECTING IP TELEPHONY TRAFFIC

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. VLADIMÍR SIVÁK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. PETR MATOUŠEK, Ph.D.

BRNO 2011

Abstrakt

Tahle práce popisuje architektúru IP telefonie s využitím signalizačního protokolu SIP a transportního protokolu RTP. Zabývá sa také popisem nástrojů pro analýzu hlasových dat. Jejím hlavním cílem je navrhnout a implementovat systém pro detekci hovorů a extrakci hlasových dat ze zachycených paketů. Systém nejprve rozezná signalizační správy protokolu SIP, následně je analyzuje a pak na základě získaných dat generuje výstupní statistiku. Pokud jsou zachycena hlasová data, systém je uloží do výstupního souboru ve formě vhodné pro následné zpracování.

Abstract

This thesis describes architecture of IP telephony networks based on signaling protocol SIP and transport protocol RTP. Also tools used for analyzing VoIP traffic are described. Main objective is to design and implement a system for the detection of calls and extraction of voice payloads from the captured packets. The system first recognizes the signaling messages of SIP protocol. These messages are analyzed afterwards. Output statistics are generated based on gathered data. Voice data will be stored in form, which is suitable for further processing.

Klíčová slova

VoIP, SIP, RTP, Wireshark, Python, pcapy

Keywords

VoIP, SIP, RTP, Wireshark, Python, pcapy

Citace

Vladimír Sivák: Sledování a účtování provozu IP telefonie, diplomová práce, Brno, FIT VUT v Brně, 2011

Sledování a účtování provozu IP telefonie

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Petra Matouška Ph.D.

.....
Vladimír Sivák
23. května 2011

Poděkování

Chcem poďakovať pánovi Ing. Petrovi Matouškovi, Ph.D., ako vedúcemu mojej diplomovej práce, za jeho odborné vedenie, rady a všetku pomoc, ktorú mi poskytol pri vytváraní tejto práce. Rovnako patrí moja vďaka celej mojej rodine a všetkým priateľom, ktorí ma neustále podporovali a dôverovali mi.

Obsah

Kapitola 1

Úvod

Technológie prenosu hlasu prešli v posledných desaťročiach výraznými zmenami, ktoré boli spôsobené hlavne zvyšujúcimi sa nárokmi na kvalitu prenášaného hlasu. Okrem toho došlo výraznému zvýšeniu počtu hlasových hovorov a postupne sa začali okrem prenosu hlasu vyžadovať aj dodatočné služby, ako napríklad identifikácia volajúceho a prenos videa. To viedlo k posunu od analógového prenosu k digitálnemu. Stále sa však jednalo o dedikované siete na prenos hlasu, prípadne videa. Hlasové dáta sú totiž citlivé na kvalitu a dobu prenosu medzi jednotlivými účastníkmi.

S postupným zvýšovaním kapacít dátových sietí a s rozvojom dodatočných služieb, vrátane zaručenia kvality služieb (QoS - Quality of Service), sa začala postupne presadzovať idea unifikovaných sietí. Prenos dát, hlasu a videa mal byť zabezpečený jednotnou sieťovou infraštruktúrou. Z viacerých možností sa napokon ujala architektúra založená na protokole IP. Následne to viedlo k vzniku širokého portfólia nových produktov (IP telefóny, telekonferenčné zariadenia, prepínače a smerovače s podporou hlasových služieb, telefónne ústredne pre IP siete atď.) a postupne začali vznikať aj nové sieťové protokoly, ktoré boli potrebné pre zabezpečenie prenosu multimediálnych dát. Ide hlavne o signalizačné protokoly, ktoré majú za úlohu zostaviť reláciu medzi užívateľmi, ale aj o transportné protokoly, ktoré dáta cez sieť prenášajú.

1.1 Cieľ práce

Cieľom tejto práce je oboznámiť čitateľa hlavne so signalizačným protokolom SIP, ako aj s ďalšími protokolmi, ktoré sú potrebné pre fungovanie IP telefónie. Protokol SIP sa v dnešnej dobe považuje za štandard, ktorý postupne nahrádza proprietárne protokoly jednotlivých výrobcov. V tejto práci sa zameriame aj na analýzu jednotlivých signalizažných správ s ohľadom na získanie informácii potrebných pre účtovanie hlasových hovorov. Okrem účtovania preskúmame aj možnosti zachytávania jednotlivých hovorov a následnej rekonštrukcie hlasových dát. Následne bude popísaný návrh programu, ktorý bude implementovať požadovanú funkcionalitu.

1.2 Štruktúra práce

Detailným popisom protokolu SIP sa zaoberá druhá kapitola. Sú v nej popísané jednotlivé komponenty, signalizačné správy a architektúra protokolu. Okrem toho sa druhá kapitola ešte zaoberá popisom protokolu SDP (Session Description Protocol), ktorý sa používa na

popis multimediálnych relácii. V stručnosti sú popísané aj protokoly transportnej vrstvy RTP a RTCP, ktoré majú na starosti prenos hlasových dát. V závere druhej kapitoly sú ukázané príklady signalizácie pri vytvorení hovoru priamo medzi užívateľmi, ako aj v prípade využitia SIP proxy servera.

Tretia kapitola popisuje najčastejšie používané nástroje pre sledovanie IP telefónie. Popísané sú v nej hlavne spôsoby, ako jednotlivé paketové analyzátory použiť s ohľadom na signalizačný protokol SIP a prenos hlasových dát. Okrem paketový analyzátorov sú popísané aj nástroje na spracovanie zachytených hlasových dát.

Návrh aplikácie pre analýzu signalizačných správ a zachytávanie hlasových prúdov popisuje štvrtá kapitola. Postupne sú v nej rozobraté jednotlivé fázy činnosti programu spolu s príkladom, na ktorom je ukázané, ako z jednotlivých správ dostať potrebné informácie z hľadiska účtovanie hlasových hovorov. Následnou implementáciou programu sa zaoberé piata kapitola. Popísané je zvolené implementačné prostredie, spolu s dôležitými funkciami programu.

Šiesta kapitola sa zaoberá testovaním aplikácie. Popísané sú jednotlivé nástroje, ktoré sme pri testovaní použili, ako aj testovacie zapojenie. Porovnané sú výsledky našej aplikácie spolu s výsledkami vybraného nástroja pre sledovanie sieťovej prevádzky. Zhodnotenie dosiahnutých výsledkov, spolu s možnosťami ďalšieho vývoja obsahuje záverečná kapitola.

Kapitola 2

Architektúra IP telefónie

V tejto kapitole sa zameriame na popis architektúry IP telefónie, založenej na signalizačnom protokole SIP. Popíšeme si jednotlivé komponenty protokolu, používané správy, adresáciu užívateľov a spôsob popisu relácií počas prenosu. Zameriame sa aj na protokoly RTP a RTCP, ktoré slúžia na samotný prenos dát medzi účastníkmi hovoru. V závere kapitoly si na konkrétnych príkladoch ukážeme postupnosť jednotlivých signalizačných správ pri vytváraní relácie. Vzhľadom ku charakteru tejto práce nie je možné do detailov vysvetliť všetky vlastnosti a schopnosti protokolov SIP, RTP a RTCP. Preto sú vždy uvádzané zdroje na príslušnú literatúru a RFC dokumenty, na ktoré sa môžu čitatelia obrátiť v prípade záujmu o ďalšie informácie.

2.1 Protokol SIP

Protokol SIP (Session initiation protocol) je aplikačný signalizačný protokol, ktorý sa v IP sieťach používa pri prenose multimediálnych relácií (reláciou rozumieme výmenu dát medzi účastníkmi). Slúži na ich vytváranie, modifikovanie počas behu a ukončovanie. Najčastejšie sa používa pri prenose hlasu ale je schopný pracovať aj s multimediálnymi konferenciami pre viacerých účastníkov, kde je okrem hlasu potrebné prenášať aj video. Pri vytváraní špecifikácie protokolu sa dbalo hlavne na to, aby bol čo najviac všeobecný a fungoval nezávisle na transportnom protokole a na type vytváranej relácie. Aktuálnu špecifikáciu popisuje RFC 3261 ([6]).

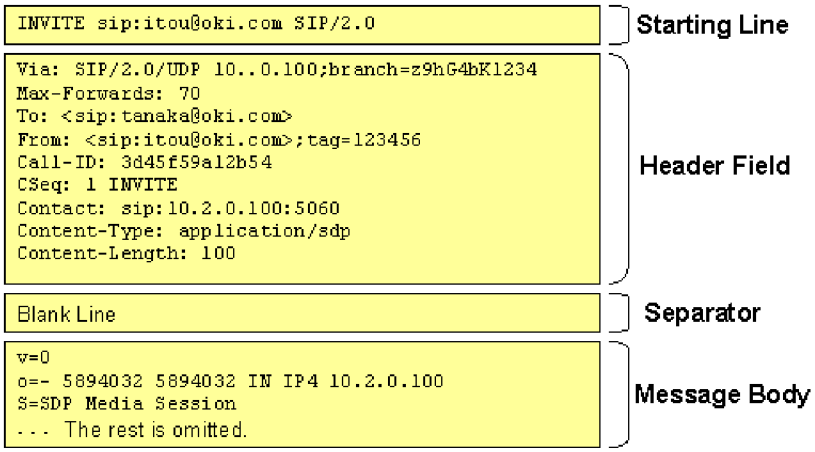
Protokol SIP poskytuje koncovým zariadeniam, ktoré sa často označujú ako užívateľskí agenti (User Agents), nasledovnú funkcionalitu:

- **lokalizácia užívateľa - user location**: rozhodnutie o tom, ktoré koncové zariadenie sa použije
- **dostupnosť užívateľa - user availability**: rozhodnutie o tom, či volaný účastník je dostupný a ochotný zapojiť sa do komunikácie
- **možnosti užívateľa - user capabilities**: zistenie schopností užívateľa (aké parametre spojenia je možné použiť)
- **zostavenie relácie - session setup**: rozhodnutie o finálnych parametroch a vlastné zostavenie relácie
- **manažment relácie - session management**: manažment relácie počas behu, vrátane upravenia parametrov a ukončenia relácie

Protokol SIP ale nie je schopný prenášať multimediálne dáta, preto pre vytvorenie komplexnej multimediálnej architektúry je ho potrebné použiť v spojení s inými protokolmi. Najčastejšie sa spolu s ním používa protokol RTP (Real-time Transport Protocol), ktorý slúži na prenos dát v reálnom čase.

2.1.1 Správy SIP

Protokol SIP je textový protokol založený na posielaní žiadostí (od klienta smerom na server) a odpovedí (zo servera ku klientovi). Formát správ je totožný s protokolom HTTP verzie 1.1. Každá správa obsahuje úvodný riadok (*starting line*), za ktorým nasleduje hlavička správy, ktorá musí byť ukončená prázdnym riadkom (*CRLF*). Po hlavičke nasleduje vlastné telo správy, ktoré sa používa na prenos správ SDP (Session description protocol) protokolu. Príklad SIP správy je na nasledujúcom obrázku 2.1.



Obrázek 2.1: Príklad správy SIP

V prípade správ typu žiadosť sa štartovací riadok označuje ako tzv. riadok žiadosti (*Request-Line*). Ten obsahuje názov použitej metódy, URI žiadosti a verziu protokolu. Všetky uvedené informácie sú vzájomne oddelené prázdnym znakom. Nasledujúce metódy sú súčasťou protokolu SIP už od verzie 1.0 a preto musia byť vždy podporované:

- **INVITE**: používa sa na pozvanie účastníka do komunikácie. Je to prvá správa počas nadväzovanie relácie SIP.
- **CANCEL**: metóda, ktorá slúži na zastavenie práve prebiehajúcej žiadosti. V prípade už zostavenej relácie však nedochádza k jej ukončeniu.
- **ACK**: potvrdzuje prijatie konečnej odpovede v rámci transakcie. Používa sa ako posledná správa pred vytvorením dátového streamu.
- **BYE**: ukončuje prebiehajúcu reláciu medzi užívateľmi.
- **REGISTER**: používa sa pri registrácii užívateľského agenta na registračnom serveri. Asociuje užívateľa a počítač, na ktorom je užívateľ prihlásený.
- **OPTIONS**: pomocou tejto metódy si užívateľský agent môže od druhej strany vyžiadať zoznam podporovaných SIP metód. V podstate slúži k odhaleniu schopností účastníkov bez vytvorenia relácie.

Okrem uvedených metód existujú aj ďalšie metódy, ktoré postupne vznikali počas rozširovania podporovanej funkcionality protokolu SIP. Napríklad s podporou posielania textových správ medzi účastníkmi (*instant messaging*) súvisia metódy **SUBSCRIBE**, **NOTIFY** a **MESSAGE**.

Správy, ktoré sú odpoveďou na predchádzajúcu žiadosť, majú ako štartovací riadok tzv. stavový riadok *Status-Line*. Skladá sa z verzie protokolu, číselného kódu stavu a z textovej reprezentácie stavu. Kód stavu je trojmiestne číslo, ktoré indikuje výsledok pokusu o vyplnenie žiadosti. Používa sa hlavne pre zjednodušené spracovanie odpovede automatickými skriptami. Stavové kódy môžu byť z nasledujúcich kategórií:

- **1xx - Informačné (Informational)**: informuje klienta, že žiadosť bola prijatá je momentálne spracovávaná.
- **2xx - Úspech (Success)**: potvrdzuje korektné prijatie a spracovanie žiadosti.
- **3xx - Presmerovanie (Redirection)**: oznamuje klientovi, že žiadosť bola prijatá, ale pre správne dokončenie je potrebná dodatočná akcia zo strany klienta.
- **4xx - Chyba na strane klienta (Client error)**: žiadosť zo strany klienta obsahovala chybu, preto ju nie je možné spracovať.
- **5xx - Chyba na strane servera (Server error)**: žiadosť bola prijatá na strane servera, ten ju však nie je schopný spracovať. Tento typ chyby je iba lokálny, preto sa klient môže pokúsiť preposlať žiadosť na iný server.
- **6xx - Globálna chyba (Global failure)**: žiadosť bola prijatá, ale server nie je schopný ju spracovať. Táto chyba však znamená, že žiaden server nie je schopný danú požiadavku spracovať, preto nemá zmysel preposlať žiadosť na iný server.

2.1.2 SIP transakcie a dialógy

V tejto časti sa bližšie oboznámime s transakciami a dialógmi, ktoré slúžia na logické zoskupovanie SIP správ. Uľahčujú tým kontrolu výmeny správ a riadenie celého hovoru.

V prípade protokolu SIP sa na transportnej vrstve väčšinou používa protokol UDP (User Datagram Protokol), ktorý však neposkytuje garanciu spoľahlivého doručenia správ. Pre korektné ustanovenie hovoru je však potrebné zaručiť, aby boli signalizačné správy medzi účastníkmi doručené. Pre tento účel sa používajú transakcie, čím sa protokol SIP radí medzi transakčné protokoly.

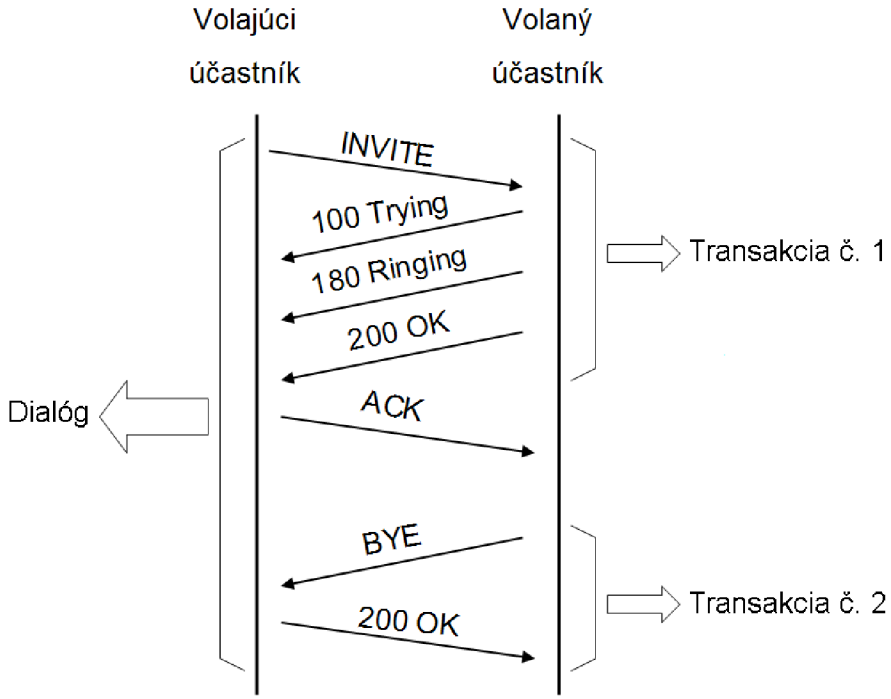
Signalizačné správy, ktoré sú cez sieť prenášané nezávisle na sebe, sú zoskupované na strane užívateľských agentov alebo proxy serverov do transakcií. Jednu transakcia predstavuje sekvenciu správ, ktoré si navzájom vymieňajú jednotlivé prvky siete SIP (napríklad užívateľský agent a proxy server, dva proxy servery medzi sebou atď.). Každá transakcia sa skladá z jednej správy typu žiadosť a všetkých odpovedí, ktoré sa k danej žiadosti vzťahujú. Jedinú výnimku predtavuje transakcia vytvorená INVITE správou, ktorá môže byť ukončená dvoma spôsobmi. V prípade úspešného vytvorenia hovoru sa za poslednú správu transakcie považuje správa typu 2xx zo strany volaného účastníka. Ak sa hovor vytvoriť nepodarí, do transakcie sa započítava aj posledná ACK správa zo strany volajúceho. Dôvodom pre takéto rozdielne zaobchádzanie s transakciou INVITE je dôležitosť doručenia správy 200 OK, kedy zodpovednosť za doručenie preberajú užívateľskí agenti.

Podľa toho, ako jednotlivé entity protokolu SIP pracujú s transakciami, je možné rozdeliť ich do dvoch skupín:

- **bez stavové (stateless)**: Ide hlavne o jednoduché implementácie proxy serverov, ktoré žiadosti iba preposielajú a neudržujú si žiadne informácie o stave jednotlivých transakcií.
- **stavové (stateful)**: Stav každej transakcie si ukladajú do pamäte a môžu napríklad opätovne zasielať jednotlivé správy, na ktoré neobdržali odpoveď v danom časovom limite.

