FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Detekce a záznam konferenčních VOIP hovorů
Student:	Bc. Matěj Polák
Vedoucí:	Mgr. Jan Starý, Ph.D.
Studijní program:	Informatika
Studijní obor:	Počítačové systémy a sítě
Katedra:	Katedra počítačových systémů
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

Zobecněte existující aplikaci VoipCall (BP Kučera, Polák, Robejšek, Šuster) na konferenční hovory. Navrhněte znovu základní strukturu "hovoru": konferenční hovor nemá zdroj a cíl, začátek a konec, ale sestává z libovolně konečně mnoha "streamů", které vedou mezi navzájem různými místy, začínají a končí v různý čas, a během hovoru případně mění kodek.

1. Pozměňte odpovídajícím způsobem základní datové struktury.
2. Rozšiřte stávající databázové schéma.
3. Popište a implementujte signalizaci konferenčních hovorů alespoň v jednom ze signalizačních protokolů (SIP, Skinny).
4. Vytvořenou funkcionalitu řádně otestujte s využitím VOIP infrastruktury FIT ČVUT, případně i vlastní.

Seznam odborné literatury

Dodá vedoucí práce.

prof. Ing. Róbert Lórencz, CSc.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 13. října 2017

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Detekce a záznam konferenčních VOIP hovorů

Bc. Matěj Polák

Katedra počítačových systémů
Vedoucí práce: Mgr. Jan Starý, Ph.D.

12. února 2019

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 12. února 2019 .....

Odkaz na tuto práci

Polák, Matěj. *Detekce a záznam konferenčních VOIP hovorů*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

Abstrakt

Práce se zabývá implementací programu v jazyku C pro detekcí a záznam hovorů signalizovaných proprietárním signalizačním protokolem Skinny Client Control Protocol firmy Cisco. Jedná se hlavně o rozšíření předchozí práce o podporu konferenčních hovorů a tím spojený návrh nové architektury s možností rozšíření i pro další protokoly.

Klíčová slova VoIP telefonie, telefonní hovory, signalizace VoIP hovorů, Skinny Client Control Protocol, Skinny, SCCP, konferenční hovory

Abstract

The purpose of this work is to implement an application in the C language for the detection and recording VoIP calls signalized by Skinny Client Control Protocol by the Cisco company. The main task is to extend and generalize previous work to cover conference calls and to design new architecture with ability to be easily extended by other VoIP protocols.

Keywords VoIP telephony, phone calls, VoIP call signaling, Skinny Client Control Protocol, Skinny, SCCP, konferenční hovory

Obsah

Seznam obrázků

Seznam tabulek

Úvod

V současné době patří k běžným prostředkům komunikace i telefonní hovory. Kromě běžných telefonů pracujících v síti GSM se používají i telefony VoIP. Tento termín označuje souhrně protokoly schopné vytvořit telefonní hovor uvnitř počítačové sítě.

Za jednu z nejznámějších aplikací patřících do této kategorie lze považovat například program Skype. Ten ale využívá vlastní, neveřejný protokol. Mezi otevřené protokoly patří SIP, H.323 nebo právě proprietární protokol Skinny společnosti Cisco, kterým se zabývá tento text.

Tato diplomová práce rozšiřuje a zobecňuje moji bakalářskou práci [1]. Ta navazovala na aplikaci vzniklou ve spolupráci třech bakalářských prací, jejichž cílem bylo vytvořit funkční prototyp aplikace na monitorování a nahrávání VoIP hovorů v protokolu SIP. Původní tři bakalářské práce byly tyto: práce F. Šustera [2] obsahující implementaci hlavní části aplikace s implementací detekce hovorů, možnost konfigurace aplikace, rozdělení na jednotlivé části, návrh databáze, logování atd... Cílem práce V. Robejška [3] bylo vytvořit webové grafické rozhraní. Třetí prací byla práce J. Kučery [4] zabývající se extrakcí zvukových dat ze zachycených a uložených packetů. Moje bakalářská práce navazovala na tyto tři, hlavně na první z nich, a zobecňovala implementaci tak, aby celá aplikace byla schopna kromě hovorů v protokolu SIP detekovat a nahrávat i hovory v protokolu Skinny. Celý název tohoto protokolu je Skinny Cisco Control Protocol, v textu je použita i zkratka SCCP.

Jedním z možných rozšíření původní implementace bylo umožnění detekovat kromě jednoduchých hovorů i konferenční hovory a nahrávat jednotlivé účastníky. Pro takovou změnu je ale potřeba úplně jiný návrh architektury a modelu databáze. A právě touto změnou, novým návrhem a problémům s tím spojeným se zabývá tato diplomová práce. Kromě zobecnění a nového návrhu je součástí této práce implementace pro protokol Skinny. Konferenční hovory lze provádět i v dalších protokolech (Skype, SIP nebo H.323), těmi se ale v této práci nezabývám. Nová architektura je navržena tak, aby přidání

podpory dalšího protokolu bylo co nejjednodušší.

Vzhledem k tomu, že tato práce navazuje na moji bakalářskou práci a tématicky se s ní překrývá, jsou z ní některé části textu převzaty.

Kapitola 1

Cíl práce

Cílem práce je upravit stávající projekt pro odchytávání telefonních hovorů na protokolu SIP a Skinny o podporu konferenčních hovorů. Technologie bude použita stejná jako pro původní projekt, tedy jazyk C a databáze PostgresSQL.

Vzhledem k tomu, že konferenční hovory mají jinou strukturu, než standardní hovory mezi pouze dvěma účastníky, je potřeba navrhnout novou architekturu celé aplikace, a to včetně změny databázového návrhu. Původní technologie je zachována hlavně kvůli možnému znovupoužití hotových komponent a funkcí. Hlavní rozdíl oproti původní implementaci je pohled na to, jak je definován pojem hovor. Konferenční hovor se skládá z více účastníků, kteří se můžou ke konferenci připojovat a odpojovat. Cílem práce je navrhnout systém, který bude schopen konferenční hovory detekovat a nahrávat.

Součástí práce je také otestování implementované funkcionality.

Kapitola 2

VoIP obecně

Tato kapitola je s úpravami převzata z mé bakalářské práce [1].

VoIP (Voice over Internet Protocol) je obecné označení pro jakoukoliv technologii umožňující realizaci telefonních hovorů přes počítačovou síť pomocí protokolu IP. Počítačovou sítí se myslí jak místní síť (LAN) tak Internet. VoIP telefonní hovory nemusí být pouze hlasové (zvukové), mohou obsahovat i video. Tato práce, stejně jako předchozí práce, na kterých je založena, se věnuje pouze hovorům obsahujícím zvuk.

Síť podporující protokol IP přepojuje packety (*packet switching*), nikoliv okruhy (*circuit switching*), kdy je pro každý hovor vyhrazena cesta sítí mezi volajícím a volaným po celou dobu hovoru. Zároveň musí být síť pro VoIP telefonii dostatečně rychlá, tj. schopná přenést v daném čase určité minimální množství dat nutné pro plynulý přenos zvuku. To v dnešní době rychlého připojení přestává být problém, minimální nutnou rychlost pro VoIP hovory splňuje i relativně nízkorychlostní připojení.

VoIP se používá většinou ve stejné síti, ve které se používají i jiné síťové technologie. Díky tomu jsou v této síti přítomny i datové packety nijak nesouvisející s telefonními hovory. U většiny datových přenosů záleží na kvalitě přenosu i na úkor rychlosti, u VoIP je tomu přesně naopak. Pokud například trvá načtení webové stránky nebo stažení nějakého souboru o sekundu nebo několik sekund déle než obvykle, uživateli to většinou nevadí a počká. Případné chyby přenosu jsou tedy raději opravovány a chybné packety posílány znovu. Příkladem takovéto přenosu je protokol TCP. Pro VoIP je tomu ale přesně naopak — pokud by se nějaký packet, který byl součástí zvukových dat, zpozdil o sekundu a čekalo by se na jeho odeslání znovu, pro uživatele by VoIP bylo nepoužitelné. Proto se pro přenos těchto packetů používá nepotvrzovaný protokol UDP, u kterého nejsou jednotlivé packety kontrolovány a v případě chyby jsou raději zahozeny, protože je mnohem důležitější, aby byla tyto data doručena včas i za cenu možných chyb přenosu, než doručena se zpožděním.

S tím také souvisí QoS (Quality of Service) — pomocí QoS je možné vyhradit pro VoIP packety nějakou šířku datového toku sítí. Tím lze docílit

toho, že i kdyby byla síť zahlcena jinými datovými přenosy, VoIP hovory by tím nebyly ovlivněny a bylo by možné tuto síť stále používat pro telefonování.

Pro VoIP se obecně používají dva různé protokoly. První je signalizační, kterým si zařízení mezi sebou vyměňují informace o tom, kdo komu volá, jestli druhý telefon vyzvání nebo jestli druhý telefon hovor přijal nebo odmítl. Obecně se pro jakékoli události, které nějak souvisí s VoIP a nejedná se přímo o zvuková data, používá právě nějaký ze signalizačních protokolů.