Aby bolo možné sledovať stav jednotlivých transakcií, každej z nich je priradený jednoznačný identifikátor. Na základe tohto identifikátoru je každá správa priradená konkrétnej transakcii. Podľa predchádzajúceho štandardu bol identifikátor vypočítaný pre každú správu pomocou transformačnej funkcie, ktorá ako vstup používala polia *To*, *From*, *Request-URI* a *CSeq* z hlavičky správy SIP. Takéto spracovanie však bolo pri veľkom počte správ časovo náročné. Preto sa spôsob výpočtu v novej špecifikácii protokolu SIP zásadne zmenil. Každá správa už identifikátor transakcie obsahuje v hlavičke protokolu v poli *Via*.

Ako dialóg sa v prípade protokolu SIP označuje súbor správ medzi dvoma užívateľskými agentmi, ktoré majú vzájomnú súvislosť. Jednotlivé transakcie, ktoré sú medzi účastníkmi hovoru prináležia tomu istému dialógu. Číslo transakcie v rámci dialógu určuje pole *CSeq* zo SIP hlavičky. V každom smere môže byť vždy aktívna iba jedna transakcia. Dialóg teda môžeme nazvať aj ako postupnosť transakcií. Rovnako ako v prípade transakcie, musíme byť schopný jednozančne identifikovať každý dialóg. Používa sa nasledujúca trojica parametrov: *From*, *To* a *Call-ID*. Správy, ktoré majú uvedené tri parametre rovnaké, tak prináležia tomu istému dialógu. Súvislosť medzi transakciami a dialógom ilustruje nasledujúci obrázok:



Obrázek 2.2: Transakcie a dialóg protokolu SIP

Na nasledujúcom príklade je uvedený príklad hodnoty poľa *Via*. Identifikátor transakcie nasleduje za kľúčovým slovom *branch=* a od ďalších dát je oddelený vhodným znakom.

```
Via: SIP/2.0/UDP 192.168.0.3;branch=z9hG4bKc0a83801344dd123005a7900000017;
```

2.1.3 Adresácia v protokole SIP

Aby bolo možné zostavovať hovory medzi účastníkmi, musí byť každému účastníkovi priradená jednoznačná adresa SIP, ktorá ho identifikuje. Protokol SIP používa adresovanie vo formáte URI (Uniform Resource Indicator). Na nasledujúcom príklade je uvedený všeobecný formát používaných adries SIP URI:

```
sip:user:password@host:port;uri-parameters?headers
sips:user:password@host:port;uri-parameters?headers
```

Jediný rozdiel medzi *sip:* a *sips:* je v tom, že zdroj na danej *sips:* adrese môže byť kontaktovaný zabezpečeným spôsobom. Znamená to, že medzi klientom, ktorý chce nadviazať spojenie, a cieľom je možné použiť TLS (Transport Layer Security). Význam jednotlivých častí URI je nasledovný:

- **user**: Identifikuje konkrétny zdroj na danej adrese v sieti. Keďže v prípade protokolu SIP môžu byť adresované aj skupiny užívateľov, prípadne konkrétne zariadenia, nedá sa hovoriť priamo o mene užívateľa.
- **password**: Heslo priradené danému užívateľovi. Syntax adresy URI dovoľuje, aby heslo bolo priamo súčasťou adresy, ale je to považované ze veľké bezpečnostné riziko. URI je totiž prenášané ako čistý text a útočníkovi teda stačí zachytiť sieťovú komunikáciu a dostane sa k užívateľskému heslu.
- **host**: Špecifikuje konkrétneho hostiteľa adresovaného zdroja. Ako adresa môže byť použité tzv. kvalifikované doménové meno (Fully-qualified domain name) alebo IP adresa vo formáte IPv4 prípadne IPv6. Ak je to možné, doporučuje sa však použiť doménové meno.
- **port**: Udáva číslo portu, na ktorý má byť poslaná žiadosť.
- **uri-parameters**: Za číslom portu, ktoré uzatvára hostiteľskú časť URI, nasleduje zoznam parametrov, ktoré môžu ovplyvniť vytvorenie samotnej žiadosti. Za menom parametra nasleduje znak „=“ a hodnota parametra. Jednotlivé parametre sú navzájom oddelené znakom „;“.Takto je možné napríklad upresniť, aká verzia transportného protokolu sa má použiť, prípadne presne zadať hodnotu time-to-live. Kompletný zoznam parametrov spolu s príkladmi použitia je možné nájsť v [6].
- **headers**: Za znakom „?“ môžu nasledovať hodnoty, ktoré sa použijú v hlavičke výslednej žiadosti zostavenej z URI. Môže sa takto špecifikovať napríklad predmet zostavovanej relácie a jej priorita.

Na nasledujúcom príklade je uvedených niekoľko konkrétnych adries SIP URI:

```
sip:alice@atlanta.com
sip:alice:secretword@atlanta.com;transport=tcp
sips:alice@atlanta.com?subject=project%20x&priority=urgent
sip:alice@192.0.2.4
```

2.1.4 Komponenty protokolu SIP

Zariadenia využívajúce protokol SIP sa delia do dvoch základných kategórií:

- **užívateľskí agenti - user agents**: koncové zariadenia pre multimediálnu reláciu (v podstate ide o účastníkov hovoru)
- **servery - SIP servers**: servery pripojené do siete, ktoré majú za úlohu spracovávať požiadavky od klientov a posielať späť odpovede

Užívateľský agent je vo väčšine prípadov počítač s inštalovaným komunikačným programom podporujúcim hlasové a video hovory, prípadne instant messaging. Môže sa však jednať aj o PDA, hardwarový telefón pripojený k počítaču alebo hlasovú bránu, ktorá spája IP sieť s verejnou telefónnou sieťou (PSTN - Public Switched Telephone Network). Na rozdiel od SIP serverov, sú užívateľskí agenti povinné komponenty pre každú reláciu. V rámci relácie sa ešte rozlišuje agent klient (User agent client (UAC) - iniciuje reláciu a posiela žiadosti), a agent server (User agent server (UAS) - spracováva žiadosti a odosiela odpovede). Tieto úlohy sa však počas relácie často menia. Používajú sa iba pre lepšie rozlíšenie smeru komunikácie.

SIP server nie je nevyhnutne potrebný pre vytvorenie relácie, keďže agenti dokážu spolu komunikovať aj napriamo (pri dodržaní určitých podmienok ohľadom adresácie atď.), ale pri rozsiahlych implementáciach signalizácie SIP zastupujú svojimi funkciami klasické telefónne ústredne. Podľa poskytovanej služby sa SIP servery ďalej delia na:

- **Registračný server - registrar server**: udržuje zoznam aktuálne prihlásených užívateľov k SIP sieti. K užívateľovi priradí jeho aktuálnu IP adresu, ktorá sa môže meniť v závislosti od lokácie užívateľa. Táto informácia je potom použitá pri nadviazaní spojenia k danému užívateľovi.
- **Proxy server**: jeho hlavnou úlohou je posielať žiadosti namiesto koncového užívateľského agenta. V praxi však často poskytuje aj dodatočné služby, napríklad autentizáciu a autorizáciu užívateľov.
- **Smerovací server - redirect server**: na rozdiel od proxy servera nevytvára spojenia namiesto užívateľských agentov, ale iba poskytuje agentom iformáciu o tom, koho majú kontaktovať. V praxi sa často používajú v prípadoch, kedy je potrebné znížiť záťaž proxy serverov pri určitých typoch volaní.

Všetky uvedené typy serverov sú definované na logickej úrovni, čo znamená, že implementačne sa uvedené servery môžu nachádzať na rôznych fyzických zariadeniach, ale aj všetky služby môže poskytovať jeden fyzický server. Konkrétny spôsob implementácie už závisí na administrátoroch siete, na očakávanej úrovni záťaže a požadovanej redundancii.

2.1.5 Architektúra protokolu SIP

V závislosti na spôsobe spracovania SIP žiadostí môžeme rozlíšiť dve rôzne architektúry:

- **klient-server**
- **peer-to-peer**

V prípade architektúry *klient - server* sú úlohy jednotlivých účastníkov jasne vymedzené. Klient požaduje určitú službu alebo zdroj a posiela príslušné požiadavky (správy typu request). Server je vyhradený na spracovávanie žiadostí od klientov a zasielanie odpovedí (správy typu response). V prípade VoIP a protokolu SIP sa jedná napríklad o registráciu užívateľa na registračnom serveri. Klient pošle žiadosť o registráciu, ktorá obsahuje všetky potrebné informácie, následne server žiadosť spracuje (vloží užívateľa do databázy) a odošle odpoveď. Výhodou tejto architektúry je hlavne škálovateľnosť a nenáročná inštalácia nových užívateľov. Na druhej strane však treba myslieť na výpadok servera, ktorý automaticky postihne všetkých užívateľov. Je teda potrebné pri implementácii myslieť aj na redundanciu.

Architektúra *peer-to-peer* naopak nemá jasne vymedzené úlohy pre učastníkov. Platí, že všetci účastníci sú schopní zastávať funkciu klienta aj servera. Nedá sa však jednoznačne povedať, ktorý účastník bude klient a ktorý server, keďže úlohy sú v tomto prípade rovnocenné a môžu sa počas hovoru meniť. Výhodou tejto architektúry je vyššia odolnosť v prípade zlyhania niektorého účastníka. Naopak nevýhoda spočíva v náročnejšej konfigurácii agentov, lebo každý agent musí byť schopný zostaviť hovor nezávisle na ostatných a musí preto vopred poznať všetky potrebné informácie.

2.1.6 SDP - Session Description Protocol

Protokol SDP (Session Description Protocol) bol vytvorený v roku 1998 a popisuje ho RFC 2327 ([9]). Pôvodne mal slúžiť pre popis multicastových relácií prenášaných cez internetovú multicastovú sieť (MBONE). V podstate sa jedná o protokol, ktorý nedokáže pracovať samostatne, kedže nemá funkcionalitu potrebnú pre vyjednanie spojenia. Jeho správy sú však vhodné pre popis vlastností multimediálnych relácií. Hoci bol vyvinutý pre použitie s multicastom, neskôr sa začal používať hlavne v spojení s protokolom SIP.

Protokol SPD je tiež textový protokol, správa SDP pozostáva z viacerých riadkov, kde na každom riadku je jeden parameter relácie. Formát jednotlivých riadkov je vždy rovnaký - na začiatku je malé písmeno, ktoré reprezentuje skrátené meno parametru, nasleduje znak = a potom už vlastný parameter. Parametre sa delia do dvoch skupín:

- **povinné (mandatory)**
  - v=(protocol version): Špecifikuje verziu SDP protokolu.
  - o=(origin): Obsahuje meno užívateľa, ktorý vytvoril reláciu, identifikátor relácie, verziu relácie, verziu použitého internetového protokolu (IPv4, IPv6) a zdrojovú adresu relácie.
  - s=(session name): Udáva názov relácie.
  - t=(time): Obsahuje začiatok a koniec relácie vo formáte NTP. Toto pole však pri použití s protokolom SIP nemá v podstate žiaden význam, keďže sa dopredu nedá odhadnúť dĺžka relácie. Preto sa používajú hodnoty 0 0.
  - m=(media): Špecifikuje základné parametre prenášanej relácie. Ide hlavne o typ relácie (audio/video), číslo použitého portu transportnej vrstvy, protokol a typ kodeku.
- **voliteľné (optional)**: Voliteľné parametre sa používajú na podrobnejší popis relácie. Môžu napríklad obsahovať informácie o telefónnom čísle, emailovú adresu, informácie o type pripojenia a šírke pásma a podobne.

Použitie SDP protokolu spolu s protokolom SIP sa popisuje ako model ponuka/odpoveď (offer/answer model). Je to všeobecný model, teda je možné ho uplatniť aj v spojení SDP protokolu s inými aplikačnými protokolmi. Detailne ho popisuje RFC 3264 ([5]). Pri vytváraní hovoru volajúca strana vytvorí správu SDP, kde popíše svoje možnosti pre príjem relácie. Túto správu následne vloží do tela správy SIP INVITE, ktorá sa odošle volanej strane. Do tela odpovede (správa SIP s kódom 200) volaná strana vloží ďalšiu správu SDP, kde si z ponúkaných parametrov pre reláciu už vyberie konkétne nastavenia. Následne je už možné vytvoriť dátový prúd medzi účastníkmi. V prípade nezhody medzi parametrami, ktoré volaná a volajúca strana podporujú, nie je možné hovor ustanoviť a volaná strana odpovie na pozvanie správou BYE s príslušným chybovým kódom.

Presný popis jednotlivých parametrov SDP protokolu spolu s príkladmi použitia je možné nájsť v uvedených RFC dokumentoch, prípadne v literatúre [2] a [10].

2.2 Protokoly RTP a RTCP

Protokol RTP (Real-time Transport Protocol) definuje štandardizovaný formát paketov pre prenos multimediálnych dát (audio a video) cez IP sieť. Používa sa hlavne v systémoch pre videokonferencie a IP telefóniu. Samotný protokol RTP má na starosti iba prenos dát medzi užívateľmi a nie je schopný zaistiť dodatočné funkcie, ako zistenie kvality služieb (Quality of Service - QoS) a monitorovanie prenosu. Preto sa používa v spojení s protokolom RTCP (RTP Control Protocol), ktorý zabezpečuje spomenuté funkcie a je schopný aj synchronizovať viacero prúdov medzi užívateľmi. Protokoly RTP a RTCP pracujú nad transportným protokolom UDP (User Datagram Protocol), ktorý je vhodnejší pre prenos dát citlivých na rýchlosť doručenia. Ak sú oba protokoly použité súčasne, RTP používa párne číslo portu (konkrétny port je vyjednaný signalizáciou pred vytvorením) a RTCP najbližšie voľné nepárne číslo. Aktuálne platné implementácie oboch protokolov popisuje RFC 3550 ([4]).

Na nasledujúcom obrázku 2.3 je znázornená hlavička RTP paketu[1].

RTP packet header

bit offset	0-1	2	3	4-7	8	9-15	16-31
0	Version	P	X	CC	M	PT	Sequence Number
32	Timestamp						
64	SSRC identifier						
96	CSRC identifiers …						
96+32×CC	Profile-specific extension header ID						Extension header length
128+32×CC	Extension header …						

Obrázek 2.3: Hlavička RTP paketu

Popis jednotlivých polí:

- **version**: Špecifikuje verziu protokolu (aktuálne verzia je 2).

[1] Obrázok pochádza zo stránok Wikipedia.org.

- **P (padding)**: Používa sa v prípade, že paket RTP obsahuje na konci dodatočné zarovnanie. Zarovnaná veľkosť sa častokrát vyžaduje napríklad pri použití rôznych šifrovacích algoritmov.
- **X (extension)**: Indikuje či medzi štandardnú hlavičku RTP a dáta je vložená aj rozšírená hlavička. Použitie rozšírenej hlavičky závisí od konkrétnej aplikácie.
- **CC (CRSC count)**: Obsahuje počet identifikátorov. CRSC
- **M (marker)**: Používa sa ak je potrebné označiť prenášané dáta, ktoré majú pre aplikačný protokol špeciálny význam.
- **PT (payload type)**: Indikuje aký typ dát paket RTP prenáša. Konkrétne typ závisí na použitom profile RTP. Príklady bežne používaných profilov RTP sú uvedené v Dodatku CC.
- **sequence number**: Sekvenčné číslo má rovnaký význam ako pri protokole TCP. Používa sa na detekciu prípadnej straty dát alebo na usporiadanie dát do správneho poradia na strane príjemcu.
- **timestamp**: Časové razítka umožňujú príjemcovi prehrávať prijatý obsah v správnych intervaloch. Konkrétne hodnoty opäť závisia na aplikácii a profile RTP.
- **SSRC**: Obsahuje jednoznačný identifikátor zdroja dátového prúdu, ktorý sa používa pri synchronizácii na strane príjemcu.
- **CSRC**: V prípade, že dátový prúd je generovaný z viacerých zdrojov, toto pole obsahuje zoznam prispievajúcich účastníkov relácie.
- **Extension header**: Toto pole je voliteľné a obsahuje rozširujúce parametre, ktoré môžu byť použité špecifickými aplikáciami na rozšírenie funkcionality protokolu RTP.

Protokol RTP bol vytvorený s ohľadom na ľahkú rozšíriteľnosť podpory pre nové typy audio a video kodekov. Ako už bolo spomenuté vyššie pri popise hlavičky paketu RTP, typ prenášaných dát (konkrétny profil RTP) špecifikuje hodnota poľa *payload*. Na základe tejto hodnoty vie aplikácia na strane príjemcu určiť, aký kodek s akými parametrami je potrebné použiť pre správne dekódovanie dát. Kompletný popis aktuálne používaných RTP profilov je uvedený v RFC 3551 ([3]).

Primárnou funkciou protokolu RTCP je poskytovať spätnú väzbu na kvalitu služby konkrétneho toku RTP. Toho sa dosahuje periodickým zasielaním kontrolných správ medzi všetkými účastníkmi hovoru. Kontrolné správy obsahujú informácie o počte odoslaných a prijatých paketov, štatistiku stratených paketov, informácie o variabilite oneskorenia dát (jitter) a dobe prenosu medzi účastníkmi (round-trip delay time). Všetky informácie sú následne pravidelne vyhodnocované účastníkmi hovoru a v prípade, že kvalita služby nie je dostatočná, môže následne dôjsť k prerušeniu relácie, prípadne k zmene parametrov (použije sa kodek s nižšou kvalitou ale menšími nárokmi na šírku pásma).

2.3 Signalizácia

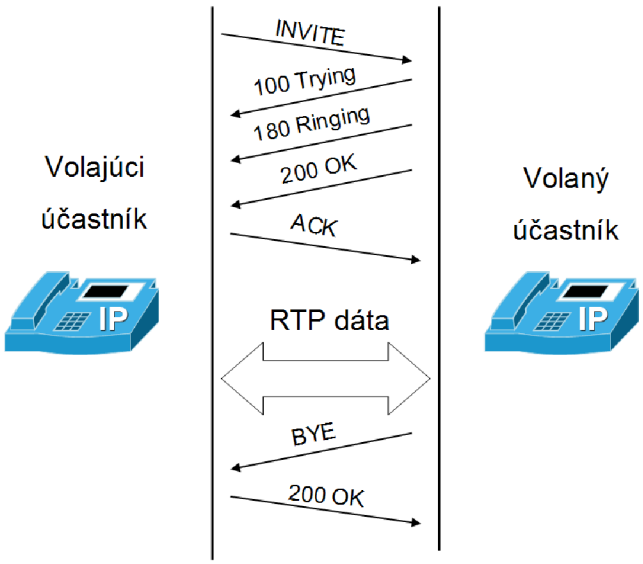
Po oboznámení sa s jednotlivými súčasťami protokolov SIP a RTP je vhodné ukázať na príkladoch celkový pohľad na signalizáciu pri vytváraní hlasového hovoru. Sú prezentované

iba dva príklady, ktoré sú však dostatočné pre pochopenie konceptu IP telefónie založenej na protokole SIP. Podrobnejšie ukážky vrátane výpisu jednotlivých správ SIP je možné nájsť v RFC 3665 ([1]), ktoré sa priamo zaoberá príkladmi signalizácie SIP pri vytváraní hovorov.

2.3.1 Komunikácia SIP peer-to-peer

Na obrázku 2.4 je znázornená postupnosť správ pri vytváraní hovoru priamo medzi dvoma účastníkmi. Tento príklad slúži hlavne na ilustráciu jednoduchého konceptu protokolu SIP. Na vytvorenie hovoru v tomto prípade totiž stačí iba päť správ:

1. **INVITE**: Úvodná správa, ktorá smeruje od UAC (volajúci účastník) ku UAS (volaný účastník). Obsahuje v tele správy popis navrhovaných parametrov relácie.
2. **100 Trying**: Informuje volajúceho, že INVITE správa bola prijatá a spracováva sa.
3. **180 Ringing**: Správa informuje o tom, že volaný účastník bol upozornený na prichádzajúci hovor (telefón začal zvoniť).
4. **200 OK**: Správa 200 OK potvrduje korektné vytvorenie spojenia medzi účastníkmi.
5. **ACK**: Na záver ešte volajúci posiela ACK správu, čím potvrdí príjem OK správy.



Obrázek 2.4: Signalizácia pri priamom nadviazaní hovoru

Po počiatočnej signalizácii nasleduje už nadviazanie samotného prúdu RTP. Adresy účastníkov hovoru a parametre dátového toku boli súčastou prvej správy INVITE. Počas samotného hovoru neprebieha žiadna signalizácia SIP, prenášajú sa iba dáta zabalené v paketoch RTP a účastníci si vymieňajú kontrolné informácie prostredníctvom správ RTCP. Ukončenie hovoru je tiež jednoduché. Strana, ktorá ako prvá ukončila spojenie pošle správu BYE a čaká na potvrdenie príjmu správou 200 OK. Po obdržaní potvrdenia dojde na oboch stranách k uvoľneniu alokovaných prostriedkov pre hovor.

Uvedený príklad je pomerne zjednodušený a v praxi sa priame nadväzovanie v podstate nepoužíva. Problémom je totiž zložitá konfigurácia užívateľských agentov, ktorí v takomto prípad musia poznať všetky potrebné informácie vopred. Hlavne je potrebné zabezpečiť správny preklad adries SIP URI na aktuálne platnú IP adresu užívateľa, čo pri väčších sieťach a dynamickom prideľovaní adries je takmer nemožné.

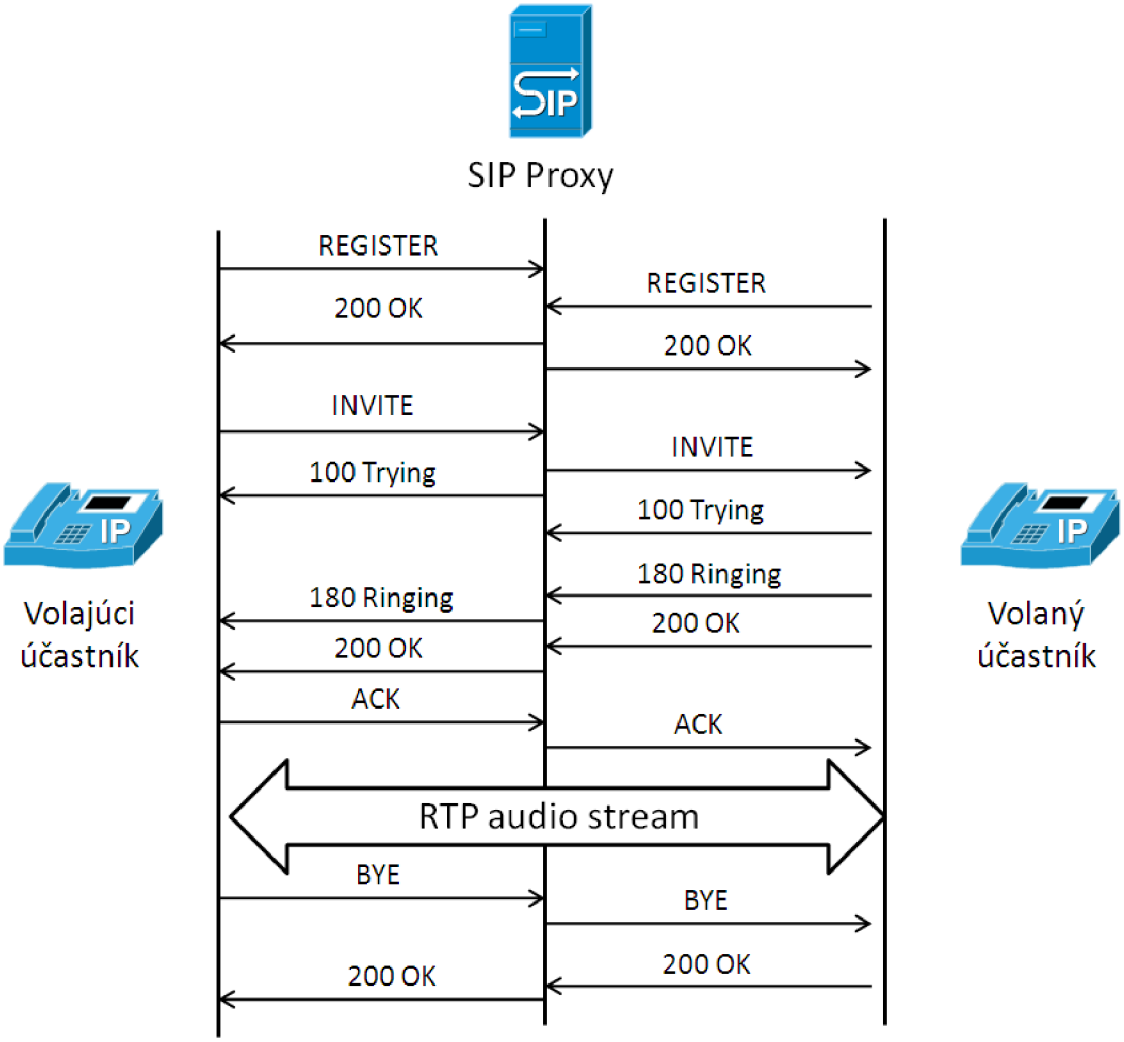
2.3.2 Komunikácia pomosou SIP proxy

V reálnom nasadení sa najčastejšie používa spojenie užívateľských agentov s registračným a proxy serverom SIP (funkcie serverov sú vysvetlené v sekcii 2.1.4). Príklad signalizácie pri vytváraní hovoru s využitím proxy SIP je na obrázku 2.5. V tomto prípade sú funkcie registračného a proxy servera zlúčené a zabezpečované jedným fyzickým zariadením.

Hlavný rozdiel oproti priamemu nadviazaniu spojenia je v tom, že užívateľskí agenti sa musia najskôr zaregistrovať. Na to slúži správa REGISTER, ktorá je po korektnom spracovaní potvrdená zo strany servera správou 200 OK. Počas registrácie si registračný server môže vyžiadať od klienta dodatočné informácie (napríklad meno a heslo). V takom prípade odmietne pôvodnú správu REGISTER, ako odpoveď pošle správu 401 UNAUTHORIZED. Následne musí klient opakovať žiadosť o registráciu a poskytnúť požadované informácie.

Po zaregistrovaní užívateľov prebieha následná signalizácia veľmi podobne ako pri priamom nadviazaní hovoru. Používajú sa tie isté správy, ale užívatelia v tomto prípade komunikujú s proxy serverom SIP, ktorý následne preposiela všetky správy volanej, prípadne volajúcej strane. Pri vyjednávaní parametrov spojenia sa však použijú koncové adresy jednotlivých účastníkov, takže dáta RTP budú adresované priamo.

Výhodou takejto architektúry je hlavne jednoduchá správa užívateľských agentov. Je potrebné nastaviť iba adresy registračného a proxy servera (častokrát jedna spoločná adresa). Táto výhoda sa prejaví pri korporátnom nasadení IP telefónie, kde je potrebné spravovať veľké množstvo užívateľských agentov. Nevýhodou je nutnosť inštalácie dodatočných serverov SIP. V dnešnej dobe je však na trhu veľké množstvo produktov, ktoré ponúkajú všetky potrebné funkcie a administrácia je možná cez rozhranie internetového prehliadača. V podnikovej sfére IP telefónia nahrádza klasické telefónne služby. Server SIP tak predstavuje náhradu za telefónnu ústredňu, ktorá slúžila na administráciu pôvodného riešenia.



Obrázek 2.5: Signalizácia pri nadviazaní hovoru cez SIP proxy

Kapitola 3

Nástroje pre sledovanie IP telefónie

V tejto kapitole si predstavíme najbežnejšie používané nástroje pre sledovanie IP telefónie. Keďže signalizácia prebieha s využitím komplexných protokolov, pre analýzu prípadných problémov je vhodné použiť niektorý špecializovaný nástroj, ktorý nám uľahčí prácu. Popíšeme si hlavne paketové analyzátory, ktoré dokážu automaticky rozpoznať bežne používané signalizačné protokoly. Rovnako dokážu pomocou vstavaných funkcií na analýzu dátového toku detekovať niektoré problémy v sieti (napríklad pri stratách paketov a hromadnom preposielaní).

Okrem paketových analyzátorov sme popísali aj niektoré nástroje na prácu so zvukovými dátami. Prevod zachytených paketov do hlasovej podoby môže uľahčiť riešenie problémov s nedostatočnou kvalitou prenosu hlasu. Zamerali sme sa hlavne na voľne dostupné nástroje, ktoré môže použiť každý bez obmedzenia.

3.1 Paketový analyzátor Wireshark

Wireshark je open source software, ktorý slúži na zachytávanie sieťových dát, ich analýzu a prípadný troubleshooting. Pôvodný projekt s názvom Ethereal bol premenovaný v roku 2006 a odvtedy vývoj pokračuje s pravidelným publikovaním nových stabilných verzií. Wireshark je v dnešnej dobe jedným z najviac používaných nástrojov pre sledovanie sieťovej prevádzky. Populárny je hlavne vďaka množstvu podporovaných funkcií a protokolov, jednoduchému grafickému užívateľskému rozhraniu a multiplatformnosti (nástroj je dostupný pre platformy Microsoft Windows, Unix, Linux, Mac OS, BSD).

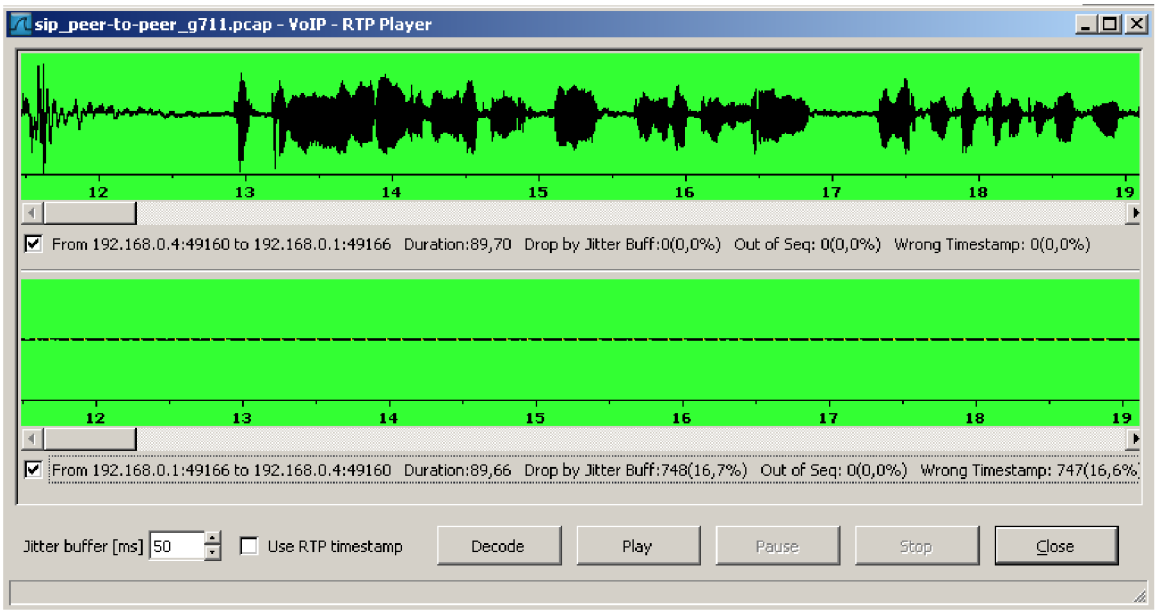
Z protokolov, ktoré sa používajú v IP telefónii, sú podporované nasledovné:

- **SIP**
- **H.323**
- **MGCP**

Analýza prebieha nad zachytenými dátami (on-line prípadne uloženými v súbore vo formáte pcap) a pre každý VoIP hovor sú dostupné rôzne štatistiky: čas kedy bol hovor vytvorený a ukončený, štatistiky počtu vyslaných a prijatých paketov, aktuálny stav hovoru a iné. Okrem vopred definovaných filtrov Wireshark umožňuje užívateľom definovať

aj vlastné filtre. Po zadaní parametrov hovoru sú výstupom už iba požadované dáta, ktoré je možné uložiť do súboru a pracovať s nimi aj v iných programoch.

Pri riešení problémov s IP telefóniou je možné využiť aj schopnosť Wiresharku zrekonštruovať určitý dátový tok RTP a uložiť ho vo forme zvukovej nahrávky. Takto sa dajú odhaliť problémy spojené s nesprávnou funkčnosťou hlasových kodekov. Aktuálne je táto funkcia podporovaná iba pre určité typy kodekov. Ukážka rozhrania použitého na prehrávanie streamov RTP je na obrázku 3.1.



Obrázek 3.1: Prehrávač RTP streamov

Kompletný popis všetkých podporovaných funkcií, spolu s ukážkami ich použitia a vzorkami testovacích dát, je možné nájsť v online dokumentácii ([8]).

3.2 Program Tcpdump

Tcpdump slúži rovnako ako Wireshark na zachytávanie a analýzu paketov. Neobsahuje však žiadne grafické užívateľské rozhranie, ovláda sa priamo z príkazového riadka a v mnohých linuxových distribúciach je priamo obsiahnutý ako sytémová utilita. Existuje aj alternatíva pre operačný systém Windows (WinDUMP), ale v praxi sa príliš nepoužíva. Ovládanie programu je možné iba cez parametre príkazového riadku, preto sa od užívateľov vyžadujú aspon základné znalosti práce v textovom prostredí. Komplentý popis všetkých parametrov programu Tcpdump sa nachádza na manuálových stránkach ([7]).

Nasledujúci príklad popisuje ako pomocou Tcpdump zachytiť signalizáciu SIP:

```
tcpdump -pi eth0 -s0 host 192.168.1.1 and udp port 5060
```

Uvedený príklad spôsobí, že na rozhraní *eth0* sa spustí promiskuitný mód (sieťová karta nekontroluje prichádzajúce dáta, ale automaticky ich posiela vyšším vrstvám na spracovanie) a začnú sa zachytávať celé pakety so zdrojovou alebo cieľovou IP adresou 192.168.1.1. Nás však zaujíma iba protokol SIP, preto je ešte špecifikovaný UDP port 5060 (východzí

port pre SIP). Uvedený príklad spôsobí, že obsah paketov sa bude priamo vypisovať na štandardný výstup. Ak chceme zachytávané pakety ukladať do súboru, použijeme parameter *-w* a zadáme názov výstupného súboru.

```
tcpdump -pi eth0 -s0 -w sip-capture host 192.168.1.1 and udp port 5060
```

Takto zachytené pakety sa dajú následne použiť k analýze prostredníctvom ďalších nástrojov. V porovnaní s nástrojom Wireshark však program Tcpdump nie je schopný rozoznať protokol RTP ktorý býva prenášaný pomocou transportného protokolu UDP. Wireshark je schopný vykonať túto detekciu pomocou inšpekcie dát signalizačných protokolov na aplikačnej úrovni.

3.3 Program Ngrep

Ngrep je paketový analyzátor, ktorý sa použitím veľmi podobá na program *Tcpdump* (neobsahuje žiadne grafické rozhranie a je ovládaný iba z príkazového riadka). Názov vznikol skrátením slovného spojenia *network grep*, ktoré najlepšie vystihuje jeho funkcionalitu. V podstate sa jedná o kombináciu paketového analyzátora tcpdump s nástrojom grep, ktorý slúži na vyhľadávanie textu v súboroch. Okrem filtrovania paketov podľa štandardných sieťových parametrov (IP adresa, protokol transportnej vrstvy, port atď.) pribudla možnosť vyhľadávania v dátach aplikačnej vrstvy s využitím regulárnych výrazov. V prípade textových protokolov (napríklad HTTP, DNS, SIP) je teda možné efektívne zachytávať iba špecifické pakety. Samozrejmosťou je možnosť ukladania dát do výstupného súboru vo formáte pcap, ako aj offline analýza zachytenej dátovej prevádzky.