Druhý protokol je transportní. Tento protokol se, na rozdíl od signalizačních, používá pouze jeden. Jedná se o univerzální protokol RTP (Real time Transport Protocol), ve kterém jsou obsažená zvuková data zkomprimována nějakým kodekem.

Kromě signalizačního a transportního protokolu využívá VoIP obvyklé služby DNS, DHCP, TFTP, DNS, a NTP, které nejsou pro VoIP specifické.

Typické pro VoIP je využití technologie PoE (Power over Ethernet). Jedná se o napájení VoIP telefonů přímo pomocí ethernetových kabelů, tedy stejných kabelů, kterými se tato zařízení připojují do počítačové sítě. To podstatně usnadňuje zapojování nových telefonů — stačí připojit pouze jeden kabel [5].

Kapitola 3

Použité nástroje

Tato kapitola popisuje nástroje použité při práci na implementaci funckionality popisované v tomto textu. Nástroj Wireshark a knihovna libpcap využívající Berkeley Packet filter byly použity i v mé bakalářské práci, následující dvě podkapitoly jsou proto z ní z větší části převzaty.

3.1 Wireshark

Wireshark je program sloužící k analýze zachycených síťových dat, která dokáže přehledně zobrazit. Obsahuje tzv. *dissectory* pro jednotlivé protokoly. Tyto dissectory slouží k „rozebrání“ binárních dat jednotlivých packetů a zobrazení jejich hodnot v přehledné stromové struktuře. Kromě toho umožňují vyhledání hodnoty různých položek v zachycených packetech. Jeden z těchto dissectorů je implementován i pro protokol Skinny a umožňuje detailní zobrazení jednotlivých položek ve Skinny zprávách. Zdrojový kód tohoto dissectoru je k dispozici například na [6]. Jedná se o jeden z hlavních zdrojů informací o struktuře protokolu Skinny, který jsem byl schopen najít.

Další funkcí tohoto programu je schopnost extrahovat RTP packety ze zachyceného síťového provozu, zrekonstruovat z nich zvukový záznam a ten následně uložit do soboru k pozdějšímu přehrání. Tato funkce je velice užitečná pro testování správnosti aplikace.

3.2 Libpcap a Berkeley Packet Filter

Libpcap je standardní knihovna pro práci s packety zachycené ze síťového provozu. Na libpcap jsou postaveny i další nástroje použité v této práci, jako třeba tcpdump nebo Wireshark. Tato knihovna zachycuje jednotlivé rámce (*frames*), které je potřeba rozebrat po jednotlivých vrstvách ISO/OSI modelu a extrahovat z nich pro aplikaci důležité údaje. Zároveň knihovna umožňuje packety filtrovat podle síťového protokolu, čísel portů apod. K tomu používá

nástroj zvaný Berkeley Packet Filter, který umožňuje filtrování packetů definováním filtru v uživatelském prostoru (*user space*) operačního systému a jejich zpracování přímo v prostoru jádra (*kernel space*). Díky tomu je možné se při zpracování packetů obejít bez jejich kopírování z prostoru jádra do uživatelského prostoru a tím se vyhnout zbytečné režii.

3.3 Databáze PostgreSQL

PostgreSQL je Open Source databázový systém použitý už při původní verzi. Pro novou verzi aplikace s podporou konferenčních hovorů jsem zvolil znovu tento databázový systém hlavně z důvodu znovupoužitelnosti implementace databázové vrstvy. Pro aktuální verzi projektu popisovaného v této práci je tento databázový systém více než dostačující. Aktuální i předchozí implementace projektu využívá pouze základní funkce databáze. Pro tyto účely by tedy byla dostačující jakákoli jiná relační databáze. Díky této volbě je ale mnohem snažší rozšiřovat celou aplikaci.

3.4 Knihovna libsndfile

Tato knihovna je použita pro extrakci zvukových dat ze zachycených packetů a uložení těchto zvukových dat do souboru. Jedná se o jednoduchou knihovnu napsanou v jazyce C, která je schopná pracovat s různými formáty zvukových dat.

Kapitola 4

Původní situace

Jak už bylo zmíněno v úvodu, tato práce se zabývá rozšířením a nutným přepracováním původního projektu určeného pro odchytávání a nahrávání VoIP hovorů s podporou protokolů SIP a Skinny. Původní projekt vznikl nejprve jako tři bakalářské práce ([2], [4] a [3]), poté jsem pokračoval já svojí bakalářskou prací s účelem rozšíření implementace i o protokol SCCP a s tím spojeným zobecněním datových struktur a databázového modelu [1]. Tato práce navazuje na mojí bakalářskou práci a dále ji zobecňuje a popisuje nutné úpravy architektury pro potřeby podpory konferenčních hovorů. Stejně jako původní projekt, i tento je napsán v jazyce C a využívá verzovací system Git (více informací je k dispozici například v [7]), díky kterému je možné zpětně dohledat každou úpravu zdrojového kódu včetně autora a času úpravy.

4.1 Struktura TCALL

Původní návrh spoléhal na to, že jeden hovor je veden jenom mezi dvěma účastníky, tedy obsahuje jeden začátek, jeden konec a dva zvukové proudy (`streams`), pro každý směr jeden. Pro reprezentaci tohoto hovoru byla zvolena struktura `TCALL` a její protějšek v databázi v podobě tabulky `calls`. Tato struktura obsahovala všechny informace o daném hovoru, a to hlavně čas začátku, konce, stav hovoru a číslo, IP adresu, port a jméno volaného a volajícího. V prvním řešení kopírovala struktura `TCALL` data získaná ze zpráv protokolu SIP, později jsem přidal i podporu protokolu Skinny. Díky tomu se zobecnil formát ukládaných dat — struktura `TCALL` (a tedy i tabulka v databázi) mohla obsahovat hovory z obou těchto protokolů. Popis jednotlivých položek v této struktuře je v tabulce 4.1.

Hlavní logika aplikace spočívala v průběžném aktualizování informací podle dat extrahovaných z právě odchycených packetů a aktualizací příslušného řádku v databázi. Při zachycení nového hovoru byla tedy vytvořena v paměti nová instance této struktury a vložen nový řádek do tabulky `calls` v databázi. Zachycení packetu s novými informacemi o hovoru způsobilo v paměti aktua-

Datový typ v jazyce C	Název	Popis
`char *`	`protocol`	Protokol, kterého se tyto data (SIP nebo Skinny)
`char *`	`sig_src_ip`	Zdrojová IP adresa signalizace
`char *`	`sig_dst_ip`	Cílová IP adresa signalizece
`char[CALL_ID_SIZE]`	`call_id`	Identifikátor hovoru
`CALL_STATUS`	`call_status`	Stav hovoru
`char *`	`src_ip`	Zdrojová IP adresa zvukových dat
`char *`	`dst_ip`	Cílová IP adresa zvukových dat
`unsigned int`	`src_port`	Zdrojový port zvukových dat
`unsigned int`	`dst_port`	Cílový port zvukovách dat
`char[64]`	`src_num`	Telefonní číslo volajícícho
`char[64]`	`dst_num`	Telefonní číslo volaného
`int`	`format`	Identifikátor formátu zvukových dat
`char[PATH_MAX]`	`dumpfile`	Název souboru, do kterého se ukládají packety zvukového proudu
`pid_t`	`child_pid`	ID procesu, který odchytává packety patřící ke zvukovému proudu tohoto hovoru
`timeval`	`invited`	Čas zachycení první inforamce o hovoru
`timeval`	`rejected`	Čas zachycení informace o zrušení hovoru
`timeval`	`started`	Čas samotného začátku hovoru (začátku nahrávání)
`timeval`	`finished`	Čas ukončení hovoru

Tabulka 4.1: Popis struktury TCALL

lizaci dat a změnu záznamu v databázové tabulce. Po zachycení packetu obsahujícího informace o ukončení hovoru byla tato struktura z paměti vymazána a v databázi pouze změněn stav na „ukončený“. Toto je pouze zjednodušený popis celé logiky, podrobný popis je v bakalářské práci ([1]), případně v předchozí práci, na které tato byla založena [2]. Tento základní princip zůstal při návrhu nové architektury nezměněný — tedy to, že existuje v paměti nějaký model aktuálně známých informací o odchytávaném hovoru, které se aktualizují a upravují podle extrahovaných informací ze zachycených packetů.

Tuto struktuu, jakožto základní strukturu určenou pro uchování informací o hovoru, bylo potřeba změnit — přístup pomocí struktury `TCALL` je dostačující pouze pro jednoduché hovory mezi dvěma účastníky. Pro konferenční hovory je nutné použít jiný přístup — hovor může obsahovat více zvukových proudů, každý začíná a končí v jiný čas a může obsahovat jinak kódovaná zvuková data. Zároveň je nutné počítat s tím, že při konferenčích hovorech nemusí být schopen běžící program odchytit signalizaci všech účastníků, ale jenom některých z nich. Kromě této koncepční změny je potřeba podobným způsobem upravit i databázový model. Podrobněji jsou tyto změny popsány a vysvětleny v kapitole 6.