Momentálne tento nástroj exituje vo verziách pre väčšinu operačných systémov a podporuje všetky bežne používané sieťové protokoly. Spustenie prebieha z príkazového riadka a ovládanie je pomocou jednotlivých parametrov, z ktorých najbežnejšie používané sú nasledovné:

-d interface : názov rozhrania, na ktorom sa majú zachytávať dáta

-I file : meno vstupného súboru vo formáte pcap

-O file : názov súboru, do ktorého sa zachytené dáta uložia v pcap kompatibilnom formáte

-W byline : špecifikuje spôsob výpisu zachytených paketov (možnosť *byline* v tomto prípade znamená, že dátový obsah paketu sa vypíše rozdelený na riadky)

-t : zobrazí časové známky vo formáte RRRR/MM/DD HH:MM:SS vždy keď niektorý paket bude zodpovedať zadaným kritériám

-T : v tomto prípade sa čas meria od spustenia programu a časové známky sa zobrazia v relatívnom formáte (výtup teda udáva časové rozostupy medzi zachytením jednotlivých paketov)

Okrem parametrov je možné zadať aj *vzor* a *filter*. Vzor je následne vyhľadávaný v aplikačných dátach, pričom okrem textovej reprezentácie je možné použiť aj regulárne výrazy zadané v štandardnom unixovom formáte. Filter slúži na špecifikovanie parametrov linkovej, sieťovej a transportnej vrstvy. Vhodnou kombináciou uvedených parametrov je tak možné získať aj z veľkého množstva analyzovaných dát prehľadný výstup o konkrétnych dátových tokoch. Na nasledujúcom príklade je ukázané, ako je možné s pomocou nástroja ngrep vyfiltrovať iba HTTP požiadavky typu GET na stránku www.google.com.

```
ngrep -d eth0 -W byline '^GET.*www.google.com' dst port 80
```

Ako vzor je použitý regulárny výraz, ktorý v paketoch s cieľovým portom 80 vyhľadáva také, ktoré majú na začiatku parameter GET (začiatok symbolizuje znak ^, čím zabránime zachyteniu paketov s výrazom GET v tele dát), za ktorým nasleduje ľubovolné množstvo znakov a potom reťazec www.google.com. Po spustení sa najskôr vypíšu aktuálne nastavenia programu a následne sa už priamo vypisujú zachytené pakety.

```
ngrep -d eth0 -W byline '^GET.*www.google.com' dst port 80

interface: eth0 (10.0.2.0/255.255.255.0)
filter: (ip or ip6) and ( dst port 80 )
match: ^GET.*www.google.com

T 10.0.2.15:49777 -> 209.85.149.106:80 [AP]
GET / HTTP/1.1.
Host: www.google.com.
User-Agent: Mozilla/5.0 Gecko/20100922 Ubuntu/10.10 (maverick) Firefox/3.6.10.
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8.
Accept-Language: en-us,en;q=0.5.
Accept-Encoding: gzip,deflate.
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7.
Keep-Alive: 115.
Connection: keep-alive.
```

Možnosť použiť regulárne výrazy podstatne rozširuje možnosti analýzy protokolu SIP. Je tak možné sledovať nielen globálne štatistiky (napríklad množstvo zasielaných INVITE alebo REGISTER správ) ale aj podrobné údaje týkajúce sa jednotlivých užívateľov, prípadne počítačov v sieti. Regulárnym výrazom so zadaným SIP menom užívateľa je možné získať prehľad o odchádzajúcich a prichádzajúcich hovoroch, používaných kodekoch, dĺžke hovorov a podobne.

Nasledujúce príklady ilustrujú použitie nástroja ngrep s ohľadom na protokol SIP. V prvom príklade sa zachytávajú iba INVITE a ACK správy, v druhom všetka signalizácia od užívateľa *alice@example.com* a v poslednom príklade sa do súboru *RTP-payload* vyexportujú dáta z prúdu RTP, ktorý prichádzal na počítač s IP adresou 192.168.10.1 a na port 20560.

```
ngrep -d eth0 -W byline '^INVITE|^ACK' dst port 5060
ngrep -d eth0 -W byline 'From.*<sip:alice@example.com>' dst port 5060
ngrep -d eth0 -O RTP-payload '' ip host 192.168.10.1 and dst port 20560
```

Ďalšie informácie o nástroji *ngrep*, spolu s detailným popisom jednotlivých parametrov a s ukážkami použitia, je možné nájsť na stránkach projektu http://ngrep.sourceforge.net/, prípadne na manuálových stránkach http://linux.die.net/man/8/ngrep.

3.4 Nástroj Cain & Abel

Predchádzajúce aplikácie sa radia do kategórie paketových analyzátorov a vhodným nastavením je možné použiť ich na analýzu hlasových hovorov. Aplikácia Cain & Abel sa však

primárne za paketový analyzátor nepovažuje. Najskôr slúžila ako nástroj pre obnovu stratených hesiel analýzou systémových registrov. Postupne však bola pridávaná ďalšia funkcionalita a z aplikácie Cain & Abel sa stal komplexný nástroj pre testovanie bezpečnosti počítačových sietí. Obsahuje totiž integrované nástroje pre odpočúvanie sieťových dát a automatické zisťovanie citlivých informácii (napríklad užívateľské mená a heslá, heslá pre prístup do bezdrôtovej siete). Okrem toho umožňuje testovať odolnosť sietí voči rozšíreným typom útokov.

V oblasti IP telefónie je funkcionalita zameraná hlavne na nahrávanie hovorov medzi užívateľmi. V prípade, že užívateľ používa softwarový IP telefón a chce nahrávať prichádzajúce alebo odchádzajúce hovory, stačí aplikáciu Cain & Abel nainštalovať a spustiť. Následne aplikácia sama rozozná prichádzajúci hovor a užívateľ môže jednoducho nastaviť nahrávanie.

V prípade nasadenia vo väčšej sieti, je princíp fungovania rovnaký ako v prípade paketových analyzátorov. Program analyzuje signalizačné protokoly a na základe získaných informácii nahráva hlasové hovory.

Výhodou programu je hlavne jednoduchosť obsluhy, keďže všetky nastavenia sú prostredníctvom grafického užívateľského rozhrania. Rovnako aj pri analýze vopred zachytených dát je proces plne automatický. Po otvorení súboru s dátami (podporovaný je formát kompatibilný s knižnicou pcap) sa automaticky zdetekujú hovory, prevedie sa extrakcia hlasových dát a prevod do audio súboru. Následne si užívateľ môže zachytené hovory prehrať.

Jediným obmedzením je, že program podporuje iba niektoré signalizačné protokoly. Momentálne sú podporované nasledujúce protokoly: SIP (Session Initiation protocol) a MGCP (Media Gateway Control Protocol). Z hľadiska podporovaných audio kodekov by užívateľ nemal naraziť na žiadne obmedzenie, keďže podporované sú všetky najrozšírenejšie typy.

3.5 Nástroje na spracovanie hlasových dát

V tejte sekcii si popíšeme niektoré nástroje, ktoré sú vhodné na spracovanie zachytených hlasových dát. Zo spomenutých paketových analyzátorov iba Wireshark obsahuje funkcie, ktoré dokážu za určitých podmienok hovor zrekonštruovať a následne prehrať. Podporované sú všetky rozšírené typy kodekov (G.711, GSM atď.). Nástroje popísane v tejto časti však na rozdiel od Wiresharku nepracujú so zachytenými paketmi, ale priamo s uloženými dátami. Spôsob, ako môžeme z prúdov RTP získať dáta je popísaný v nasledujúcej kapitole.

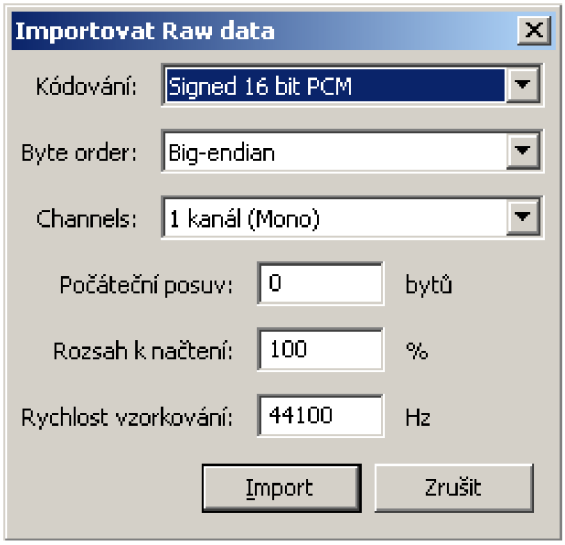
Program Audacity

Audacity je známy hlavne ako voľne dostupný program pre nahrávanie a editáciu zvuku. Medzi jeho hlavné výhody patrí jednoduché a prehľadné užívateľské rozhranie, široké možnosti úpravy zvuku pomocou vstavaných funkcií a dostupnosť pre všetky bežne používané operačné systémy.

S využitím rôznych knižníc Audacity umožňuje prekonvertovať jednotlivé audio formáty navzájom medzi sebou a dosiahnuť tak požadované parametre z hľadiska veľkosti výsledného záznamu, kvality, prípadne počtu zvukových kanálov. V prípade spracovania dát extrahovaných z prúdov RTP, ktoré neobsahujú žiadnu informáciu o formáte, môžeme využiť schopnosť importovať tzv. nespracovaných (raw) dát.

V systémovom menu sa táto možnosť nachádza pod voľbou *File*, následne je potrebné vybrať položku *Import* a z dostupných možností *Raw Data*. To nám umožní otvoriť v pod-

state ľubovolný súbor. Audacity v tomto prípade nekontroluje obsah súboru a po otvorení si od užívateľa vyžiada dodatočné informácie. Užívateľovi sa zobrazí ponuka podobná tej, ktorá je znázornená na obrázku 3.2.



Obrázek 3.2: Nastavenie parametrov importovaných dát

Aby mohli byť importované dáta správne dekódované, musí užívateľ poznať typ použitého kodeku spoločne s dodatočnými informáciami o počte kanálov a rýchlosti vzorkovania. Neexistuje totiž spôsob, ako tieto informácie dostať priamo zo súboru. Nasleduje už len dekódovanie vstupného súboru na základe špecifikovaných parametrov, po ktorom si užívateľ môže zvukovú stopu prehrať, uložiť, prípadne s ňou ďalej pracovať. V prípade zachytených hovorov je výhodne použiť funkciu synchronizovaného prehrávania viacerých zvukových stôp. Po načítaní oboch hlasových prúdov dát je tak možné hovor automaticky prehrať v stave, v akom sa odohral v skutočnosti.

Nástroj SoX

Nástroj SoX (názov predstavuje skrátené spojenie *Sound Exchange*) bol vytvorený špeciálne za účelom konvertovania medzi jednotlivými audio formátmi. Podobne ako v prípade Audacity sa jedná nástroj podporovaný na viacerých operačných systémoch. Neobsahuje však grafické užívateľské rozhranie. Ovládanie je priamo z príkazového riadka. S využitím dostupných hardwarových prostriedkov dokáže zvukové súbory aj nahrať a následne prehrať. Jeho hlavnou výhodou je široká podpora audio formátov bez nutnosti inštalácie dodatočných zásuvných modulov a knižníc. Nasledujúci príklad demonštruje použitie programu.

Vo vstupnom súbore `input-stream.raw` je uložený záznam hovoru s nasledujúcimi parametrami:

- **použitý kodek**: G.711 PCMA (štandard pre Európu)
- **počet kanálov**: 1
- **rýchlosť vzorkovania**: 8000 Hz
- **veľkosť vzorky**: 8 bitov

Uvedený súbor môžeme prekódovať do formátu .wav príkazom:

```
sox -r 8k -e al -b 8 -c 1 input-stream.raw input-stream.wav
```

Vysvetlenie jednotlivých parametrov programu:

- **-r**: udáva rýchlosť vzorkovania, podporované sú hodnoty od 8 kHz (používané hlavne v IP telefónii) až po 96 kHz (profesionálne audio systémy)
- **-e**: špecifikuje použitý kodek (v našom prípade skratka `al` znamená kodek G.711 A-Law)
- **-b**: veľkosť vzorky v bitoch
- **-c**: počet zvukových kanálov

Po dekódovaní vstupu môžeme súbor z nekomprimovaného formátu .wav previesť do niektorého formátu s použitím kompresie (napríklad formát .ogg) a následne prehrať.

```
sox input-stream.wav input-stream_compressed.ogg
```

```
play input-stream_compressed.ogg
```

Uvedený príklad iba stručne demonštruje použitie programu. Vzhľadom na ovládanie aplikácie priamo z príkazového riadka, je program SoX vhodný najmä pre automatizované spracovanie veľkého množstva vstupných súborov. Kompletný popis všetkých podporovaných možností kódovania spolu so zoznamom podporovaných zvukových formátov je možné nájsť na manuálových stránkach programu http://sox.sourceforge.net/sox.html.

3.6 Zhrnutie

V tejto kapitole sme sa zaoberali popisom nástrojov, ktoré sú vhodné pre analýzu sieťových dát s ohľadom na IP telefóniu. Ukázali sme, že existuje pomerne široké spektrum voľne dostupných programov, ktoré poskytujú dostatočnú funkcionalitu pre základnú analýzu signalizačných protokolov. Pre užívateľa bude asi najviac vhodný nástroj Wireshark, ktorý ponúka možnosti detailnej analýzy väčšiny bežne používaných sieťových protokol. Je vhodný aj vzhľadom na prítomnosť grafického rozhrania, ktoré uľahčuje prácu a poskytuje prehľadné výstupy vo forme grafov. Pre skúsených administrátorov, ktorí požadujú možnosť práce v spojení s automatickými skriptami, sa ako najvhodnejší javí nástroj Tcpdump.

V tabuľke 3.6 je uvedené porovnanie popísaných nástrojov vzhľadom k vybraným parametrom. Nástroje pre prácu so zvukom nie sú súčasťou porovnania, keďže ich funkcionalita je v porovnaní s paketovými analyzátormi úzko špecializovaná.

Motiváciu pre vývoj nového nástroja je fakt, že vyššie uvedené nástroje potrebujú pre prácu so signalizáčným protokolom SIP určitú konfiguráciu od užívateľa. Našim zámerom je vytvoriť nástroj, ktorý by automaticky rozoznal hlasové hovory a prípadne by bol schopný zo zachytených paketov protokolu RTP získať hlasové dáta. Ovládanie programu by malo byť čo najjednoduchšie, s ohľadom na presne danú požadovanú funkcionalitu. V nasledujúcej kapitole si popíšeme návrh takéhoto nástroja.

	Wireshark	**Tcpdump**	**Ngrep**	**Cain & Abel**
podporované protokoly	SIP, H.323, MGCP, RTP	SIP, H.323	SIP, H.323	SIP, MGCP
analýza VoIP	áno	nie	nie	nie
analýza RTP	áno	nie	nie	nie
prehrávanie RTP	áno	nie	nie	áno
grafické rozhranie	áno	nie	nie	áno
možnosť skriptovania	nie	áno	áno	nie
on-line analýza	áno	áno	áno	áno
off-line analýza	áno	áno	áno	áno
platforma	Windows, Linux, Unix, Mac OS	Linux Unix	Linux Unix	Windows

Tabulka 3.1: Porovnanie uvedených nástrojov

Kapitola 4

Návrh aplikácie

Táto kapitola sa zaoberá návrhom aplikácie, ktorá bude slúžiť na detekciu VoIP hovorov založených na signalizačnom protokole SIP v rámci dátového toku. Najskôr však musíme určiť, ktoré informácie sú potrebné z hľadiska účtovania a ako ich získať z príslušných správ SIP. Samotná inšpekcia protokolu SIP však nie je dostatočná, keďže vlastný prenos hlasu už má na starosti protokol RTP. Fungovanie aplikácie môžeme rozdeliť do nasledujúcich troch fáz:

- inšpekcia protokolu SIP počas ustanovenia hovoru
- zachytenie hlasových dát
- ukončenie hovoru

Analýza signalizačných paketov na sieťovej vrstve v tomto prípade nie je dostatočná, keďže zdrojové a cieľové adresy vo väčšine prípadov neodpovedajú IP adresám jednotlivých účastníkov. Rovnako dáta zo sieťovej (a prípadne aj transportnej) vrstvy neposkytujú dostatok informácií, preto je potrebné analyzovať aplikačné dáta protokolu SIP. Je teda vhodnejšie hovoriť o inšpekcii než o analýze signalizačných správ.

V prípade zachytávania hlasových dát budeme vychádzať z informácií získaných v predchádzajúcej inšpekcii. Rovnako je potrebné nájsť vhodný spôsob, ako hlasové dáta uložiť s ohľadom na ich následné spracovanie. Kedže analýza protokolu RTP bude prebiehať nezávisle od inšpekcie protokolu SIP, musíme byť na základe vhodne zvolenej metódy priradiť každý prúd hlasových dát ku konkrétnemu hovoru.

V poslednej fáze je potrebné rozoznať signalizáciu ukončenia aktuálneho hovoru, ukončiť zachytávanie príslušných dátových prúdov, zachytené dáta uložiť a vygenerovať štatistiky o práve ukončenom hovore. Počas hovoru sa všetky dáta ukladajú v pamäti, ktorú môžeme po ukončení uvoľniť, aby sme zbytočne nespotrebovávali systémove prostriedky.

V tejto kapitole sa postupne budeme zaoberať jednotlivými fázami a ukážeme si, ako naimplementovať požadovanú funkcionalitu vhodným spôsobom.

4.1 Inšpekcia protokolu SIP počas ustanovenia hovoru

Keďže po spustení programu nemáme žiadne informácie o aktuálnom stave, prvou úlohou je rozoznať v dátovom toku požadované dáta. Podľa štandardu sa pre prenos správ protokolu SIP používa transportný protokol UDP a port 5060. V prípade šifrovaného prenosu sa používa port 5061, ale v našom projekte sa budeme zaoberať iba nešifrovaným prenosom.

Spomenutý port 5060 je aj zaregistrovaný u organizácie IANA (Internet Assigned Numbers Authority) a teda vyhradený iba pre protokol SIP. Zo začiatku bude preto postačovať zamerať sa na pakety so zdrojovým a cieľovým portom 5060. Teraz už môžeme pristúpiť k samotnej inšpekcii protokolu SIP. Najskôr si musíme stanoviť, ktoré informácie sú zaujímavé z hľadiska účtovania hlasových hovorov.

Aby sme získali prehľad o uskutočnených hovoroch, je dôležité vedieť, ktorý účastník hovor vytvoril, následne komu sa snažil dovolať, kedy sa hovor uskutočnil a ako dlho trval. Tieto informácie by boli dostatočné v prípade klasickej telefónie. Pri prenose hlasu cez IP sieť je potrebné sledovať aj ďalšie parametre, ako sú adresy IP a čísla portov, ktoré nám následne umožnia zahytiť jednotlivé hlasové hovory. Pri každom hovore teda budeme sledovať nasledujúce parametre:

- identifikátor volajúceho účastníka
- identifikátor volaného účastníka
- IP adresu priradenú volajúcemu
- IP adresu priradenú volanému
- čas vytvorenia hovoru
- čas ukončenia hovoru
- dĺžku trvania hovoru
- číslo zdrojového portu
- číslo cieľového portu
- použitý kodek

Aby sme mohli sledovať väčšie množstvo hovorov súčasne, musíme nájsť spôsob, ako jednoznačne priradiť každú signalizačnú správu k hovoru. Na nasledujúcom príklade je znázornený obsah správy typu INVITE, ktorou volajúci účastník zahájí signalizáciu smerom k volanému. Uvedená správa je podstatne zjednodušená, ale obsahuje všetky potrebné informácie, ktoré potrebujeme pre účtovanie. Príklad je prevzatý z RFC 3665 ([1]) a kompletný výpis všetkých signalizačných správ je uvedený v prílohe B.

```
INVITE sip:bob@biloxi.example.com SIP/2.0
Via: SIP/2.0/TCP client.atlanta.example.com:5060;branch=z9hG4bK74bf9
Max-Forwards: 70
From: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
To: Bob sip:bob@biloxi.example.com
Call-ID: 3848276298220188511@atlanta.example.com
CSeq: 1 INVITE
Contact: sip:alice@client.atlanta.example.com;transport=tcp
Content-Type: application/sdp
Content-Length: 151
v=0
o=alice 2890844526 2890844526 IN IP4 client.atlanta.example.com
s=-
```

```
c=IN IP4 192.0.2.101
t=0 0
m=audio 49172 RTP/AVP 0
a=rtpmap:0 PCMU/8000
```

Správa v uvedenom príklade má dve časti: hlavičku a telo, ktoré obsahuje správu protokolu SDP. V hlavičke sa nachádzajú základné informácie o účastníkoch hovoru (kto komu volá) a v poli Call-ID sa nachádza identifikátor, ktorý je unikátny pre daný hovor. Je to v podstate kombinácia náhodného reťazca a mena užívateľa, prípadne IP adresy. Reláciu medzi účastníkmi však jednoznačne identifikuje trojica parametrov z hlavičky SIP správy: Call-ID, To tag a From tag. Z hlavičky teda použijeme nasledujúce polia:

- **Call-ID**: obsahuje reťazec, ktorý priradí signalizačnú správu ku konkrétnemu hovoru
- **From**: obsahuje identifikáciu volajúceho účastníka (užívateľské meno a SIP URI)
- **To**: obsahuje identifikáciu volaného účastníka (užívateľské meno a SIP URI)

Z ostatných polí sú zaujímave ešte nasledovné:

- **Via**: špecifikuje adresu, na ktorú sa má poslať odpoveď na žiadosť. V prípade, že signalizácia prechádza cez viacej proxy serverov SIP sa toto pole vždy zmení s ohľadom na to, ktorému serveru sa má odpoveď preposlať.
- **Cseq (Command Sequence)**: okrem názvu metódy obsahuje aj číslo, ktoré udáva poradie žiadosti v rámci dialógu (v podstate ide o sekvenčné číslo). Tento parameter je vhodné sledovať pri komplexných reláciach, prípadne pri problémoch v sieti, kedy odhalí opakované zasielanie jednotlivých žiadostí.

Pre identifikáciu relácie však informácie z hlavičky správy SIP nestačia. Detailný popis relácie v podstate ani nie je možný, keďže polia v hlavičke sú limitované a nie je možné popísať všetky potrebné parametre. Preto sa využíva protokol SDP, ktorý bol vytvorený pre popis multimediálnych relácii. Neposiela sa však samostatná správa s navrhovanými parametrami, ale namiesto toho sa správa SDP vloží do tela správy SIP. Z nej môžeme následne získať detaily o prúde RTP, ktorý bude prenášať hlasové dáta. Jedná sa však iba o návrh parametrov, ktoré ešte musí potvrdiť volný účastník. Z uvedeného príkladu sú z hľadiska účtovania dôležité nasledujúce informácie:

- **IP adresa užívateľa**: nachádza sa v políčku c= spolu s identifikáciu typu adresy. V našom prípade je aktuálna adresa užívateľa Alice 192.0.2.101 a je typu IPv4.
- **popis multimediálnej relácie**: navrhované paremetre pre reláciu na prenos hlasu sú špecifikované pod parametrom m=. Keďže každý parameter má špeciálny význam, je vhodné vysvetliť každý parameter z uvedeného príkladu:
  - **Audio**: prvý parameter uvádza typ prenášanej relácie. V našom prípade ide iba o prenos zvuku. Okrem zvuku sa však môže prenášať aj video, prípadne aplikačné dáta.
  - **49172**: ako druhý parameter nasleduje číslo transportného portu, na ktorý sa zašle prúd dát. V prípade prenosu cez protokol UDP, by táto hodnota mala byť v rozmedzí od 1024 do 65535.

– **RTP/AVP**: ktorý protokol sa má použiť na transportnej vrstve udáva tretí parameter. Keď je spojenie typu IPv4 tak vo väčšine prípadov sa použije protokol UDP. V našom prípade hodnota RTP/AVP znamená Realtime Transport Protocol (prenášaný cez UDP) s použitím Audio/Video profilov.
– **0**: Posledný parameter udáva, ktorý profil RTP sa má použiť. V podstate ide o typ kodeku, ktorý bude dáta kódovať. Kompletný zoznam podporovaných profilov, spolu s detailnými informáciami (rýchlosť vzorkovania, nároky na prenosové pásmo atď.) je uvedený v RFC 3551. V prílohe sa nachádza zoznam najčastejšie použitých profilov pre IP telefóniu. V uvedenom príklade hodnota 0 znamená profil pre kodek G.711 u-law, ktorý sa bežne používa v Severnej Amerike.

Teraz vieme, ako zo signalizačnej správy INVITE získať potrebné informácie pre účtovanie hovorov. To však ešte nie je dostačujúce. Správou INVITE sa proces signalizácie iba začína a parametre pre reláciu predstavujú iba návrh, ktorý ešte musí potvrdiť volaný účastník. Rovnako po analýze správy INVITE ešte nepoznáme adresu IP volaného účastníka a transportný port. Chýbajúce informácie sa na strane volajúceho užívateľa doplnia až po prijatí správy 200 OK od volaného užívateľa (kompletný sled správ je na obrázku 2.4). Rovnako ako správa INVITE, obsahuje aj správa 200 OK vo svojom tele popis relácie vo formáte SDP, ktorou volaná strana doplní potrebné informácie a potvrdí navrhované parametre (hlavne typ použitého kodeku). Na nasledujúcom príklade je uvedená správa, ktorou sa vytvorí hovor iniciovaný správou INVITE z predchádzajúceho príkladu:

```
SIP/2.0 200 OK
Via: SIP/2.0/TCP client.atlanta.example.com:5060;branch=z9hG4bK74bf9;
From: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
To: Bob <sip:bob@biloxi.example.com>;tag=8321234356
Call-ID: 3848276298220188511@atlanta.example.com
CSeq: 1 INVITE
Contact: sip:bob@client.biloxi.example.com;transport=tcp
Content-Type: application/sdp
Content-Length: 147
v=0
o=bob 2890844527 2890844527 IN IP4 client.biloxi.example.com
s=-
c=IN IP4 192.0.2.201
t=0 0
m=audio 3456 RTP/AVP 0
a=rtpmap:0 PCMU/8000
```

Hlavička protokolu SIP sa vo všetkých sledovaných poliach zhoduje so správou INVITE. Rozdiely sa nachádzajú až v správe protokolu SDP, kde je špecifikovaná adresa IP priradená volanému užívateľovi a číslo portu transportnej vrstvy. Obaja užívatelia teraz vzájomne poznajú svoje IP adresy a čísla portov, medzi ktorými majú nadviazať dátové prúdy RTP. Vytvorenie hovoru sa ešte potvrdí správou ACK zo strany volajúceho a hovor je nadviazaný. Signalizácia je následne ukončená a hovor medzi účastníkmi prebieha bez dodatočných SIP správ.

Analýzou správy INVITE a 200 OK získame dostatok informácii potrebných pre účtovanie. Rovnako poznáme všetky potrebné parametre pre zachytenie, uloženie a dekódovanie

hlasových dát. Z hľadiska účtovania však potrebujeme ešte informáciu u čase, kedy bol hovor vytvorený a ako dlho trval. Správy protokolu SIP v sebe nenesú žiadnu informáciu o čase, keďže z pohľadu signalizácie nie je potrebná. Môžeme však využiť knižnicu Libpcap, ktorú použijeme pri zachytávaní paketov priamo zo sieťovej karty. Každý paket, ktorý je zachytený pomocou Libpcap knižnice, sa ukladá do štruktúry, ktorá pozostáva z nasledujúcich dvoch častí:

- **dáta paketu**: v tomto prípade je označenie mierne zavádzajúce, lebo v podstate sa jedná o kompletný rámec zachytený na linkovej vrstve v rovnakej podobe, ako bol vyslaný
- **hlavička paketu**: ako hlavička sa sa označujú dáta, ktoré sú pridané ku každému paketu pri jeho zachytení na sieťovom rozhraní. Z hlavičky sa dajú získať nasledujúce údaje:
  - **časová známka v sekundách**: dátum a čas, kedy bol zachytený paket. Hodnota je udaná ako počet sekúnd od 1.1.1970 (takzvaný Unixový čas) a pre vhodnejšiu reprezentáciu je potrebné skonvertovať ju použitím vhodnej funkcie.
  - **časová známka v mikrosekundách**: upresňuje predchádzajúcu hodnotu na mikrosekundy
  - **veľkosť zachytených dát v bytoch**
  - **skutočná veľkosť paketu v bytoch**: v prípade, že sa skutočná veľkosť paketu odlišuje od zachytenej veľkosti, je veľkosť zachytávaných dát obmedzená na systémovej úrovni

Pri každom pakete teda najskôr z hlavičky formátu pcap získame informáciu o čase zachytenia a následne môžeme analyzovať vlastný obsah paketu.

4.2 Zachytávanie hlasových dát

Po získaní všetkých informácii popísaných v predchádzajúcej časti, môžeme začať sledovať jednotlivé hlasové prúdy dát, prenášané pomocou protokolu RTP. Rozoznať protokol RTP v dátach na sieti však vôbec nie je jednoduchá úloha. Pri prenose dát sa totiž ako protokol transportnej vrstvy použije protokol UDP. Hlavička protokolu RTP nasleduje za hlavičkou protokolu UDP, ktorá však neobsahuje žiadne pole, ktoré by identifikovalo, aký protokol sa prenáša na vyššej vrstve.

Určité nástroje (napríklad Wireshark) dokážu odlíšiť RTP pakety od ostatných dát. Využívajú však inšpekciu aplikačných protokolov, pri ktorej na pozadí automaticky analyzujú signalizačné protokoly a snažia sa získať čísla portov, ktoré sa potom použijú pre multimediálne prenosy. V prípade, že signalizačné správy nie sú zachytené, program nie je schopný rozoznať, či protokol UDP prenáša multimediálne dáta (s využitím protokolu RTP) alebo bežné dáta.

Náš program bude využívať rovnaký prístup. Počas prvej fázy, pri ktorej budeme sledovať iba protokol SIP, získame okrem informácií o zdrojových a cieľových IP adresách prúdov RTP aj čísla portov na transportnej vrstve. Po ustanovení hovoru sa medzi volajúcim a volaným účastníkom automaticky nadviažu dva nezávislé prúdy RTP. Každý z nich prenáša jednosmerne hlasové dáta. Každý zachytený paket musíme jednoznačne priradiť k práve prebiehajúcemu hovoru a ešte ho musíme začleniť do správneho prúdu dát RTP. Môžeme

však použiť iba informácie získané zo sieťovej a transportnej vrstvy. Prenos hlasu totiž funguje nezávisle na signalizačnom protokole, preto jednotlivé pakety neobsahujú žiadne informácie, na základe ktorých by ich bolo možné priradiť k signalizácii.

Ku každému aktívnemu hovoru teda poznáme IP adresy a čísla portov volajúceho aj volaného účastníka. V nasledujúcej tabuľke je uvedené, ako môžeme pomocou nich identifikovať jednotlivé hlasové prúdy z predchádzajúceho príkladu.

	prúd RTP od Alice k Bob	**prúd RTP od Bob k Alice**
zdroj. IP adresa	192.0.2.101	192.0.2.201
cieľ. IP adresa	192.0.2.201	192.0.2.101
zdroj. číslo portu	49172	3456
cieľ. číslo portu	3456	49172

Tabulka 4.1: Parametre jednotlivých RTP prúdov hlasových dát

Teraz už vieme, ako identifikovať jednotlivé prúdy. Aby sme mohli následne rekonštruovať hlasový záznam hovoru, musíme najskôr z RTP paketov dátá získať a potom vo vhodnej forme uložiť. V tomto prípade predstavujú hlavičky paketov nadbytočné informácie, ktoré treba odstrániť. Neuvažujeme hlavičku linkovej vrstvy, ktorú musíme odstrániť ešte pred dotazom na zdrojovú a cieľovú IP adresu. Aby sme sa dostali k vlastným dátam, musíme teda ostrániť nasledujúce časti paketu:

- **hlavičku IP protokolu**: Minimálna veľkosť je 20 bajtov, ktorá však môže narásť pri použití IP options až na 160 bajtov. Preto hlavička IP obsahuje pole *Header Length*, z ktorého je možné zistiť celkovú dĺžku hlavičky IP daného paketu.
- **hlavičku UDP protokolu**: V prípade protokolu UDP sú polia obsiahnute v hlavičke protokolu pevne dané, teda aj veľkosť hlavičky je vždy rovnaká - 8 bajtov.
- **hlavičku RTP protokolu**: Veľkosť hlavičky protokolu RTP tiež nie je pevne určená. Minimálna dĺžka je 12 bajtov. Niektoré aplikácie však používajú dodatočné polia CSRC, ktoré zväčšujú celkovú veľkosť vždy o 4 bajty. Celkovú veľkosť hlavičky ale je možné dopočítať, keďže počet polí CSRC udáva ďalšie pole CC.

Po odstránení uvedených hlavičiek dostaneme už priamo hlasové dáta zakódované príslušným kodekom. Každý paket sme teda schopní priradiť k hlasovému prúdu a následne z neho extrahovať dáta. Po vhodnom spojení takto získaných dát získame kompletný záznam hovoru v každom smere. Kedže dáta nebudú obsahovať žiadne pridané údaje, sú vhodné k následnému spracovaniu pomocou vhodného audio programu.

4.3 Ukončenie hovoru

Správna detekcia ukončenia hovoru je dôležitá hlavne pre korektné ukončenie zachytávania paketov RTP. Z hľadiska účtovania je rovnako dôležité poznať presný čas, kedy bol hovor ukončený. Z rozdielu časov vytvorenia a ukončenia následne určíme dĺžku hovoru.

Na rozdiel od vytvorenia hovoru, pri ktorom je potrebných viacej signalizačných správ, ukončenie prebieha poslaním iba jednej žiadosti s textom *BYE* a následným potvrdením správou *200 OK*. Na obrázku 2.4 sú správy potrebné pre korektné ukončenie hovoru znázornené v kompletnom kontexte spolu s vytvorením hovoru.

Po ukočení hovoru je potrebné uložiť vo vhodnom formáte zachytené hlasové dáta a vygenerovať do výstupného súboru popis uskutočneného hovoru. Popis musí obsahovať všetky sledované parametre, spolu s priradením príslušných hlasových dát. Formát výstupu musí byť prehľadný a vhodný pre následné spracovanie. Aby sme mohli jednotlivé hovory od seba jednoducho odlíšiť aj v prípade, že medzi rovnakými účastníkmi bude viacej nezávislých hovorov, bude vhodné zaviesť jednotný globálny identifikátor hovoru v rámci našej aplikácie. V nasledujúcej kapitole je popísané, ako boli uvedené kritériá zohľadnené pri implementácii aplikácie.

4.4 Inšpekcia registrácie užívateľov

Okrem účtovania uskutočnených hlasových hovorov je vhodné mať prehľad aj o registráciách jednotlivých užívateľov. Pri nasadení protokolu SIP v komplexnom prostredí, s veľkým počtom užívateľov je nemožné dosiahnuť stav, pri ktorom by každý užívateľský agent poznal všetky aktuálne IP adresy ostatných užívateľov. Adresy sa totiž môžu meniť v závislosti na pohybe užívateľa medzi rôznymi lokalitami, prípadne v závislosti na použitom type telefónu (hardwarový vs. softwarový).Registrácia je proces, pri ktorom užívateľ oznámi registračnému serveru svoje SIP URI a aktuálnu IP adresu. Vznikne tak vzájomná väzba medzi identifikátorom užívateľa, ktorý má užívateľ pridelený od poskytovateľa služby a je teda nemenný, a IP adresou, ktorá sa môže meniť.

Proces registrácie užívateľa je pomerne jednoduchý. Užívateľský agent pošle registračnému serveru žiadosť typu REGISTER. Adresu servera (prípadne doménové meno) musí mať agent vopred nastavenú. Registračný server žiadosť spracuje a odpovie správou s kódom 200 v prípade, že žiadosť bola akceptovaná. Ak server požaduje dodatočné overenie užívateľa (zadanie mena a hesla), odpovie správou s kódom 401 (status Unauthorized). V takomto prípade musí užívateľ registračnú žiadosť zaslať opakovane, spolu s chýbajúcimi údajmi. Na nasledujúcom príklade je uvedený príklad registračnej správy.

```
REGISTER sips:ss2.biloxi.example.com SIP/2.0
Via: SIP/2.0/TLS client.biloxi.example.com:5061;branch=z9hG4bKnashds7
Max-Forwards: 70
From: Bob <sips:bob@biloxi.example.com>;tag=a73kszlfl
To: Bob <sips:bob@biloxi.example.com>
Call-ID: 1j9FpLxk3uxtm8tn@biloxi.example.com
CSeq: 1 REGISTER
Contact: <sips:bob@client.biloxi.example.com>
Content-Length: 0
```

Z hľadiska účtovania nás zaujímajú hlavne nasledovné informácie:

- **identifikátor registrovaného užívateľa**
- **kontaktná adresa registrovaného užívateľa**
- **doménové meno registračného serveru**
- **IP adresa registračného serveru**
- **čas registrácie**

Prvé tri uvedené parametre sme schopní získať priamo z jednotlivých polí v hlavičke registračnej správy. Identifikátor a kontaktnú adresu registrovaného užívateľa obsahujú polia *From* a *Contact*. V našom prípade má užívateľ Bob pridelený indetifikátor vo formáte SIP URI *sips:bob@biloxi.example.com* a momentálne je dostupný s využitím adresy *sips:bob@client.biloxi.example.com*. Doménové meno registračného serveru je uvedené hneď na prvom riadku žiadosti za kľúčovým slovom REGISTER. Užívateľ Bob používa registračný server *sips:ss2.biloxi.example.com*. Z hľadiska prípadnej lokalizácie registračného serveru je vhodné poznať, aká je jeho aktuálna IP adresa. Tento údaj už získame analýzou hlavičky protokolu IP, keďže registračný paket je smerovný priamo na cieľovú IP adresu. Čas registrácie opäť získame analýzou hlavičky, ktorú pridá knižnica *libpcap* pri zachytení paketu.