4.2 Webové rozhraní

Původní projekt obsahoval kromě výše popsané klíčové funkcionality i grafické rozhraní (GUI) ve formě webové aplikace napsané v jazyce PHP na serverové části [3] a v HTML a CSS bez použití frameworku na straně klienta. Tato aplikace využívala již zmíněnou databázi, umožňovala přihlášení různých uživatelů, nastavení jejich oprávnění, nastavování filtrů pro odchytávání hovorů a nastavení rekordérů a dekóderů. Po nutných úpravách databázového modelu pro podporu konferenčních hovorů je toto grafické rozhraní pro tuto práci nepoužitelné a bylo by potřeba ho patřičně upravit, aby podporovalo nový datový model a umožňovalo správně přehrávat a mixovat zachycené zvukové proudy konferenčních hovorů. Proto není součástí této práce.

4.3 Možnosti konfigurace

Běh aplikace bylo možné ovlivňovat podle možností a nastavení upravitelných v textovém konfiguračním souboru, který aplikace četla při startu. Stejná možnost zůstala i v novém návrhu aplikace, pouze se změnily některé direktivy. Nový návrh direktiv a jejich význam je popsán v odstavci 7.3.

4.4 Modulárnost

Jedním z hlavním konceptů předchozího projektu byla modulárnost, a to do té míry, že část aplikace zachytávající packety (rekordér) fungovala nezávisle

na druhé části určené pro dekódování zachycených packetů a extrakci zvukových dat z nich (dekódér). Rekordér pouze ukládal do souboru zachycené packety a název souboru uložil do databáze. Dekódér potom, jako samostatný proces s možností spuštění kdykoli později, dekódoval z těchto packetů zvuková data, která následně ukládal do zvukových souborů pro možnost pozdějšího přehrání. Výhoda v tomto řešení je v oddělených závislostech pro obě části a hlavně v nižší výpočetní náročnosti pro rekórder, který není zatížený dekódováním přijatých zvukových dat. V případě většího provozu na síti může být zpracovávání zvukových dat zachycených hovorů výkonnostní problém. Dekóder, který obsahuje mnohem náročnější dekódování a kódování zvuku, může běžet na jiném zařízení a tím tento problém řešit. Nevýhodou takového řešení je složtější implementace a konfigurace, kdy se musí správně nastavit obě části. Hlavně kvůli složitější implementaci nebyla tato modulárnost zachována — současná verze neukládá zachycené packety, ale rovnou se je snaží dekódovat a ukládat z nich extrahovaná zvuková data do souborů. Do databáze se k jednotlivých proudům ukládá rovnou název souboru s dekódovanými zvukovými daty.

Kapitola 5

Skinny protokol

Tento protokol patří mezi signalizační protokoly VoIP komunikace (do stejné kategorie patří například protokoly SIP nebo H.323, více informací například v [8]). Celý originální název tohoto protokolu je Skinny Client Control Protocol, zkráceně SCCP nebo Skinny. Jedná se o proprietární VoIP protokol používaný zařízeními společnosti Cisco, který pracuje na protokolu TCP na portu 2000. Na rozdíl od protokolu SIP, který je svojí strukturou podobný textovému protokolu HTTP, nepoužívá Skinny pro přenos dat text, ale binární data s předem definovanou délkou a strukturou. Pro šifrovanou komunikaci se používá TLS na portu 2443 (také nad TCP). Šifrování je dostupné od Call Manageru verze 4 [9].

Pro fungování telefonů používajících tento protokol je potřeba, aby ve stejné síti, ve které jsou tyto telefony zapojeny, byl také přítomen a dostupný server se systémem nazývaným Cisco Unified Communications Manager, dále zmiňovaný v této práci jako Call Manager nebo CCM (Cisco Call Manager). Call Manager zajišťuje komunikaci mezi jednotlivými telefony, reaguje na jejich události (např. zvednutí nebo položení sluchátka, stisk tlačítek) a zajišťuje spojení hovoru po vytočení čísla. Kromě toho poskytuje operační systém a konfigurační soubory (pomocí protokolu TFTP) pro jednotlivé telefony, které si tato data samy při prvním zapojení stahují a následně používají pro síťovou komunikaci.

5.1 Životní cyklus Skinny telefonu

Tato sekce popisující životní cyklus telefonu je s úpravami převzata z mé předchozí práce [1].

Po připojení Cisco telefonu do sítě LAN telefon požádá o IP adresu přes protokol DHCP. Server, který na tuto žádost odpovídá může být úplně jiný, než Call Manager. Důležité ale je, že tato DHCP odpověď obsahuje kromě požadované IP adresy telefonu i IP adresu TFTP serveru (jako DHCP op-

tion 150), ze kterého si následně telefon stáhne obraz systému a další nutné systémové konfigurační soubory, které bude poté používat [5].

Po stažení těchto dat se telefon zaregistruje u Call Manageru. Postup registrace a popis zpráv, které si při ní vyměňují telefon s Call Managerem není zde potřeba uvádět. V případě zájmu je možné více informací najít například v kapitole „SCCP Call Flows“ (Appendix D) v [10]. Po úspěšné registraci posílá telefon Call Manageru veškeré události, na které Call Manager reaguje dalšími instrukcemi. Ve Skinny fungují telefony jako jakési „opravdu hloupé terminály“ [11], které odesílají zprávy o stiscích všech tlačítek a jiných událostech a očekávají od Call Manageru instrukce, co zobrazit na displeji, jaké tóny přehrávat, jak a kdy vyzvánět nebo na jakých portech vysílat nebo přijímat data hovoru. Podrobnější popis vysílaných zpráv je na obrázku 5.1.

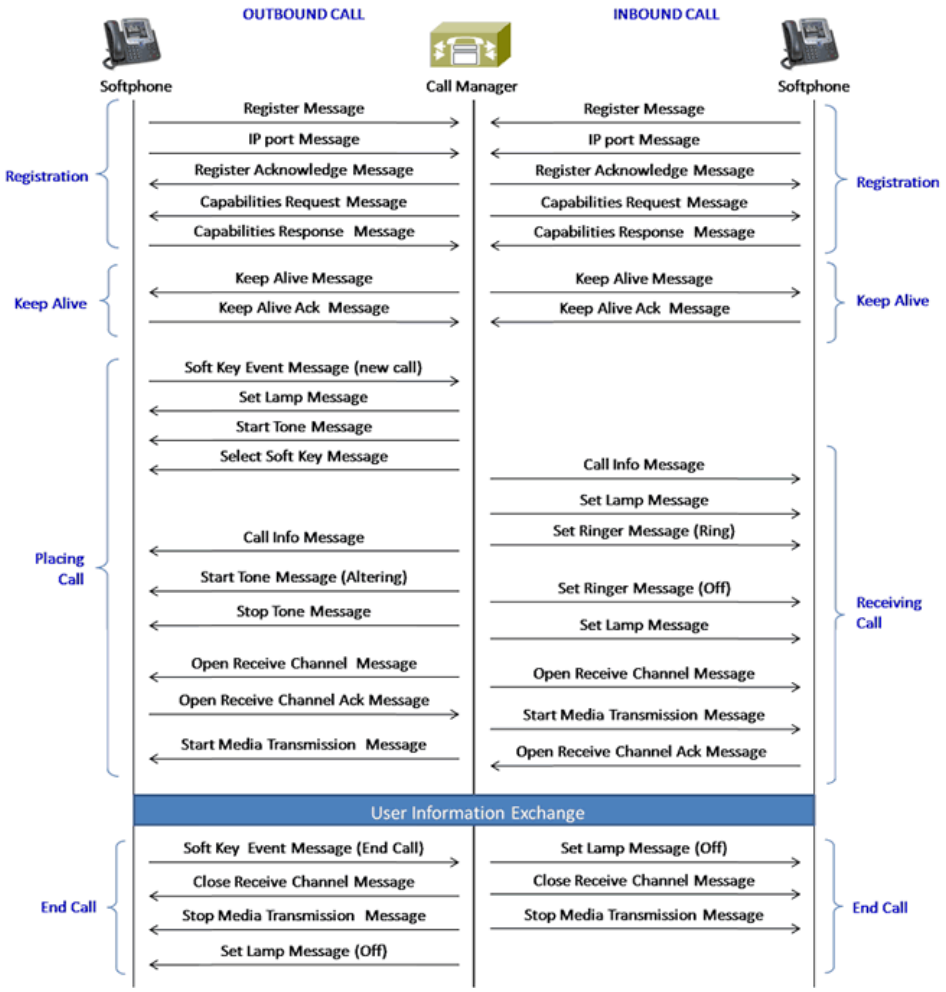
Zde se budu soustředit pouze na zprávy, které mají nějaký vliv na samotný průběh hovoru. Některé zprávy obsahují pouze typ zprávy a v podstatě žádná další relevantní data (např. *OffHook* pouze informuje Call Manager, že bylo zvednuto sluchátko) a některá i několik desítek dalších položek s detailními informacemi (např. *CallInfo* obsahuje informace o tom, kdo komu volá, a to jak číslo, tak název, informace o směru hovoru a přesměrování).