Samotná analýza registračnej správy však nestačí. Získame z nej síce potrebné informácie, ale registráciu môžeme považovať za úspešnú až po prijatí potvrdzujúcej správy z registračnéhu serveru. Musíme si tak získané informácie vhodne uložiť, aby sme k nim mohli po obdržaní správy s kódom 200 opäť pristúpiť. Rovnako ako aj v prípade signalizácie vytvorenie hovoru, možeme použiť hodnotu uloženú v poli *Call-ID*, keďže registrácia užívateľa sa považuje za samostatnú transakciu.

Kapitola 5

Implementácia aplikácie

Cieľom tejto práce je navrhnúť a implementovať systém na detekciu hlasových hovorov na základe analýzy signalizačného protokolu SIP. Návrh aplikácie, ktorá bude sledovať parametre dôležité z hľadiska účtovania, je uvedený v predchádzajúcej kapitole. Proces implementácie navrhnutej aplikácie je popísaný v tejto kapitole.

5.1 Popis implementačného prostredia

Pred začiatkom samotného vývoja bolo potrebné rozhodnúť, akým spôsobom sa bude pristupovať k paketom na sieťovej vrstve. V tomto prípade bolo najvhodnejšie použiť knižnicu libpcap, prípadne niektorú jej portáciu v závislosti na zvolenom programovacom jazyku. V dnešnej dobe predstavuje knižnica libpcap štandard v oblastí zachytávania paketov a ich následnej analýzy.

Ako programovací jazyk sme zvolili Python, keďže pri online spracovávaní dát bude hlavný dôraz kladený na rýchlosť pri práci s veľkým objemom dát. Výhodou je tiež dostupnosť knižníc, ktorými môžeme vhodne rozšíriť požadovanú funkcionalitu.

Voľba operačného systému v podstate nie je veľmi dôležitá, keďže uvedené prostriedky existujú vo verziách pre všetky bežne používané systémy. Zvolili sme však systém Linux, konkrétne distribúciu Ubuntu, ktorá priamo obsahuje niektoré nástroje vhodné pre vývoj a následné ladenie sieťových programov.

Z dostupných knižníc sme použili nasledujúce:

- **pcapy**: Implementuje rozhranie knižnice *libpcap* pre jazyk Python. Umožňuje tak programom prístup k sieťovej karte a následnému zachytávaniu paketov. Výhodou je, že je kompatibilná s Unixovou knižnicou *libpcap* aj s knižnicou *WinPcap* pre systém Windows.

- **Impacket**: Knižnica Impacket predstavuje súbor viacerých tried zameraných hlavne na prácu so sieťovými paketmi. Umožnuje získavať hodnoty jednotlivých polí z hlavičiek protokolov, ako aj pracovať priamo s dátami. Obsahuje tried pre prácu s protokolmi IP, TCP a UDP dokáže pracovať aj s niektorými protokolmi vyšších vrstiev.

5.2 Zachytávanie paketov

Aplikácia môže po spustení pracovať v `online` alebo v `offline` režime. Oba režimy sa navájom od seba líšia iba v spôsobe zachytávania paketov. Knižnica *pcapy*, ktorá má na starosti

prácu so sieťovým rozhraním, vytvorí pri spustení programu globálny objekt, pomocou ktorého sa následne pristupuje k danému médiu. Vytvorenie tohto objektu (v dokumentácii sa označuje ako *packet capture descriptor*) je odlišné v závislosti na zvolenom režime práce programu.

Offline režim je vhodný pre analýzu vopred zachytených dát. Pri spustení je potrebné zadať názov vstupného súboru vo formáte pcap, ktorý obsahuje dáta pre analýzu. Na vytvorenie globálneho objektu pre prístup k dátam sa v tomto prípade použije metóda *open_offline* z knižnice *pcapy*, ktorá ako parameter použije zadané meno vstupného súboru. Volanie metódy je uvedené na následujúcom príklade.

```
pc = pcapy.open_offline(input_file)
```

V prípade, že chceme aplikáciu použiť ako paketový analyzátor v reálnom čase, musíme byť schopní pristúpiť priamo k sieťovému rozhraniu počítača, na ktorom bude aplikácia spustená. Pre takýto spôsob použitia je vhodný online režim. Pri spustení potrebujeme poznať názov rozhrania, na ktorom má aplikácia zachytávať dáta. Samozrejme je potrebné zaručiť, aby bol náš program spustený s dostatočnými systémovými právami, keďže program sám nedokáže svoje prístupové práva zmeniť. Volanie metódy, ktorá vytvorí globálny objekt pre prístup k sieťovému rozhraniu, je uvedené na nasledujúcom príklade.

```
pc = pcapy.open_live(interface, max_bytes, promiscuous, read_timeout)
```

Ako je z príkladu zrejmé, je potrebné zadať viacero parametrov:

- **interface**: špecifikuje názov rozhrania, na ktorom má aplikácia zachytávať dáta
- **max_bytes**: limit pre veľkosť paketu v bajtoch (v prípade rozhrania typu Ethernet je vhodné uviesť hodnotu 1500 bajtov)
- **promiscuous**: udáva, či sa rozhranie má prepnúť do promiskuitného módu, pri ktorom sieťová vrstva pošle na spracovanie vyšším vrstvám aj dáta, ktoré nie sú určené pre dané rozhranie
- **read_timeout**: udáva hodnotu v milisekundách, ktorú má program počkat pred poslaním zachytených dát na spracovanie

Okrem názvu rozhrania sú ostatné parametre predvolené s nasledujúcimi hodnotami: *max_bytes* = 1500 bajtov, *promiscuous* = True, *read_timeout* = 100 ms.

Pre samotným spracovaním paketov je vhodné špecifikovať globálny filter, ktorý sa použije na zachytené dáta. V určitých prípadoch tak dokážeme jednoducho eliminovať množstvo paketov, ktoré nás vzhľadom na zameranie aplikácie nezaujímajú. V našom prípade chceme najskôr analyzovať iba pakety protokolu SIP, ktorý používa port 5060 a transportný protokol UDP. Aplikáciou nasledujúceho filtra tak zaručíme, že žiadne iné pakety nebudú analyzované. Po ustanovení hovoru však musíme filter vhodným spôsobom upraviť, aby sme mohli zachytávať aj prípadné hlasové dáta.

```
pc.setfilter('udp port 5060')
```

5.3 Spracovanie zachytených paketov

Pred spracovaním paketov musíme najskôr rozhodnúť, akým spôsobom budeme ukladať získané informácie. Pri nasadení aplikácie v reálnom prostredí očakávame viacero súčasne vytváraných a prebiehajúcich hovorov. Môžeme teda predpokladať, že bude potrebné spracovávať množstvo signalizačných správ, ktoré budú prislúchať k rôznym hovorom. Informácie získané z analýzy týchto správ teda musíme ukladať tak, aby sme k nim mohli pristupovať rýchlo a na základe spoločného identifikátora hovoru.

Z dátových štruktúr, ktoré obsahuje jazyk Python sa nám pre tento účel najviac hodí štruktúra slovník (dictionary). Ide v podstate o asociatívne pole, pri ktorom môžeme k jednotlivým prvkom pristupovať na základe kľuča. Ako už bolo vysvetlené v predchádzajúcej kapitole, ako kľúč na priradenie jednotlivých správ k hovorom nám postačí hodnota z poľa *Call-ID* z hlavičky SIP správy. Tento istý kľúč teda môžeme použiť aj pre prístup k prvkom uloženým v slovníku. Pre každý sledovaný parameter si vytvoríme globálny slovník, do ktorého uložíme dáta získané počas analýzy príslušnej správy.

Samotné spracovanie zachytených paketov sa vykonáva v cykle, pomocou volania metódy *loop()* globálneho objektu `pc`.

```
pc.loop(packet_limit, recv_pkts)
```

Metóda pracuje s dvomi parametrami *packet_limit* a *recv_pkts*. Prvý udáva maximálny počet zachytených paketov. V prípade použitia filtra sa však jedná o počet paketov, ktoré už filtrom prešli. Ak nechceme počet paketov obmedzovať, môžeme použiť hodnotu -1. V takom prípade bude program v prípade online režimu pracovať až do prerušenia užívateľom, v prípade offline režimu program skončí po spracovaní všetkých dát zo vstupného súboru. Druhý parameter špecifikuje názov funkcie, ktorá má na starosti vlastné spracovanie paketov. V našom programe sme si definovali pre tento účel nasledujúcu funkciu:

```
def recv_pkts(hdr, data)
```

Keďže pracujeme s knižnicou *pcapy*, každý zachytený paket je uložený do štruktúry, ktorá pozostáva z hlavičky a dáta. Podrobný popis je uvedený v predchádzajúcej kapitole. Preto aj funkcia *recv_pkts()* pracuje s dvomi parametrami, ktoré dostane po zachytení paketu od metódy *loop()*. Prvý z nich je odkaz na pridanú hlavičku, druhý obsahuje zachytený paket.

Zachytený paket je predaný na spracovanie vo formáte rámca linkovej vrstvy. Aby sme mohli pracovať s hlavičkami sieťovej (protokol IP) a transportnej vrstvy (protokoly TCP a UDP), musíme vedieť efektívnym spôsobom zachytený rámec rozbaliť. Po rozbalení dostaneme paket sieťovej vrstvy, z ktorého môžeme získať informáciu o zdrojovej a cieľovej adrese protokolu IP. Aby sme sa dostali k hlavičke transportnej vrstvy, musíme celý proces znovu opakovať. Na rozbalenie paketu použijeme metódu *child()*, ktorú môžeme volať opakovane a získať tak prístup k hlavičke protokolu na požadovanej vrstve. Na nasledujúcom príklade je ukážka použitia spomenutej metódy.

```
ethernet_frame = EthDecoder().decode(data)
ip_packet = ethernet_frame.child()
datagram = ip_packet.child()
```

Najskôr získané dáta dekódujeme a potom volaním metódy *child()* vytvoríme ďalšie objekty, s ktorými môžeme následne pracovať.

Pakety spracovávané v našom programe môžeme rozdeliť na správy protokolu SIP a hlasové dáta. Signalizačné správy môžeme ešte ďalej rozdeliť na správy typu žiadosť a následné odpovede. Pre spracovanie každej z uvedených skupín sme definovali samostatnú funkciu. Funkcia *recv_pkts()* však musí najskôr každý prijatý paket zatriediť do niektorej kategórie a predať ho správnej funkcii na ďalšie spracovanie. Podľa cieľového čísla portu rozoznáme prijatú správu SIP, keďže protokol SIP používa port 5060. Žiadosti od odpovedí odlíšime pomocou prvého slova na štavovom riadku správy. Správa typu žiadosť obsahuje ako prvé kľúčové slovo, ktoré presne definuje, o aký typ žiadosti ide (napríklad INVITE, REGISTER atď.). Odpovede protokolu SIP majú na stavovom riadku najskôr špecifikáciu verzie protokolu (aktuálne používaná verzia je SIP/2.0), za ktorou nasleduje číselný kód odpovede. Postupne si popíšeme jednotlivé implementované funkcie, ktoré majú na starosti spracovanie uvedených typov paketov.

5.3.1 Žiadosti SIP

Na spracovanie správ typu žiadosť sme definovali nasledujúcu funkciu:

```
def process_sip_request(hdr, data, iphdr):
```

Funkcia pracuje s tromi parametrami `hdr`, `data` a `iphdr`. Parameter `hdr` obsahuje hlavičku pridanú pri zachytení paketu knižnicou pcapy. Použijeme ho na získanie informácie o čase zachytenia paketu. Parameter `iphdr` obsahuje hlavičku protokolu IP a získame z neho adresu registračného servera. Žiadosť o registráciu užívateľa je totiž adresovaná priamo a preto pole s cieľovou adresou IP bude priamo obsahovať požadovanú informáciu. Dáta aplikačnej vrstvy obsahuje parameter `data`. Ide v podstate o kompletnú správu protokolu SIP. V závislosti na type žiadosti sa mierne odlišuje aj spôsob spracovania. V našom programe pracujeme s nasledujúcimi typmi žiadostí:

INVITE Prvá správa pri vytváraní hovoru.

REGISTER Správa, ktorou užívateľ žiada o registráciu na registračnom serveri.

ACK Posledná správa pri vytvorení hovoru.

BYE Správa, ktorú pošle užívateľ ukončujúci hovor.

Ako prvý krok pri spracovaní každého typu správy najskôr získame hodnotu uloženú v poli Call-ID. Následne zisťujeme, o aký typ správy sa presne jedná. V prípade správy INVITE je spracovanie najkomplexnejšie. Z hlavičky správy SIP získame informácie o volajúcom a volanom účastníkovi. Zo správy SDP, ktorá sa nachádza v tele správy SIP získame informácie o navrhovaných parametroch RTP relácie. Popis jednotlivých polí, z ktorých dáta získavame je uvedený v predchádzajúcej kapitole, ktorá sa zaoberá návrhom aplikácie. Spracovanie správy REGISTER je v podstate rovnaké ako v prípade správy INVITE. Po získaní potrebných informácii ich uložíme do príslušných globálnych slovníkov pod kľúčom získaným z Call-ID. Je potrebné však sledovať stav každého hovoru, prípadne registrácie. Hodnoty v slovníkoch je možné ľubovolne meniť, preto nám stačí vytvoriť ďalší slovník, v ktorom budeme uchovávať stav hovorov a registrácii.

V prípade žiadostí typu ACK a BYE je však situácia odlišná. Správa ACK totiž potvrdzuje vytvorenie hovoru. Po obdržaní tejto správy tak musíme začat zachytávať hlasové dáta práve vytvoreného hovoru. Je teda nutné vhodne upraviť paketový filter, aby

okrem správ protokolu SIP akceptoval aj ďalšie pakety. Pre tento účel sme definovali funkciu `update_pckfilter()`. S využitím znalosti o zdrojovom a cieľovom porte RTP relácie upraví paketový filter do požadovanej formy. Formát filtrovacieho výrazu je totožný s filtrami použitými v nástroji tcpdump. Východzí filter má tvar `udp port 5060`. Každý zachytený RTP paket pred spracovaním ešte testujeme na správnost zdrojovej a cieľovej adresy IP, takže si môže dovoliť filtrovať iba na základe čísel portov. V prípade, že vytvorený hovor bude používať dvojicu portov 49158 a 49160, výsledný filter bude mať tvar `udp port 5060 or 49158 or 49160`.

Správa typu BYE hovor ukončuje. Po ukončení hovoru musíme najskôr vygenerovať do výstupného súboru štatistiky o uskutočnenom hovore. Následne musíme uložiť zachytené prúdy RTP a uvoľniť alokované systémové prostriedky. Pre tieto činnosti sme definovali nasledujúce funkcie:

```
def generate_output(key)
def write_rtp_stream(key)
def release_call(key)
```

Každá z funkcií pracuje iba s parametrom key, ktorý obsahuje hodnotu Call-ID práve ukončeného hovoru. Funkcie generate_output() a release_call() pomocou tohto kľúča priamo pristupujú k dátam uložným v slovníkoch. Funkcia write_rtp_stream() však použije hodnotu Call-ID iba na zostavenie ďalších kľúčov, pod ktorými sú uložené hlasové dáta. Nie je totiž možné priamo asociovať prúdy RTP so signalizáciou protokolu SIP. Kľúče pre prístup k hlasovým dátam vytvoríme kombináciou adries IP a čísel transportných portov účastníkov hovoru.

5.3.2 Odpovede SIP

Po každej správe typu žiadosť nasleduje jedna alebo viac správ typu odpoveď. Informujú žiadateľa o stave jeho žiadosti (preto sa niekedy tieto typy správ v literatúre označujú aj ako stavové správy - status messages). V prípade nášho programu nás zaujímajú nasledujúce typy odpovedí: 100, 180 a 200. Definovali sme nasledujúcu funkciu, ktorá dané pakety spracováva.

```
def process_sip_status(data)
```

Rovnako ako pri predchádzajucom type správ najskôr získame hodnotu Call-ID, aby sme signalizačnú správu mohli priradiť k správnemu hovoru. Spracovanie správ s kódom 100 a 180 spočíva v zmene stavu príslušného hovoru. Korektná sekvencia signalizačných správ je nasledujúca:

```
INVITE -> 100 Trying -> 180 Ringing -> 200 OK -> ACK
```

Takýmto spôsobom kontrolujeme správnu postupnosť signalizácie. Spracovanie správy s kódom 200 je však odlišné. Okrem potvrdenia ustanovenia hovoru totiž môže potvrdzovať aj ukončenie hovoru a úspešnú registráciu užívateľa na registračnom serveri. V prípade potvrdenia ustanovenia hovoru sa v tele správy nachádza opäť správa protokolu SDP. Ak však správa potvrdzuje ukončenie hovoru, okrem hlavičky protokolu SIP už neobsahuje žiadne dáta v tele správy. Správy môžeme navzájom rozlíšiť podľa celkovej dĺžky paketu, prípadne dotazom na prítomnosť reťazca špecifického pre správu protokolu SDP. V našom programe používame druhý z uvedených prístupov.

Poslednou možnosťou je, že správa potvrdzuje úspešnú registráciu užívateľa. Správy tohto druhu však odlíšime už pomocou hodnoty Call-ID, keďže stav registrácie sledujeme rovnakým spôsobom ako stav hovoru. Po potvrdení registrácie zapíšeme do výstupného súboru informácie získané počas analýzy správy REGISTER.