5.2 Struktura zpráv

Každý packet tohoto protokolu může obsahovat jednu nebo více zpráv. Každá taková zpráva má hlavičku, která obsahuje informaci o délce těla zprávy a verzi protokolu. Po hlavičce následuje samotné tělo zprávy, které obsahuje údaj o typu zprávy a následně další data, jejichž struktura je daná právě typem zprávy. Jednotlivé zprávy jsou řazeny bez nějakého odlišení v packetu za sebou. Jediná mně známá možnost, jak rozlišit, že daný packet obsahuje více zpráv, je porovnání délky packetu s délkou zprávy uvedenou v hlavičce zprávy Skinny protokolu. Pokud je délka packetu větší, předpokládá se, že packet obsahuje další zprávy. Stejným způsobem se rozhoduje o přítomnosti třetí a další zprávy v packetu.

Vzhledem k proprietární povaze protokolu SCCP se mi nepodařilo najít jeho přesnou specifikaci. Většina informací pochází z pozorování zachycených packetů pomocí programu Wireshark. Tento program obsahuje dissector pro protokol Skinny, díky kterému se mi podařilo zjistit pravděpodobný význam, jméno a pořadí jednotlivých položek v tomto protokolu. Popis datových prvků v jednotlivých zprávách pochází tedy z tohoto zdroje.

Jednotlivé položky zprávy jsou ve většině případů binární čísla s daným počtem bytů a pořadím, přičemž pořadí jednotlivých bytů je *little endian* (nejméně významný byte na nejnižší adrese). Tím se tedy liší od zbytku síťové komunikace, která používá pro všechny hodnoty *big endian* (nejméně významný byte na nejvyšší adrese). O tomto pořadí bytů ve Skinny jsem nenašel žádnou informaci, a to ani v [5], kde je protokol Skinny jinak ve-



Obrázek 5.1: Graf časové posloupnosti vysílaných Skinny zpráv v průběhu typického hovoru [12]. Na obrázku jsou chybně vyznačeny zprávy *KeepAlive* a *KeepAliveAck*, správně by měly být opačně. Zpráva *KeepAlive* je posílána z telefonu na Call Manager a *KeepAliveAck* z Call Manageru na telefon

Hlavička zprávy		Tělo zprávy	
Počet bytů v těle	Verze protokolu	Typ zprávy	Samotná data zprávy
4 byty	4 byty	4 byty	

Tabulka 5.1: Základní struktura Skinny zprávy

lice podrobně popsán. Nikde v protokolu (alespoň ve verzích, které jsem měl k dispozici) není žádná proměnná, která by umožňovala zvolit pořadí bytů; předpokládám tedy, že se jedná o *little endian* vždy a že je to vlastnost protokolu, přestože není nikde zmíněná.

Jak už bylo zmíněno výše, každá zpráva Skinny protokolu obsahuje hlavičku. Tato hlavička se skládá ze dvou čtyřbytových položek. První určuje velikost těla zprávy v bytech, druhá pak verzi Skinny protokolu.

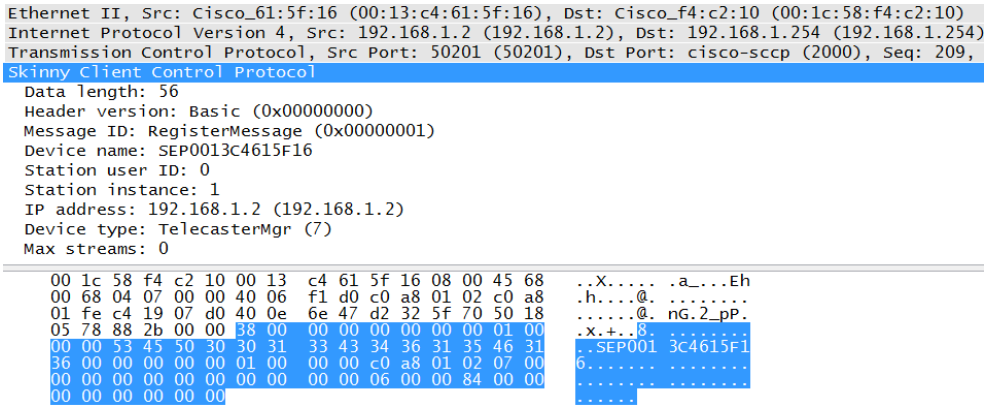
Ve všech zdrojích, které jsem měl k dispozici, jsem našel pouze informaci, že jako verze se standardně používá 0. V některých zdrojích bylo dokonce uvedeno, že tyto čtyři byty jsou „vyhrazeny“ („reserved“) a nulu obsahují vždy. V reálném prostředí byla tato verze ovšem 22.

Po hlavičce následuje tělo zprávy. Rozlišení na hlavičku a tělo zprávy je založeno na tom, že velikost zprávy v hlavičce protokolu je uvedena bez prvních osmi bytů. Předpokládám tedy, že těchto prvních osm bytů tvoří hlavičku a číslo označující délku zprávy určuje délku zbytku zprávy, tedy těla.

Prvním údajem v těle každé zprávy je čtyřbytová číselná hodnota, která určuje typ zprávy. Struktura zbytku zprávy je specifická pro každý typ zprávy. Podle první hodnoty v těle lze tedy naprosto jednoznačně určit strukturu zbytku zprávy. Obecně se dá říct, že většina položek v těle je číselná hodnota o velikosti čtyř bytů. Jedna z vyjímek je textový řetězec, který má pevně danou délku, každý znak je v ASCII kódu, a za posledním znakem je byte s hodnotou 0. V kódování textového řetězce je ale i jediná vypozorovaná vyjímka mezi verzemi protokolu. Ve verzi 0 je to řešeno právě fixní délkou, ale ve verzi 22, kterou jsem pozoroval při reálném provozu, nemají řetězce maximální fixní délku. Fakt, že řetězec je ukončen bytem s hodnotou 0, ale zůstává. Pro zjišťování délky těchto textových položek je tedy potřeba najít v bytech hodnotu 0. Další položky v těle zprávy následují ihned za tímto bytem. Tento způsob kódování textu ztěžuje parsování zpráv, protože se nelze spolehnout, že požadovaná hodnota bude na předem známé pozici.

Základní strukturu každé zprávy ve Skinny protokolu a velikosti jednotlivých položek v každé zprávě zobrazuje tabulka 5.1.

Výše popsaná struktura zprávy je vidět například na obrázku 5.2, který zobrazuje registrační zprávu posílanou telefonem Call Manageru zachycenou a zobrazenou v aplikaci Wireshark.



Obrázek 5.2: Screenshot z aplikace Wireshark ukazující jednu z mnoha možných Skinny zpráv [5].

Typ zprávy	Název zprávy	Stručný popis
Zprávy posílané z telefonu na Call Manager		
0x0000	*KeepAliveMessage*	Kontrola dostupnosti telefonu
0x0001	*RegisterMessage*	Registrace telefonu k příslušnému Call Manageru
0x0003	*KeypadButtonMessage*	Stisknutí číselné klávesy při vytáčení
0x0006	*OffHookMessage*	Zvednutém sluchátka
0x0007	*OnHookMessage*	Položení sluchátka
0x0022	*OpenReceiveChannelAck*	Potvrzení otevření portů pro přijímání zvukových dat
0x0154	*StartMediaTransmissionAck*	Potvrzení otevření portů pro odesílání zvukových dat

Tabulka 5.2: Výběr typů zpráv Skinny protokolu, posílaných z telefonu na Call Manager, další zprávy je možné najít v [6].

5.3 Seznam typů zpráv

Protokol Skinny může obsahovat teoreticky veliké množství zpráv rozlišených podle třetí čtyřbytové hodnoty v binárních datech zprávy. Zdrojový kód aplikace Wireshark, ze kterého jsem čerpal nejvíce informací o tomto poroktolu ([6]), rozlišuje podle typu 159 různých zpráv. Většinu těchto zpráv se mi nepodařilo při testování jednoduchých ani konferenčních hovorů zachytit. Výběr několika typů těchto zpráv a jejich popis je uveden v tabulkách 5.2 a 5.3.