5.3.3 Spracovanie dát protokolu RTP

Spracovanie paketov s hlasovými dátami sa zásadne odlišuje od spracovania signalizácie. Pracujeme totiž s hlavičkami protokolov IP, UDP aj RTP. Definovali sme nasledujúcu funkciu:

```
def process_rtp_packet(packet)
```

V tomto prípade parameter `packet` obsahuje kompletný zachytený paket. S využitím metód z knižnice Impacket získame potrebné informácie, ktoré následne použijeme pre zostavenie unikátneho kľúča pre RTP dáta. Na nasledujúcom príklade sú znázornené jednotlivé kroky v poradí, v akom nasledujú po sebe.

```
ip_packet = packet.child()
ip_src = ip_packet.get_ip_src()
ip_dst = ip_packet.get_ip_dst()
udp_packet = ip_packet.child()
udp_src_port = udp_packet.get_uh_sport()
udp_dst_port = udp_packet.get_uh_dport()
rtp_key=ip_src+‘‘:’’+str(udp_src_port)+‘‘:’’+ip_dst+‘‘:’’+str(udp_dst_port)
```

Uvedeným spôsob zostavíme pre každý hovor dva rôzne kľúče, keďže aj prúdy RTP dát sú dva. Na základe takto získaného kľúča priradíme prijatý paket niektorému z existujúcich prúdov. Ďalším krokom v spracovaní je získanie hlasových dát bez pridaných hlavičiek uvedených protokolov. Postupnou aplikáciou metódy child() sa dostaneme až na úroveň dát protokolu UDP. Musíme teda ešte odstrániť hlavičku protokolu RTP. Minimálna veľkosť hlavičky je 12 bajtov. Celková veľkosť však závisí od počtu voliteľných polí, z ktorých každé pridá celkovej veľkosti 4 bajty. Druhý bajt v hlavičke RTP udáva počet tých voliteľných polí, preto po získaní tejto hodnoty môžeme vypočítať dĺžku dát, ktoré ešte musíme odstrániť.

```
hex_data = hexlify(udp_data.get_packet())
cc = int(hex_data[1],16)
rtp_hdr_size = 12 + 4*cc
```

Po odstránení hlavičky protokolu RTP získame už priamo hlasové dáta, ktoré priradíme na základe vypočítaného kľúča k danému prúdu RTP. Pakety spracovávame v takom poradí, v akom prichádzajú. S prípadnými výpadkami sa musí vysporiadať už aplikácia, ktorá hlasové dáta prijíma a spätne dekóduje do zvukovej podoby.

5.4 Práca s aplikáciou

V tejto časti si stručne popíšeme prácu s programom. Jedná sa o terminálovú aplikáciu, takže spustenie je priamo z príkazového riadka. Rovnako aj ovládanie je pomocou parametrov. Všetky možnosti spustenia sú uvedené na nasledujúcom príklade. Prvý parameter určuje režim práce programu. Ako druhý parameter je zadané meno rozrania, prípadne názov vstupného súboru.

```
./voip_sip_sniffer.py <online/offline> <interface_name/input_file>
```

Na nasledujúcom príklade sú postupne znázornené jednotlivé príklady spustenia. V prvom príklade chceme pracovať v *online* móde a chceme analyzovať dáta na rozhraní *eth0*. V druhom príklade pracujeme v *offline* režime, pričom ako vstupný súbor sme použili *sip-capture.pcap*. Posledný príklad ilustruje možnosť, pri ktorej si necháme vypísať nápovedu. Nápoveda sa vypíše aj v prípade nekorektne zadaného vstupu.

```
./voip_sip_sniffer.py online eth0

./voip_sip_sniffer.py offline sip-capture.pcap

./voip_sip_sniffer.py -h
```

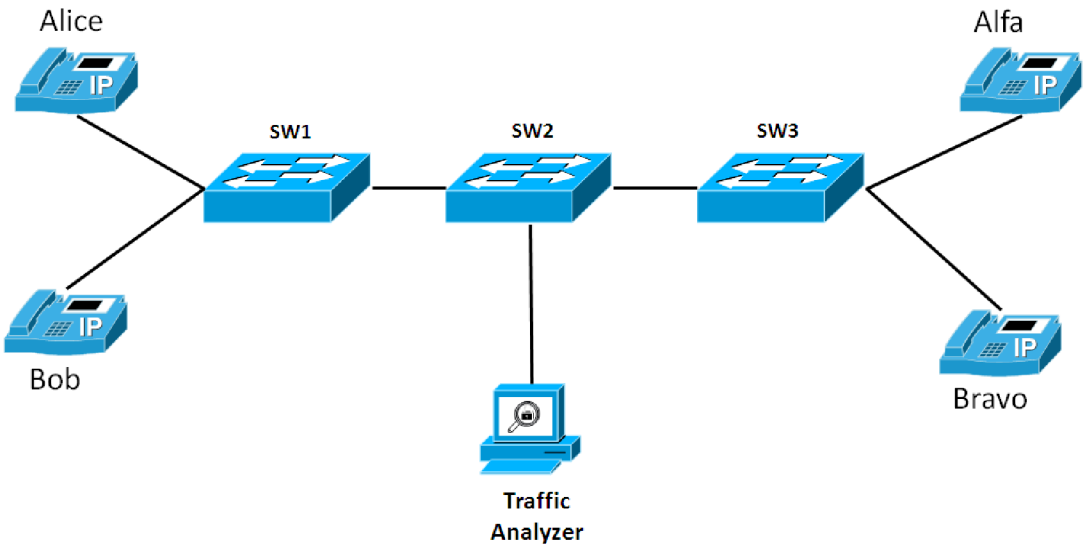
Kapitola 6

Testovanie aplikácie

Táto kapitola popisuje testovanie vytvorenej aplikácie. Postupne popíšeme jednotlivé druhy testov s rôznymi vstupnými dátami. Aby sme mohli reálne vyhodnotiť dosiahnuté výsledky, musíme výstup programu porovnať s referenčným nástrojom. Pre tento účel sme si zvolili nástroj Wireshark, keďže ponúka najviac funkcionality z hľadiska spracovania signalizácie a zachytených hlasových dát.

6.1 Popis testovacej topológie

Pre účely získania testovacích dát sme si v laboratórnom prostredí zapojili testovaciu topológiu, ktorá je znázornená na nasledujúcom obrázku. Laboratórne prostredie má tú výhodu, že môžeme ľubovolne manipulovať s nastaveniami siete a ovplyvniť tak podobu testovacích dát. V reálnom prostredí by sme si takéto zásahy dovoliť nemohli, keďže by hrozil priamy dopad na koncových užívateľov.



Obrázek 6.1: Testovacia topológia

Na obrázku sú znázornení štyria užívatelia, medzi ktorými vytvoríme dva súbežné hlasové hovory tak, aby dáta všetky dáta prechádzali prostredným prepínačom. Dáta zachytávame na hraničnom bode siete. V našom prípade sa jedná o prepínač SW2. Takéto zapojenie

je dôležité z hľadiska zachytenia signalizácie spolu hlasovými dátami. V nasledujúcej tabuľke sú uvedené adresy IP jednotlivých užívateľov spolu s adresami vo formáte SIP URI.

Užívateľ	Adresa IP	Adresa SIP URI
Alice	192.168.0.1	sip:Alice@192.168.0.1
Alfa	192.168.0.3	sip:Alfa@192.168.0.3
Bob	192.168.0.2	sip:Bob@192.168.0.2
Bravo	192.168.0.4	sip:Bravo@192.168.0.4

Tabulka 6.1: Užívateľské nastavenia

Pri každom teste boli zostavené dva hovory paralelne. Najskôr sme vytvorili hovor medzi užívateľmi Alice a Bravo. Po otestovaní správneho prenosu hlasu a vygenerovaní hlasových dát v oboch smeroch, sme vytvorili ďalší hovor. Druhý hovor bol medzi užívateľmi Alfa a Bob. Rovnako ako v prvom prípade sme otestovali prenos hlasu. Následne boli oba hovory ukončené v približne rovnakom čase. Po celú dobu testovania sme zahcytávali dáta, ktoré prechádzali prepínačom SW2. Tieto dáta boli následne analyzované referenčným nástrojom Wireshark a našim programom. Postupne si popíšeme jednotlivé testy a zhodnotíme výsledky.

6.2 Realizované testy

Aby sme overili funkčnosť nášho programu, realizovali sme 4 testy vždy s inými vstupnými dátami. Všetky testy prebiehali nad vopred zachytenými dátami a charakteristika jednotlivých testovacích vstupov je nasledovná:

- **kompletné dáta**: zachytené dáta obsahujú kompletnú signalizáciu aj hlasové pakety
- **dáta bez signalizácie**: zachytené sú len hlasové dáta, signalizácia chýba
- **dáta iba so signalizáciou**: zachytené sú iba pakety signalizačného protokolu SIP
- **neúplné dáta**: chýba časť signalizácie aj časť hlasových dát

Test s kompletnými dátami

V prípade tohto testu sme použili vstupný súbor, ktorý obsahoval všetky pakety signalizácie aj kompletné hlasové dáta. V podstate sa jedná o ideálnu situáciu, kedy pakety zachytávame na hraničnom bode v sieti. Všetky dáta tak prechádzajú jedným miesto. Môže sa napríklad jednať o situáciu, kedy chceme analyzovať hovory medzi dvomi pobočkami firmy a dáta zachytávame na prenajatej linke medzi týmito pobočkami. Na nasledujúcich príkladoch je znázornené spustenie nášho programu a terminálový výstup.

```
$./voip_sip_sniffer.py offline sip_peer-to-peer_g711.pcap
Call established
Call established
Call released
Call released
```

Po ukončení programu sa vygenerovali nasledujúde súbory:

```
output
call_1_caller_to_calling.raw
call_1_calling_to_caller.raw
call_2_caller_to_calling.raw
call_2_calling_to_caller.raw
```

Súbor `output` obsahuje výstupné štatistiky o uskutočnených hovoroch. Zostávajúce štyri súbory obsahujú už zachytené hlasové dáta z jednotlivých hovorov. Už podľa počtu môžeme určiť, že program rozoznal dva hovory a ku každému hovoru priradil dva prúdy hlasových dát. Podrobnejšie informácie o zachytených hovoroch zistíme priamo zo súboru `output`.

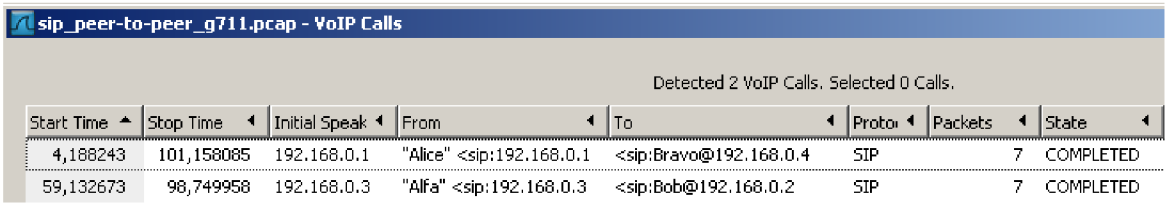
```
Call# 2: From: ,,Alfa‘‘ <sip:192.168.0.3>
Call# 2: To: <sip:Bob@192.168.0.2>
Call# 2: Media: audio 49158 RTP/AVP 8 3 97 98 0 101
Call# 2: Established: 16 May 2011 13:14:10
Call# 2: Released: 16 May 2011 13:14:43
Call# 2: Duration: 33sec
Call# 2: Caller IP: 192.168.0.3
Call# 2: Calling IP: 192.168.0.2
Call# 2: Call-ID: 6E0E5EA68D514E3EA2C28ADA247856360xc0a83801
Call# 2: voice stream from ,,Alfa‘‘ <sip:192.168.0.3> to
         <sip:Bob@192.168.0.2>: call_2_caller_to_calling.raw
Call# 2: voice stream from <sip:Bob@192.168.0.2> to
         ,,Alfa‘‘ <sip:192.168.0.3>: call_2_calling_to_caller.raw

Call# 1: From: ,,Alice‘‘ <sip:192.168.0.1>
Call# 1: To: <sip:Bravo@192.168.0.4>
Call# 1: Media: audio 49166 RTP/AVP 8 3 97 98 0 101
Call# 1: Established: 16 May 2011 13:13:16
Call# 1: Released: 16 May 2011 13:14:45
Call# 1: Duration: 89sec
Call# 1: Caller IP: 192.168.0.1
Call# 1: Calling IP: 192.168.0.4
Call# 1: Call-ID: F168F99ED7B244BAA76016E1AEDE14DA0xc0a83801
Call# 1: voice stream from ,,Alice‘‘ <sip:192.168.0.1> to
         <sip:Bravo@192.168.0.4>: call_1_caller_to_calling.raw
Call# 1: voice stream from <sip:Bravo@192.168.0.4> to
         ,,Alice‘‘ <sip:192.168.0.1>: call_1_calling_to_caller.raw
```

Zo získaných štatistík môže zistiť podrobné informácie o jednotlivých hovoroch. Pre dekódovanie hlasu môžeme použiť niektorý z uvedených nástrojov pre spracovanie zvuku (napríklad nástroj Audacity). Z uvedených parametrov multimediálnej relácie je zrejmé, že na prenos zvuku sa použil profil RTP s číslom 8. Znamená to, že bol použitý kodek G.711 A-Law. Parametre tohto kodeku, ktoré sú potrebné pre dekódovanie, sú uvedené v prílohe C.

Zostáva nám porovnať výstup nášho programu s nástrojom Wireshark. Pre analýzu IP telefónie je možné použiť veľké množstvo funkcií, ktoré nástroj Wireshark podporuje. Nám bude stačiť detekcia telefónnych hovorov a detekcia prúdov RTP v dátach. Štatistiku uskutočnencých hovorov získame po vybratí možnosti *VoIP Calls* zo systémovej ponuky

Telephony. Pre analýzu prúdov RTP musíme postupovať podobne. Z ponuky *Telephony* zvolíme možnosť *RTP* a následne *Show All Streams*. Výtupy uvedených štatistík sú znázornené na nasledujúcich obrázkoch.



Start Time	Stop Time	Initial Speak	From	To	Proto	Packets	State
4,188243	101,158085	192.168.0.1	"Alice" <sip:192.168.0.1	<sip:Bravo@192.168.0.4	SIP	7	COMPLETED
59,132673	98,749958	192.168.0.3	"Alfa" <sip:192.168.0.3	<sip:Bob@192.168.0.2	SIP	7	COMPLETED

Obrázek 6.2: Analýza VoIP hovorov

Detected 4 RTP streams. Choose one for forward and reverse direction for analysis

Src IP addr	Src port	Dst IP addr	Dst port	SSRC	Payload	Packets	Lost
192.168.0.1	49166	192.168.0.4	49160	0x27F90F21	g711A	4487	0 (0,0%)
192.168.0.2	49164	192.168.0.3	49158	0x1C118769	g711A	1621	0 (0,0%)
192.168.0.3	49158	192.168.0.2	49164	0xBD2EA6B	g711A	1620	0 (0,0%)
192.168.0.4	49160	192.168.0.1	49166	0xE7244EB	g711A	4485	0 (0,0%)

Obrázek 6.3: Analýza prúdov RTP

Je zrejmé, že aj nástroj Wireshark detekoval v danom vstupnom súbore dva hlasové hovory a k nim priradil štyri prúdy hlasových dát. V tomto prípade môžeme povedať, že naše výsledky sa zhodujú s výsledkami referenčného nástroja. Testovacie dáta, ktoré boli použité v tomto teste, je možné nájsť na priloženom CD v adresári `test-data\test1`.

Test s dátami bez signalizácie

Pre druhý test sme zvolili vstupný súbor, ktorý neobsahuje žiadnu signalizáciu protokolu SIP. Takáto sitácia môže nastať vtedy, ak dáta zachytávame iba na niektorých portoch prepínača. Signalizácia totiž nemusí prebiehať priamo medzi užívateľmi. Na rozdiel od signalizácie sú však hlasové prúdy vytvorené napriamo. Na nasledujúcom príklade je uvedený terminálový výstup po spustení aplikácie s daným vstupným súborom.

```
$./voip_sip_sniffer.py offline sip_peer-to-peer_rtp_only.pcap
$
```

Náš program nevypísal žiadnu informáciu o uskutočnencých hlasových hovorov. Rovnako neboli vygenerované žiadne výstupné súbory s hlasovými dátami, prípadne súbor so štatistikami. Jedná sa o korektnú reakciu, keďže detekcia hlasových dát je podmienená prítomnosťou signalizácie. Ako už bolo vysvetlené v predchádzajúcich kapitolách, rozoznať hlasové dáta protokolu RTP od normálnych dát protokolu UDP, je bez prítomnosti signaličného protokolu nemožné.

Na overenie ešte použijeme nástroj Wireshark. Ihneď po spustení vidíme, že vo vstupnom súbore sa nachádzajú iba dáta protokolu UDP. Na rozdiel od predchádzajúceho prípadu, kedy Wireshark sám rozoznal, že sa jedná o dáta protokolu RTP. Bez prítomnosti dodatočného zdroja informácie teda nástroj nerozozná prítomnosť hlasových dát. Rovnakým spôsobom ako v prípade prvého testu overíme, či boli detekované telefónne hovory, prípade prúdy RTP dát. V oboch prípadoch sú však výsledky negatívne.

Na priloženom CD v adresári `test-data\test2` sú uložené testovacie dáta použité v tomto teste. Záujemcovia si tak môžu uvedené testy zreprodukovať.

Test s dátami bez hlasových paketov

Ako tretí v poradí sme vykonali test, kedy máme zachytené iba signalizačné pakety. Otestovali sme tak prípadné nasadanie aplikácie v komplexnom prostredí s proxy serverom. V prípade zachytávania paketov priamo na serveri totiž môžeme očakávať, že zachytíme signalizáciu a minimum hlasových dát (prípadne žiadne). Po spustení programu sme dostali nasledujúci terminálový výstup:

```
$ ./voip_sip_sniffer.py offline sip_peer-to-peer_signalization_only.pcap
Call established
Call established
Call released
Call released
```

Program teda opäť správne zdetekoval oba hlasové hovory. Výsledné štatistiky vyzerajú rovnako ako v prípade prvého testu. Rovnako boli aj vytvorené výstupné súbory pre hlasové dáta. Pre overenie sme si však vypísali veľkosti vytvorených súborov.

```
$ ls -al

 0 2011-05-22 16:17 call_1_caller_to_calling.raw
 0 2011-05-22 16:17 call_1_calling_to_caller.raw
 0 2011-05-22 16:17 call_2_caller_to_calling.raw
 0 2011-05-22 16:17 call_2_calling_to_caller.raw
```

V uvedenom výstupe sú zobrazené iba informácie o súboroch s prípadnými hlasovými dátami. Je však vidieť, že veľkosť všetkých súborov je 0 bajtov. Znamená to, že program nenašiel žiadne relevantné dáta, ktoré by mohol priradiť k očakávaným hlasovým prúdom.