Pro účely detekce začátku a konce hovoru, včetně konferenčního, a informace o něm, stačí rozlišovat pouze několik málo typů zpráv, které obsahují

Kód zprávy	Název zprávy	Stručný popis
Zprávy posílané z Call Manageru na telefon		
`0x0081`	*RegisterAckMessage*	Potvrzení žádosti o registraci telefonu
`0x0085`	*SetRingerMessage*	Nastavení stavu vyzvánění
`0x008A`	*StartMediaTransmission*	Požadavek na začátek odesílání zvukových dat
`0x008B`	*StopMediaTransmission*	Požadavek na ukončení odesílání zvukových dat
`0x008F`	*CallInfoMessage*	Informace o hovoru, obsahuje čísla volaného a volajícího, informace o přesměrování a další, nevýznamné pro další zpracování
`0x0099`	*DisplayTextMessage*	Požadavek na zobrazení textu na displeji
`0x009A`	*ClearDisplay*	Požadavek na vymazání textu na displeji
`0x0100`	*KeepAliveAckMessage*	Potvrzení přijetí zprávy *KeepAliveMessage*
`0x0105`	*OpenReceiveChannel*	Požadavek na otevření portů pro příjímání zvukových dat
`0x0106`	*CloseReceiveChannel*	Požadavek na uzavření portů pro příjímání zvukových dat
`0x0111`	*CallStateMessage*	Informace o změně stavu hovoru, obsahuje položku callState, která určuje požadovaný stav
`0x011D`	*DialedNumberMessage*	Informace o právě vytočeném čísle, odeslána po zadání všech čísel jako potvrzení vytočení čísla
`0x014A`	*CallInfoMessageV2*	Stejné jako *CallInfoMessage*, pouze s dynamickou délkou řetězců (ukončených znakem 0, ne statickou délkou, jako u *CallInfoMessage*)

Tabulka 5.3: Výběr typů zpráv Skinny protokolu posílaných z Call Manageru na telefon, další zprávy je možné najít v [6].

všechny potřebné informace — tj. jméno a číslo volaného a volajícícho, informace o stavu hovoru, IP adresy a porty pro rozpoznání a odchycení zvukových proudů mezi dalšími packety na síti a formát těchto zvkukových dat. Popis těchto zpráv a údajů z nich získávaných je více popsán v 6.6.

Kapitola 6

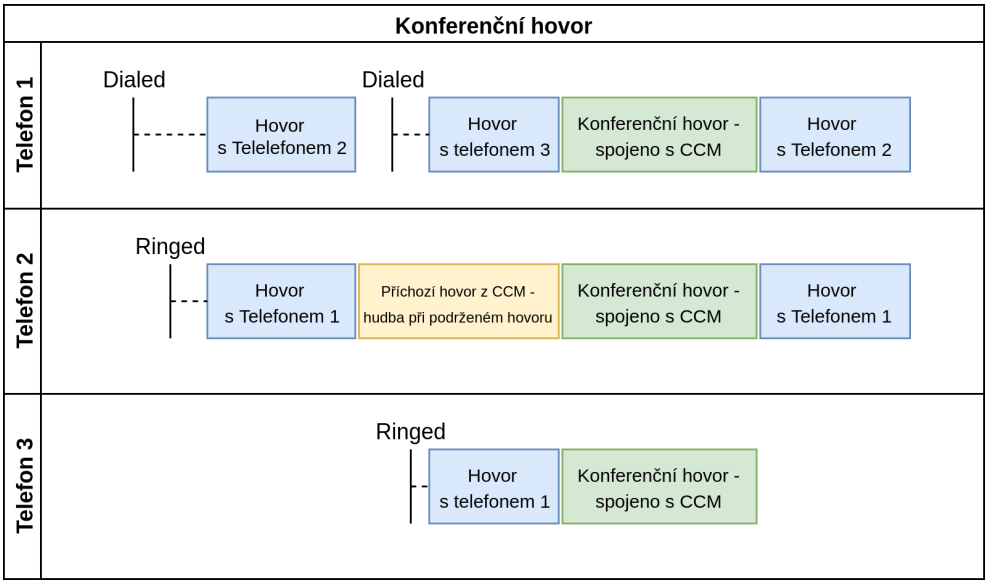
Návrh řešení a realizace

Jak už bylo zmíněno výše, bylo potřeba navrhnout novou architekturu a datový model. Tato kapitola tento nový datový model popisuje. Původní návrh počítal s přesně dvěma účastníky na jeden hovor, proto pro jeden hovor byla zavedena jenom jedna entita (TCALL) obsahující časy začátku vyzvánění, začátku hovoru, konce hovoru, informace o identitě volajícho a volaného (telefonní číslo a jméno) a odkaz na příslušný soubor obsahující zachycená audio data v packetech.

Pro konferenční hovory je toto ale nedostačující, proto jsem navrhl nový datový model skládající se z „párů“ (v kódu a databázi označovaných jako `pair`). Každý pár obsahuje aktuální stav (viz tabulka 6.1) a informaci o orientovaných zvukových proudech mezi dvěma zařízeními, a to buď mezi dvěma telefony nebo mezi telefonem a CCM. Každý pár má tedy informaci o až dvou zvukových proudech — jeden pro odchozí směr a druhý pro příchozí. V určitých situacích může dojít k tomu, že je tento zvukový proud pouze jeden. Tato situace nastává například při podrženém hovoru, kdy lze zachytit pouze zvukový proud směrem od CCM k telefonu (hrající hudba indikující

STATE_UNKNOWN	Inicializační stav při zachycení prvních informcí o páru, kdy ještě není znám konkrétní stav.
DIALING	Vytáčení — byla zachycena zpráva značící vytáčení čísla a to vytáčením (odchozí hovor).
RINGING	Zvonění — byla zachycena zpráva značící začátku hovoru a to zvoněním (příchozí hovor).
CALLING	Právě probíhající spojení — hovor, respektive spojení mezi dvěmi účastníky právě probíhá a packety tohoto hovoru jsou odchytáván a ukládány.
FINISHED	Hovor skončil.

Tabulka 6.1: Stavy páru a jejich popis



Obrázek 6.1: Diagram jednoduchého konferenčního hovoru a zobrazení jednotlivých párů podle navržené architektury.

podržený hovor), od telefonu směrem k CCM žádný zvukový proud není — v testovacím provozu zachycen nebyl. Jednotlivé stavy párů tvoří jednoduchý stavový automat. Při zpracování zachycených zpráv dochází k přechodům mezi stavy. Po dosažení stavu `FINISHED` je pár z paměti odstraněn.

Na obrázku 6.1 je vidět diagram jednoduchého konferenčního hovoru se třemi účastníky a jsou znázorněny jednotlivé páry tak, jak je aplikace chápe. Konferenční hovor obsahuje tři zařízení, označená jako Telefon 1, Telefon 2 a Telefon 3. Časová osa je zleva doprava. Scénář je následující: nejprve zavolá Telefon 1 na Telefon 2 a naváže hovor, poté podrží hovor pro Telefon 2 a vytáčí Telefon 3, po spojení Telefonu 1 a Telefonu 3 se vytvoří konferenční hovor mezi všemi třemi telefony, tento hovor poté ukončuje Telefon 3, pak následuje jednoduchý hovor mezi Telefony 1 a 2. Popisek Dialed určuje čas, který se uloží jako čas vytáčení čísla pro odchozí hovor, popisek Ringed zase definuje čas vyzvánění příchozího hovoru. Jednotlivé obdélníky určují jednotlivé páry, levá strana určuje začátek a pravá konec tohoto páru. Modře označené páry jsou ty, které jsou přímo mezi dvěma telefony. Žlutý pár je pro telefon v průběhu podrženého hovoru — obsahuje pouze příchozí zvukový proud (hrající hudbu při podrženém hovoru). Zeleně jsou pak označeny páry v průběhu samotného konferenčního hovoru — v tomto případě je jako druhé zařízení pro každý telefon CCM, ze kterého telefon přijímá zmixovaný příchozí zvukový proud od ostatních účastníků konference.

6.1 Datový model v paměti

Každý pár obsahuje v datovém modelu místní IP adresu a port (`node_this`) a vzdálenou IP adresu a port (`node_that`). Kromě těchto údajů obsahuje také IP adresy a porty signalizačních packetů, informaci o jaký protokol se jedná (v současné době pouze Skinny), ID páru získaného ze síťového protokolu a čas, kdy došlo k vytáčení, vyzvánění, začátku a konci hovoru. Časy vytáčení a vyzvánění jsou platné pouze pro páry na začátku (konferenčního hovoru), poté už k vytáčení ani k vyzvánění nedochází. Implementace páru je tvořena datovou strukturou `pair_t` a `pair_data_t`, která je obsažena uvnitř `pair_t`. Uvnitř struktury `pair_t` jsou metadata získaná z packetů, uvnitř `pair_data_t` informace extrahované přímo z packetů. Toto rozdělení potom zpřehledňuje práci s těmito daty, zejména porovnávání již známých informací s informacemi extrahovanými ze zachycených packetů a jejich doplnění o takto získaná data.

Zde je struktura `pair_t` tak, jak je definovaná ve zdrojovém kódu:

```
typedef struct pair {

    pair_state_t state;
    const char * database_id;
    pair_data_t data;

    timestamp_t dialed;
    timestamp_t ringed;
    timestamp_t started;
    timestamp_t finished;

    char filename_in[PATH_MAX];
    char filename_out[PATH_MAX];

} pair_t;
```

Popis jednotlivých členských proměnných struktury `pair_t`:

- `state` — Výčtový typ obsahující stav páru popsaný v tabulce 6.1.
- `database_id` — ID řádku v databázi v tabulce `pairs`. Určené pro synchronizaci nově zachycených dat s databází.
- `data` — Struktura `pair_data_t` obsahující další informace o páru, popsaná níže.
- `dialed` — Čas zachycení packetu s informací, že se jedná o vytáčení čísla (odchozí hovor).
- `ringed` — Čas zachycení packetu s informací, že se jedná o zvonění (příchozí hovor).
- `started` — Čas zachycení packetu s informací, že se jedná o začátek hovoru.

- `finished` — Čas zachycení packetu s informací, že se jedná o konec hovoru.
- `filename_in` — Název souboru, kam se ukládá nahrávaný příchozí zvukový proud.
- `filename_out` — Název souboru, kam se ukládá nahrávaný odchozí zvukový proud.

Následuje struktura `pair_data_t` tak, jak je definovaná ve zdrojovém kódu:

```
typedef struct pair_data {

    const char * protocol;

    uint32_t call_id; // 0 means not set

    /**
     * unique pair identifier
     */
    uint32_t pair_id; // 0 means not set

    node_identifier_t node_this;
    node_identifier_t node_that;

    pair_state_t requestedState;

    int audio_format;

    // identifiers of signaling communication
    char sig_from_ip[INET6_ADDRSTRLEN];
    unsigned int sig_from_port;
    char sig_to_ip[INET6_ADDRSTRLEN];
    unsigned int sig_to_port;

    int stream_in, stream_out;

    struct pair_data * next; // linked list

} pair_data_t;
```

Popis jednotlivých členských proměnných struktury `pair_data_t`:

- `protocol` — Název protokolu. V současné implementaci vždy hodnota „Skinny“.
- `call_id` — Identifikátor hovoru, nikoli páru. Určen pro spolehlivější určení, zda zachycená data odpovídají známému páru.
- `pair_id` — Unikátní identifikátor páru.
- `node_this` — Identifikátor lokálního zařízení (struktura obsahující IP adresu, port, jméno a telefonní číslo)

- `node_that` — Identifikátor vzdáleného zařízení (struktura obsahující IP adresu, port, jméno a telefonní číslo)
- `requestedState` — Požadovaný stav páru, použito při extrakci struktury `pair_data_t` ze zachyceného packetu. Při nastavení způsobuje přechod stavu ve stavovém automatu příslušného páru.
- `audio_format` — Formát zvuku zachycených zvukových proudů.
- `sig_from_ip` — IP adresa signalizace zdrojového zařízení.
- `sig_from_port` — Port signalizace zdrojového zařízení.
- `sig_to_ip` — IP adresa signalizace cílového zařízení.
- `sig_to_port` — Port signalizace cílového zařízení.
- `stream_in` — Příznak určující, jestli se má po zachycení packetu začít nahrávat příchozí zvukový proud, pokud jsou dostupné potřebné informace.
- `stream_out` — Příznak určující, jestli se má po zachycení packetu začít nahrávat odchozí zvukový proud, pokud jsou dostupné potřebné informace.
- `next` — Ukazatel na následující strukturu. Použito pouze v případě, kdy se extrahují jednotlivá data párů z přijatého packetu, kde jich může v jednom packetu být více. Díky této hodnotě vzniká spojový seznam (*linked list*). V ostatních částech aplikace se s touto hodnotou nepracuje.

6.2 Databázový model

Jak už bylo zmíněno výše, původní návrh databázového modelu není pro uchovávání informací o konferenčních hovorech dostačující. Proto jsem vytvořil nový návrh, který reflektuje model datových struktur v paměti. Původní návrh se strukturou `TCALL` obsahoval jako hlavní tabulku `calls`, v novém návrhu je hlavní tabulkou `pairs`, která obsahuje reference přes cizí klíče na tabulku `streams`. Tabulka `pairs` obsahuje informace o jednotlivých párech, tabulka `streams` pak informace o jednotlivých zvukových proudech (streams) vázaných na tyto páry. Podrobnější popis a vysvětlení jednotlivých sloupců obsahují tabulky 6.2 a 6.3. Databázový model se tedy shoduje s datovým modelem v paměti v tabulce (a struktuře) `pairs` (`t_pair`), která obsahuje data pro jeden konkrétní pár. Datový model v paměti obsahuje ve struktuře `pair_data` informace o místním a vzdáleném zařízení. Tyto informace obsahují data potřebná pro filtrování síťového provozu a správnou detekci a nahrávání (například IP adresa a port signalizace), do databáze už se neukládají. V databázovém modelu jsou na pár navázané entity reprezentující

Název	Datový typ v databázi	Popis
`pair_id`	`SERIAL PRIMARY KEY`	Primární klíč, identifikátor páru
`call_id`	`integer`	Identifikátor hovoru
`stream_in_id`	`integer`	Cizí klíč referencující řádek v tabulce streams, reference na informace o příchozích zvukovém proudu
`stream_out_id`	`integer`	Cizí klíč referencující řádek v tabulce streams, reference na informace o odchozím zvukovém proudu
`dialed`	`timestamp`	Čas vytočení čísla (pro odchozí hovor)
`ringed`	`timestamp`	Čas začátku vyzvánení (pro příchozí hovor)
`this_number`	`text`	Telefonní číslo zdrojového zařízení
`this_name`	`text`	Jméno zdrojového zařízení
`that_number`	`text`	Telefonní cílového zařízení
`that_name`	`text`	Jméno cílového zařízení

Tabulka 6.2: Popis databázové tabulky `pairs`

jednotlivé proudy (`streams`), nikoli entity reprezentující příjemce a odesílatele, jak je to ve strukturách v paměti. Pro pozdější zpracování je výhodnější reprezentovat hovory takto, pro detekci hovorů při běhu aplikace je ale naopak výhodnější mít struktury pro stranu příjemce a odesílatele. Entita pro reprezentaci zvukového proudu se vytváří až při začátku nahrávání hovoru.

6.3 Podpora podrženého hovoru (HOLD)

Díky tomuto návrhu jsem získal kromě podpory konferenčních hovorů i podporu pro „podržený“ hovor (HOLD). Podržený hovor vzniká ve chvíli, kdy jeden z účastníků stiskne na telefonu tlačítko HOLD. V případě hovoru mezi dvěma účastníky vytvoří Call Manager zvukový proud s generovanou hudbou druhému účastníku. V průběhu podrženého hovoru je tedy vytvořený pouze jeden zvukový proud, místo standardně dvou.

6.4 Podpora konferenčních hovorů zachycených jen z části

Při detekci konferenčního hovoru s více účastníky se může stát, že nebude zachycena signalizační komunikace mezi všemi účastníky — například v případě pěti účastníků může být k dispozici jenom komunikace dvou zařízení v lokální

Název	Datový typ v databázi	Popis
`stream_id`	`SERIAL PRIMARY KEY`	Primární klíč, identifikátor zvukového proudu
`this_ip`	`text`	IP adresa první strany spojení
`this_port`	`integer`	port první strany spojení
`that_ip`	`text`	IP adresa druhé strany spojení
`that_port`	`integer`	port druhé strany spojení
`started`	`timestamp not null`	Čas vytvoření spojení, začátku zvukového proudu
`finished`	`timestamp`	Čas ukončení spojení, konce zvukového proudu
`file`	`text not null`	Název souboru s dekódovanými zvukovými daty ze zachycených packetů

Tabulka 6.3: Popis databázové tabulky `streams`

síti, ale záznam odeslaných a přijatých packetů zbývajících třech zařízení už k dispozici nebude. I v tomto případě bude program fungovat. Problém bude s rozpoznáním ostatních, pro program neviditelných, zařízení. Pro dvě dostupná zařízení budou k dispozici pouze 4 proudy (pro každý zařízení dva — odchozí a příchozí). Zvuková data všech účastníků budou i tak dostupná v příchozích proudech známých zařízení, pouze budou smíchaná (zmixovaná) dohromady prostředníkem, tedy zařízením CCM. Stejně tak, pokud budou dostupná data pouze jednoho účastníka, záznam v databázi potom bude obsahovat pouze jeden pár a dva proudy, jeden odchozí a jeden příchozí, ve kterém budou smíchány zvukové stopy ostatních účastníků konference.

Jiné řešení v podstatě z principu protokolu Skinny není možné. Požadované informace nejsou ani v jiných signalizačních zprávách. Není mi ani známo, jak zjistit počet účastníků takového konferenčního hovoru jinak, než detekcí různých zvukových stop (případně hlasů) ze zachycených zmixovaných zvukových proudů. Takový úkol už je mnohem náročnější a je i mimo rozsah této práce.

6.5 Popis detekce signalizace hovoru

Proces detekce signalizace hovoru probíhá následovně:

1. Extrakce signalizačních packetů

   V první fázi je nutné nejprve extrahovat packety patřící k VoIP signalizačnímu protokolu. Současná implementace pracuje pouze se Skinny protokolem, ale neměl by být problém přidat bez větších obtíží protokol jiný

(např. už zmiňovaný SIP). Tato funkcionalita byla z větší části převzata z přechozího projektu — jedná se pouze o extrakci binárních dat ze zachycených packetů.