Pre overenie sme uvedený súbor analyzovali aj nástrojom Wireshark. Pri analýze hovorov boli opäť detekované dva zostavené aj ukončené hovory. Výstup programu je rovnaký ako v prípade prvého testu 6.2. V prípade analýzy zachytených RTP prúdov však neboli detekované žiadne. Testovacie dáta z tohoto testu sú rovnako k dispozícii na priloženom CD v adresári `test-data\test3`. Opäť môže tvrdiť, že náš program správne analyzoval vstupné dáta. Vytvorenie výstupných súborov pre hlasové dáta síce môže niektorých užívateľov pomýliť, ale po prípadnom dekódovaní budú vytvorené prázdne súbory. Tým bude jasné, že daný testovací súbor hlasové dáta neobsahoval.

Test s nekompletnými dátami

V tomto teste sme sa zamerali na možnosť, že sa nám nepodarí zachytiť kompletnú signalizáciu daného hovoru. Rovnako aj prípadné hlasové dáta nie sú zachytené v kompletnej podobe. Takáto situácia môže nastať v prípade, že analýza sieťových dát bude spustená v dobe, kedy niektoré hovory už prebiehajú. V takom prípade nám bude chýbať úvodná signalizácia pri vytváraní hovoru. Straty hlasových dát môžu nastať za rôznych podmienok. Najčastejšie však v prípade preťaženie niektorých sieťových spojení. V takom prípade dochádza k zahadzovaniu paketov aktívnymi sieťovými prvkami. Keďže na prenos hlasu sa

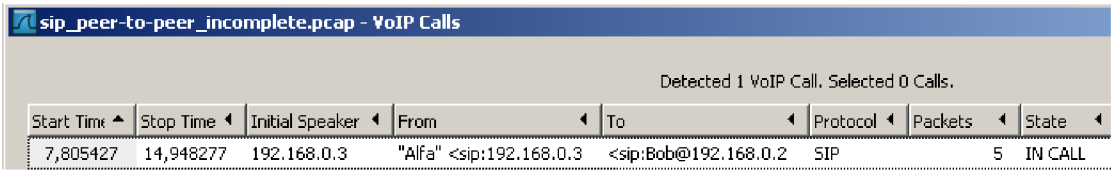
používa protokol UDP, ktorý neobsahuje žiadnu kontrolu doručenia dát, zahodené pakety nebudú opätovne preposlané.

Aby sme mohli uvedený prípad otestovať, upravili sme súbor s dátami z prvého testu. Odstránili sme signalizáciu vytvorenia prvého hovoru (medzi užívateľmi Alice a Bravo) a rovnako sme odstránili signalizáciu ukončenia druhého hovoru. Z hlasových dát bola odstránená podstatná časť paketov zachytených po zostavení druhého hovoru. Po spustení nášho programu s takto upravenými dátami sme dostali nasledujúci výstup:

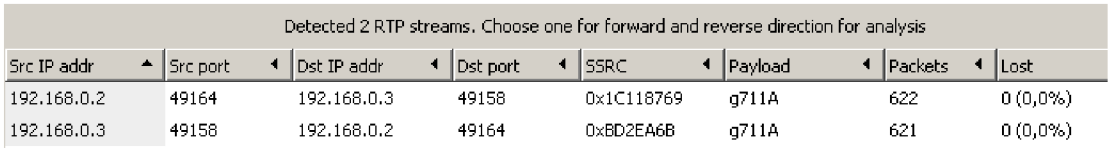
```
$ ./voip_sip_sniffer.py offline sip_peer-to-peer_incomplete.pcap
Call established
```

Program teda zdetekoval vytvorenie jedného hlasového hovoru. Je potrebné si však všimnúť, že nebolo detekované ukončeni hovoru. To má zásadný vplyv na celkový výstup programu. Pogram bol navrhnutý tak, aby prípadné štatistiky uložil do výstupného súboru až po detekovaní ukončenia hovoru, v tomto prípade teda žiadne štatistiky o vytvorenom hovore nedostaneme. Rovnako môžeme predpokladať, že boli zachytené určité časti hlasových dát. Neboli však uložené, lebo rovnako ako v prípade generovania štatistík sa toto deje až pri ukončení hovoru. Priebežné ukladanie dát by totiž bolo neefektívne.

Opäť sme ten istý súbor analyzovali pomocou nástroja Wireshark. Výsledky analýzy uskutočnených hovorov spolu s testom na prítomnosť hlasových prúdov sú na nasledujúcich obrázkoch.



Obrázek 6.4: Analýza VoIP hovorov



Obrázek 6.5: Analýza prúdov RTP

Z uvedených výsledkov je vidieť, že nástroj Wireshark zdetekoval jeden hlasový hovor a k nemu priradil dva prúdy multimediálnych dát s použitím protokolu RTP. Za povšimnutie stojí, že detekovaný hovor je v stave IN CALL. To znamená, že bola síce zaznamenaná signalizácia pri vytvorení hovoru, ale signalizácia ohľadom ukončenia sa v zachytených dátach nevyskytuje. V podstate sa dá povedať, že hovor by zostal v stave IN CALL stále, bez ohľadu na dobu behu programu. V prípade hlasových dát môžeme zistiť rozdiely už vo výslednej veľkosti súborov. Nástroj Wireshark totiž tiež umožňuje extrakciu dát z RTP prúdov. Rovnako zistíme rozdiely aj po prekódovaní dát do zvukovej podoby. Zvuková stopa v tomto teste je podstatne kratšia ako v prípade prvého testu. Zvuk je však možné prehrať bez problémov.

V tomto prípade sa náš program choval odlišne od referenčného nástroja. Bol schopný iba zdetekovať vytvorenie hovoru. Poskytnutie výstpnej štatistiky ako aj prípadných hlasových dát (aj keď neúplných) nebolo možné. Nástroj Wireshark bol schopný analyzovať

vstupné dáta a poskytnúť aspon čiastočné údaje o zostavenom hovore. Rozdielne správanie je spôsobené návrhom nášho programu, ktorý pracuje s hlasovými hovormi ako s kompletným súborom signalizačných správ. V prípade chýbajúcich dát ohľadom vytvorenia alebo ukončenia hovoru, nie je schopný daný hovor správne detekovať a poskytnúť aspoň čiastočné štatistiky. Rovnako aj v prípade tohoto testu sú upravené testovacie dáta poskytnuté na priloženom CD v adresári `test-data\test4`.

6.3 Zhodnotenie testov

Na uvedených testoch sme otestovali reálne nasadenie našej aplikácie. Účel testovanie bol získať čo najviac informácii o uskutočnených hlasových hovoroch zostavených pomocou signalizačného protokolu SIP. Ukázali sme si, ako program spracuje dáta, kde sú zachytené všetky signalizačné správy spolu s hlasovými dátami. Okrem toho sme sa zamerali aj na situácie, kedy nemáme všetky potrebné dáta k dispozícii.

Po porovnaní s referenčným nástrojom Wireshark môžeme povedať, že náš program v podstate splňuje požiadavky, ktoré sme popísali v návrhu aplikácie. Aby sme boli schopní rozoznať aj tie hovory, ktoré boli síce korektne zostavené počas behu programu, ale neboli ukončené, museli by sme významným spôsobom zasiahnuť do architektúry programu. Z hľadiska bežného použitia však táto situácie nie je veľmi pravdepodobná. Predpokladáme totiž dlhodobú analýzu a v prípade, že správne zachytíme zostavenie hovoru, môžeme rovnako očakávať aj zachytenie ukončenia hovoru.

Kapitola 7

Záver

Cieľom tejto diplomovej práce bolo zoznámiť sa s architektúrou IP telefónie, naštudovať signalizačný protokol SIP a transportný protokol RTP. Následne bolo potrebné navrhnúť aplikáciu, ktorá bude mať za úlohu detekovať VoIP hovory na hraničnom bode siete.

V teoretickej časti sú popísané protokoly SIP, RTP a RTCP spolu s názornými príkladmi signalizácie pri vytvárani hlasového hovoru. Nasleduje popis nástrojov používaných k analýze sieťovej prevádzky. Špeciálne sme sa zamerali hlavne na také nástroje, ktoré sú buď priamo špecializované na analýzu hlasových dát, alebo obsahujú funkcionalitu pre prácu so signalizačnými protokolmi a multimediálnymi dátami. Okrem paketových analyzátorov sme popísali aj vybrané nástroje určené na spracovanie hlasových dát. Ide hlavne o nástroje, ktoré umožňujú hlasové dáta dekódovať do zvukovej podoby.

Najväčšiu pozornosť sme venovali návrhu systému, kde popisujeme, ktoré informácie sú dôležité z hľadiska účtovania prevádzky. Uvádzame aj príklady jednotlivých signalizačných správ, na ktorých vysvetlíme ako môžeme požadované informácie získať. V návrhu sa venujeme aj hlasovým dátam. Je popísaný spôsob, ako zo zachytených dát získať iba hlasové dáta bez pridaných hlavičiek sieťových protokolv. Rovnako sme sa zaoberali aj postupom, ako čo najefektívnejšie priradiť hlasové dáta k zostaveným hovorom.

Navrhnutý program sme následne implementovali v jazyku Python s využitím knižníc pcapy a Impacket. V kapitole, ktorá sa zaoberá implementáciou sme popísali definované funkcie pre spracovanie signalizačných správ protokolu SIP ako aj funkcie definované za účelom spracovanie hlasových dát.

Veľmi podstatná je časť, ktorá sa zaoberá testovaním implementovanej aplikácie. Na sade rôznych testov sme ukázali, že nami implementované rišenie v podstate spĺňuje zadanie. Aplikácia poskytuje dostatočné štatistiky uskutočnených hovorov. Okrem toho dokáže získať zo zachyteného hlasového prúdu čisto hlasové dáta, ktoré sú vhodné na následné spracovanie. Výstup nášho programu sme porovnali s referenčným nástrojom Wireshark, ktorý vo väčšine prípadov dosiahol rovnaké výsledky.

Z hľadiska ďalšieho vývoja aplikácie by bolo vhodné zamerať sa na vylepšenie práce s hovormi, ktoré boli síce korektne zostavené, ale neboli správne ukončené. Rovnako pripadá do úvahy rozšíriť funkcionalitu programu o rodinu signalizačných protokolov H.323.

Literatura

[1] A. Johnston and S. Donovan and R. Sparksa and C. Cunningham and K. Summers: Session Initiation Protocol (SIP) Basic Call Flow Examples. RFC 3665, December 2003.

[2] Alan B. Johnson: *SIP: Understanding the Session Initiation Protocol*. Artech House Publishers, 2003, iSBN 1-58053-655-7.

[3] H. Schulzrinne and S. Casner: RTP Profile for Audio and Video Conferences with Minimal Control. RFC 3551, July 2003.

[4] H. Schulzrinne and S. Casner and R. Frederick and V. Jacobson: RTP: A Transport Protocol for Real-Time Applications. RFC 3550, July 2003.

[5] J. Rosenberg and H. Schulzrinne: An Offer/Answer Model with the Session Description Protocol (SDP). RFC 3264, June 2002.

[6] J. Rosenberg and H. Schulzrinne and G. Camarillo and A. Johnston and J. Peterson and R. Sparks and M. Handley and E. Schooler: SIP: Session Initiation Protocol. RFC 3261, June 2002.

[7] Jacobson, V.; Leres, C.; McCanne, S.: Man page of TCPDUMP. 2009, [Online; cit. 23-Máj-2011].
URL http://www.tcpdump.org/tcpdump_man.html

[8] Lamping, U.; Sharpe, R.; Warnicke, E.: Wireshark User's Guide. 2010, [Online; cit. 23-Máj-2011].
URL http://www.wireshark.org/docs/wsug_html_chunked/

[9] M. Handley and V. Jacobson: SDP: Session Description Protocol. RFC 2327, April 1998.

[10] Travis Russell: *Session Initiation Protocol (SIP): Controlling Convergent Networks*. McGraw-Hill Osborne Media, 2008, iSBN 0-07-164367-2.

Dodatek A

Obsah CD

- **/src/**: aplikácia implementovaná v jazyku Python
- **/test-data/**: testovacie vstupné súbory
- **/results/**: výsledky testovania
- **/documentation/**: zdrojové súbory textovej práce

Dodatek B

Správy SIP

Pre lepšiu ilustráciu ustanovenia (a následného ukončenia) hovoru s použitím protokolu SIP uvádzame kompletný výpis potrebných signalizačných správ. Uvedený príklad je prevzatý z RFC 3665 [1] a jedná sa o jednoduchšiu verziu, kedy sa hovor vytvorí priamo medzi účastníkmi. V citovanom dokumente RFC je však možné nájsť aj komplexné príklady, s využitím viacerých registračných serverov SIP, prípadne príklady signalizácie pri presmerovaní hovoru atď. Tieto príklady však svojim rozsahom prekračujú možnosti tejto práce.

Užívateľ, ktorý hovor vytvára, pošle najskôr správu INVITE spolu s navrhovanými parametrami budúcej multimediálnej relácie.

```
INVITE sip:bob@biloxi.example.com SIP/2.0
Via: SIP/2.0/TCP client.atlanta.example.com:5060;branch=z9hG4bK74bf9
Max-Forwards: 70
From: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
To: Bob <sip:bob@biloxi.example.com>
Call-ID: 3848276298220188511@atlanta.example.com
CSeq: 1 INVITE
Contact: <sip:alice@client.atlanta.example.com;transport=tcp>
Content-Type: application/sdp
Content-Length: 151

v=0
o=alice 2890844526 2890844526 IN IP4 client.atlanta.example.com
s=-
c=IN IP4 192.0.2.101
t=0 0
m=audio 49172 RTP/AVP 0
a=rtpmap:0 PCMU/8000
```

Volaný účastník následne odpovedá najskôr správou s kódom 180. Táto správa znamená, že na strane volaného účastníka bol telefónu doručený tón zvonenia. Po akceptovaní hovoru pošle užívateľ správu s kódom 200 a pripojí aj finálne parametre relácie.

```
SIP/2.0 180 Ringing
Via: SIP/2.0/TCP client.atlanta.example.com:5060;branch=z9hG4bK74bf9
;received=192.0.2.101
From: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
```

```
To: Bob <sip:bob@biloxi.example.com>;tag=8321234356
Call-ID: 3848276298220188511@atlanta.example.com
CSeq: 1 INVITE
Contact: <sip:bob@client.biloxi.example.com;transport=tcp>
Content-Length: 0

SIP/2.0 200 OK
Via: SIP/2.0/TCP client.atlanta.example.com:5060;branch=z9hG4bK74bf9
;received=192.0.2.101
From: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
To: Bob <sip:bob@biloxi.example.com>;tag=8321234356
Call-ID: 3848276298220188511@atlanta.example.com
CSeq: 1 INVITE
Contact: <sip:bob@client.biloxi.example.com;transport=tcp>
Content-Type: application/sdp
Content-Length: 147

v=0
o=bob 2890844527 2890844527 IN IP4 client.biloxi.example.com
s=-
c=IN IP4 192.0.2.201
t=0 0
m=audio 3456 RTP/AVP 0
a=rtpmap:0 PCMU/8000
```

Ako poslednú v rámci vytvorenia hovoru pošle volajúci užívateľ správu ACK, čím sa hovor považuje za ustanovený. Následne sa nadviažu prúdy RTP a začne sa vlastný prenos hlasových dát.

```
ACK sip:bob@client.biloxi.example.com SIP/2.0
Via: SIP/2.0/TCP client.atlanta.example.com:5060;branch=z9hG4bK74bd5
Max-Forwards: 70
From: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
To: Bob <sip:bob@biloxi.example.com>;tag=8321234356
Call-ID: 3848276298220188511@atlanta.example.com
CSeq: 1 ACK
Content-Length: 0
```

Ukončiť hovor môže každý účastník poslaním správy BYE. Tá sa potvrdí druhou stranou zaslaním správy s kódom 200, ktorá už na rozdiel od správy pri vytváraní hovoru neobsahuje popis relácie.

```
BYE sip:alice@client.atlanta.example.com SIP/2.0
Via: SIP/2.0/TCP client.biloxi.example.com:5060;branch=z9hG4bKnashds7
Max-Forwards: 70
From: Bob <sip:bob@biloxi.example.com>;tag=8321234356
To: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
Call-ID: 3848276298220188511@atlanta.example.com
CSeq: 1 BYE
```

```
Content-Length: 0

SIP/2.0 200 OK
Via: SIP/2.0/TCP client.biloxi.example.com:5060;branch=z9hG4bKnashds7
;received=192.0.2.201
From: Bob <sip:bob@biloxi.example.com>;tag=8321234356
To: Alice <sip:alice@atlanta.example.com>;tag=9fxced76sl
Call-ID: 3848276298220188511@atlanta.example.com
CSeq: 1 BYE
Content-Length: 0
```

Dodatek C

Profily RTP

V nasledujúcej tabuľke je uvedený zoznam profilov RTP a príslušných kodekov, ktoré sa používajú pri prenose hlasových dát. Kompletný zoznam, vrátane profilov používaných pre prenos videa, je možné nájsť v RFC 3551 [3]. Uvedená je aj veľkosť dátového toku, ktorý jednotlivé kodeky generujú. Stručný popis charakterizuje iba najznámejšie typy.

RTP payload	**Názov kodeku**	**Rýchlosť vzork. (Hz)**	**Počet kanálov**	**Popis**
0	G.711 PCMU	8000	1	ITU-T štandard, špecifikácie u-Law pre Severnú Ameriku a Japonsko, generuje dátový tok 64kbit/s
3	GSM	8000	1	pôvodne vyvinutý pre mobilné siete, generuje dátový tok 13kbit/s
4	G.723	8000	1	používa sa pri obmedzenej šírke pásma, produkuje dátový tok maximálne 6,3kbit/s
7	LPC	8000	1	dátový tok 5,6 kbit/s
8	G.711 PCMA	8000	1	ITU-T štandard, špecifikácia A-Law pre Európu, ostatné vlastnosti rovnaké ako pri u-Law
9	G.722	8000	1	ITU-T štandard, môže generovať dátový 48, 56 alebo 64kbit/s
10	L16	44100	2	nekomprimovaný 16-bit stereo zvuk, potrebná šírka pásma 1411,2 kbit/s
11	L16	44100	1	podobne ako predchádzajúci typ, prenáša sa iba jeden kanál, šírka pásma 705,6 kbit/s
12	QCELP	8000	1	šírka pásma 8 kbit/s
14	MPA	90000	1	prenos zvuku pri využití MPEG-1 a MPEG-2 bez kódovania obrazu
15	G.728	8000	1	ITU-T štandard, požadovaná šírka pásma 16 kbit/s
18	G.729	8000	1	ITU-T štandard, generuje dátový 8 kbit/s