2. Dekódování packetů na strukturu `pair_data`

   Pro účely nové architektury jsem zavedl novou entitu — `pair_data`. Tato entita obsahuje informaci o protokolu a volitelně i další informace, které se podaří z packetu extrahovat — tedy ID (unikátní identifikaci) páru jako celé číslo, IP adresu a port místního (`node_this`) a vzdáleného (`node_that`) zařízení, IP adresu a port signalizačního protokolu, požadovaný stav páru (`requested_state`) a kód formátu zvukových dat. Cílem této fáze je extrahovat z packetu co nejvíce dostupných informací. Ve Skinny protokolu obecně platí, že jedna Skinny zpráva se může převést na jednu nebo žádnou strukturu `pair_data`. Jeden packet může obsahovat více Skinny zpráv, proto výstupem této fáze může být 0 nebo více struktur `pair_data`.

3. Nalezení nebo vytvoření nových párů

   Vstupem do této fáze jsou struktury `pair_data` získané z fáze předchozí. Pro každou tuto strukturu se aplikace pokouší nalézt odpovídající známý pár, nebo, pokud takový není, vytváří nový.

   Nový pár se ale vytváří pouze v případě, že požadovaný stav (hodnota `requestedState`) je nastavený a není FINISHED. Nemá smysl vytvářet nový pár, když jeho stav není známý, nebo pokud by se vzápětí odstranil z paměti, což se děje s páry ve stavu FINISHED.

   Seznam aktuálně sledovaných párů je v paměti tvořen spojovým seznamem, který je pro nalezení odpovídajícho páru lineárně projít. Pro větší počet sledovaných párů (řádově stovky a více) by bylo lepší místo spojového seznamu použít nějakou stromovou strukturu. Řešení spojovým seznamem je ale implementačně jednodušší a pro menší počet hovorů plně dostačující.

   Hledání nebo vytvoření nového páru je v kódu implementováno pomocí funkce `find_or_create_pair()`. Pro každý pár následuje rozhodování, zda tomuto páru odpovídají data v extrahované struktuře `pair_data`. Tato kontrola se provádí ve funkci `pair_matches_pair_data()` a dalšími funkcemi volanými z této. Konvence návratové hodnoty této a dalších funkcí je taková, že hodnota 1 značí shodu, hodnota -1 neshodu a hodnota 0 neschopnost rozhodnout.

   Nejprve je tedy porovnán název protokolu, který musí souhlasit. V aktuální implementaci bude vždy nastaven na hodnotu „Skinny“, tato kontrola je zde hlavně pro snadné rozšíření o jiné protokoly. Následně se kontroluje, zda mají obě struktury nastavenou proměnnou `pair_id`, která jednoznačně identifikuje pár, v tom případě se porovná tato hodnota a další kontroly už se neprovádí. V opačném případě pokračují další kontroly. Následuje

kontrola na shodné IP adresy a porty signalizace pomocí funkce s názvem `pair_data_matches_sig()`, která porovnává tato data v obou strukturách, pokud jsou známa. Pokud nedokáže rozhodnout (informace nejsou kompletní), proces kontroly pokračuje dále. Pokračování kontroly je ve funkci `stream_identifiers_matches()`, kde se nejprve kontroluje, zda se nejedná o sice stejný pár, ale v opačném směru (v tom případě se vrátí hodnota -1 indikující, že se nejedná o shodu) a teprv potom se kontroluje, zda se jedná o stejný směr.

Funkce vrací 1 (shodu) pouze v případě, kdy jsou shodné IP adresy a porty buď nejsou známé nebo jsou známé a shodné.

4. Změna stavu párů

Součástí struktury `pair_data` je také proměnná `requestedState` určující požadovaný stav páru. V případě, že tato proměnná má hodnotu jinou, než STATE_UNKNOWN, dochází ke změně páru do požadovaného stavu. Vyjímka je pouze při požadovaném stavu CALLING, kdy se ještě provádí kontrola na přítomnost potřebných informací k začátku nahrávání, které by bylo touto změnou stavu spuštěno. Potřebné informace k začátku nahrávání (odchytávání a dekódování packetů obsahujících zvuková data) jsou IP adresy a porty. K jednoznačné identifikaci zvukového proudu na síti stačí z celkem čtyř informací (zdrojová a cílová IP adresa a port) pouze informace tři — i tak se jedná o v daném čase přesně identifikovatelné síťové spojení protokolu UDP. Tato vlastnost je využita pro úspěšné odchytávání podržených hovorů, protože v tomto případě není k dispozici před začátkem nahrávání zvukového proudu zdrojový port. Tato informace se totiž posílá ve zprávě `StartMediaTransmission`, která v tomto případě poslána není. Posílá se totiž z CCM na zdrojové zařízení a v případě podrženého hovoru je tímto zdrojovým zařízením CCM, takže tato zpráva se na síti nevyskytne. Naštěstí pro začátek odchytávání síťového spojení stačí pouze tři údaje.

Dalším speciálním stavem je stav FINISHED. Při změně páru do tohoto stavu se ukončí nahrávání hovoru a dotyčný pár je odstraněn ze seznamu párů v paměti.

Každá změna stavu se také ihned zapíše do databáze společně s časovým razítkem, kdy k této změně došlo.

Díky tomuto návrhu je mnohem jednodušší vytvořit podporu pro jiný protokol, než v předchozí verzi. Jediné, co je potřeba, je implementovat dekódování zachycených packetů do struktury `pair_data`. Vše ostatní už je vyřešeno.

6.6 Mapování Skinny zpráv na datovou strukturu `pair_data`

Tato kapitola popisuje mapování (převod) Skinny zpráv na datovou strukturu `pair_data`. Z velkého počtu různých typů posílaných zpráv jsem vybral pouze několik a to ty, které rozhodují o stavu hovoru, obsahují informace o účastnících hovoru, případně informace nutné k nahrávání hovoru, tj. IP adresy a porty pro posílaná zvuková data přes protokol UDP a formát (kódování) těchto dat. Ve zdrojovém kódu se jedná o funkci `skinny_process_message()`, která dostává jako argument binární data extrahovaná ze zachyceného packetu a vrací seznam datových struktur `pair_data` obsahujících data extrahovaná ze Skinny zpráv v daném packetu. Jeden packet totiž může obsahovat více zpráv protokolu Skinny. Pokud packet neobsahuje žádné Skinny zprávy, nebo obsahuje zprávy, které nejsou kontrolovány, může vrácený seznam být i prázdný.

Příznaky `stream_in` a `stream_out` rozhodují o tom, jestli daný pár obsahuje příchozí (`stream_in`) a/nebo odchozí (`stream_out`) zvukový proud. Tento příznak je potom rozhodující pro začátek nahrávání hovoru. Rozdělení na dva příznaky je hlavně kvůli případu, kdy je k dispozici pouze spojení z jedné strany — tato situace nastává při podrženém hovoru, kdy je dostupné pouze příchozí spojení daného zařízení (`stream_in`) a odchozí (`stream_out`) už ne.

Zde je seznam kontrolovaných Skinny zpráv a popis potřebných informacích v nich obsažených:

- `DialedNumber`

  Tato zpráva je posílána z CCM na zařízení v okamžiku, kdy zařízení vytočí číslo.

  Extrahovaná data: místní IP adresa, číslo cílového zařízení.

  Požadovaný stav páru: `DIALING`.

- `CallState`

  Tato zpráva je poslána z CCM na zařízení, když se mění stav hovoru v rámci Skinny protokolu. Obsahuje informaci o novém stavu. Pouze v případě, že se jedná o stav „Ringin“ [1], je tato zpráva zpracována, jinak je ignorována.

  Extrahovaná data: místní IP adresa.

  Požadovaný stav páru: `RINGING`.

- `CallInfo`

  Tato zpráva je během signalizace několikrát poslána ze strany CCM na telefon a obsahuje nejvíce informací o probíhajícím hovoru. Bohužel

[1] Název stavu „Ringin“ je převzat ze zdrojového kódu aplikace Wireshark

neobsauje aktuální stav hovoru, proto samotná tato zpráva nestačí k vytvoření záznamu páru v paměti a v databázi. Tato zpráva se vyskytuje i po skončení zvukového proudu, kdy už je pár označen za dokončený (stav FINISHED) a smazán z paměti. Proto není žádoucí, aby tato zpráva způsobila vytvoření nového páru, pokud nějaký nebyl nalezen na základě informací v této zprávě obsažených.

Extrahovaná data: číslo a jméno zdrojového a cílového zařízení, místní IP adresa.

Požadovaný stav páru: *žádný*.

- `OpenReceiveChannel`

  Tato zpráva je poslána na zařízení ze strany CCM v okamžiku začátku hovoru, respektive příchozího zvukového proudu. Obsahuje pouze informace o tom, z jaké IP adresy má zařízení čekat příchozí spojení se zvukovými daty. Zprávy týkající se už zvukového proudu obsahují ID páru, které v protokolu Skinny odpovídá datům označovaným jako `passThruPartyId`.

  Extrahovaná data: formát zvukových dat, ID páru, vzdálená IP adresa, místní IP adresa, příznak `stream_in`.

  Požadovaný stav páru: `CALLING`.

- `OpenReceiveChannelAck`

  Tato zpráva signalizuje potvrzení příchozího spojení. Je tedy odeslána ze strany zařízení k CCM.

  Extrahovaná data: ID páru, místní IP adresa a port, příznak `stream_in`.

  Požadovaný stav páru: `CALLING`.

- `StartMediaTransmission`

  Jedná se o požadavek ze strany CCM na začátek vysílání zvukového proudu na požadovanou IP adresu a port. Uvnitř zprávy jsou přítomny všechny potřebné údaje (dvě IP adresy a port), díky tomu je zachycení této zprávy dostatečné ke spuštění nahrávání zvukového proudu.

  Extrahovaná data: ID páru, místní IP adresa, vzdálená IP adresa a port, formát zvukových dat proudu, příznak `stream_out`.

  Požadovaný stav páru: `CALLING`

- `StartMediaTransmissionAck`

  Tato zpráva potvrzuje začátek vysílání. Je poslána ze strany zařízení na CCM.

  Extrahovaná data: ID páru, lokální IP adresa a port, příznak `stream_out`.

  Požadovaný stav páru: `CALLING`.

- `CloseReceiveChannel` a `StopMediaTransmission`

  Formát těchto dvou zpráv je shodný, proto jsou zpracovávány jako jeden typ. V obou případech se jedná o zprávu ze strany CCM telefonu požadující ukončení spojení a uzavření portu. V případě zprávy `CloseReceiveChannel` je to tedy ukončení přijímání a v případě zprávy `StopMediaTransmission` ukončení vysílání.

  Extrahovaná data: ID páru, lokální IP adresa.

  Požadovaný stav páru: `FINISHED`.

6.7 Bezpečnost při zpracování dat a běhu aplikace

6.7.1 Parsování Skinny zpráv

Vzhledem ke stejné implementaci parsování zpráv protokolu Skinny je tento odstavec s úpravami převzat z mé předchozí práce [1].

Kvůli binárnímu formátu Skinny zpráv je potřeba při parsování dávat pozor na přetečení, tedy jestli skutečná velikost zprávy odpovídá očekávané velikosti zprávy a že nebudou čteny byty, které už nepatří do packetu, z jiné části paměti. To platí jak pro čtyřbytové číselné položky tak i pro položky obsahující řetězec s fixní velikostí. V případě novější verze protokolu, kde jsou řetězce ukončeny znakem 0 a nemají fixní velikosti, je třeba kontrolovat, jestli řetězec skončil před koncem packetu. V případě absence těchto kontrol je možné, aby někdo poslal po síti speciálně upravený packet a způsobil přetečení (buffer overflow) v programu. Běžným důsledkem je pád aplikace, v extrémním případě tato programátorská chyba může vést ke spuštění kódu v tomto upraveném packetu na zařízení, které se snaží tento packet přečíst. Jisté verze systému od společnosti Cisco trpěly touto zranitelností a v případě úspěšného útoku mohlo dojít k restartu Call Manageru [9]. Moje implementace parsování Skinny protokolu hlídá před přečtením každé položky, jestli nedošlo k přetečení, které, pokud je detekováno, způsobí vypsání chyby a ignorování packetu.

6.7.2 Práva při běhu aplikace

Hlavním úkolem aplikace je odposlouchávat živý provoz na síti. Pro připojení na síťové rozhraní jsou ale potřeba nejvyšší možná práva v systému — je tedy nutné aplikaci spustit pod uživatelem root. Toto spuštění je ale nebezpečné, protože by aplikace mohla způsobit při nějaké chybě nebo špatné konfiguraci nenávratné změny v systému. Proto je možné aplikaci nastavit tak, aby se ihned po připojení k síťovému rozhraní vzdala práv uživatele root a následně běžela s právy jiného uživatele. Jméno tohoto uživatele je nutné nastavit v konfiguračním souboru a musí v systému existovat. Díky tomuto opatření vzniká

nová vrstva ochrany a zvyšuje se úroveň zabezpečení aplikace. Stejný způsob používala i předchozí verze aplikace.

Kapitola 7

Instalace a konfigurace

7.1 Režimy spuštění

Aplikace funguje ve dvou režimech — „live“ a „offline“.

Při spuštění bez argumentů se aplikace pokusí načíst konfigurační soubor a spusit se v režimu „live“. V tomto režimu se aplikace připojí k rozhraní a detekuje a nahrává hovory probíhajících na tomto rozhraní. Druhý možný režim je „offline“, který vyžaduje zadání názvu souboru ve formátu pcap. Aplikace předpokládá, že v tomto souboru jsou uložené packety zachycené programem tcpdump a pokusí se z těchto packetů detekovat a extrahovat hovory.

V obou režimech se aplikace pokusí připojit k databázi a informace o těchto hovorech uložit do definovaných tabulek.

7.2 Požadavky a instalace

Pro instalaci a spuštění aplikace je potřeba zkompilovat zdrojové kódy. Pro kompilaci stačí spustit v adresáři se zdrojovými kódy příkaz `make`, pro instalaci pak `make install`. Úspěšná kompilace závisí na přítomnosti několika knihoven v systému:

- `libpq` [2] — Knihovna pro připojení do PostgresSQL databáze.
- `libpcap` [3] — Knihovna pro práci s packety.
- `sndfile` [4] — Knihovna pro práci se zvukovými soubory, hlavně pro uložení zvukových dat z přijatých packetů do souboru.

[2] Dostupné například zde: https://www.postgresql.org/download/
[3] Dostupné například zde: https://www.tcpdump.org/#latest-releases
[4] Dostupné například zde: http://www.mega-nerd.com/libsndfile/#Download

Bez přítomnosti těchto knihoven není možné program úspěšně zkompilovat.

Pro práci s databází je potřeba vytvořit v databázi tabulky, k tomu slouží SQL příkazy v souboru `db.sql`. Příkazy v tomto souboru je potřeba spustit nad databází, se kterou bude aplikace pracovat.

7.3 Konfigurace

Pro konfiguraci aplikace je určen soubor `confcall.conf` v adresáři spustitelného souboru. Pro požadované chování aplikace je potřeba správné nastavení, i přesto aplikace dokáže s určitými omezeními běžet s výchozím nastavením.

Je potřeba nastavit hlavně správné připojení k databázi (direktivy `dbhost`, `dbport`, `dbuser` a `dbpass`). Při špatném nastavení aplikace sice poběží, bude detekovat, nahrávat hovory a ukládat je do souborů, ale vypíše varování o nemožnosti se připojit k databázi a informace o těchto hovorech (seznam účastníků a časů jednotlivých hovorů) se ukládat nebudou.

Další důležité nastavení je zvolení uživatele a skupiny, pod kterými aplikace poběží v režimu „live“. Pro přístup k síťovému rozhraní je potřeba zpravidla nejvyšší možné oprávnění v systému. Pokud se tyto hodnoty nenastaví, nebo pokud nebude možné se na daného uživatele a skupinu přepnout, aplikace poběží dál s nejvyšším oprávněním, což může představovat bezpečnostní riziko.

Ostatní direktivy jsou volitelné.

7.3.1 Formát konfiguračního souboru

Formát konfiguračního souboru vychází z předchozího projektu, zejména z [2], kde je tento formát popsán. Zachována je struktura i většina direktiv. Každý řádek v konfiguračním souboru obsahuje jednu direktivu a její hodnotu. Prázdné řádky a řádky začínající znakem „#“ se ignorují. U ostatních řádků se čeká, že obsahují text s názvem direktivy, bílé znaky a poté hodnotu pro danou direktivu. Pokud je v souboru víckrát použita stejná direktiva, použije se hodnota té poslední.

7.3.2 Seznam konfiguračních direktiv

- `user`
  Výchozí hodnota: *není*
  Jméno existujícího uživatele v systému, na kterého se má aplikace přepnout po připojení k síťovému rozhraní v režimu „live“. Pokud není zadáno (společně s direktivoou `group`), k přepnutí na uživatele nedojde a aplikace poběží stále pod nejvyšším oprávněním (pod uživatelem root).

- `group`
  Výchozí hodnota: *není*

- `iface`
  Výchozí hodnota: návratová hodnota funkce `pcap_lookupdev()`
  Název síťového rozhraní, na kterém má aplikace odchytávat packety v režimu „live“.

- `skinny_port`
  Výchozí hodnota: `2000`
  Číslo portu TCP, na kterém se mají očekávat packety Skinny protokolu.

- `dbhost`
  Výchozí hodnota: `127.0.0.1`
  IP adresa pro připojení na server databáze PostgresSQL.

- `dbport`
  Výchozí hodnota: `5432`
  Port pro připojení na server databáze PostgresSQL.

- `dbname`
  Výchozí hodnota: `confcall`
  Název databáze po připojení k serveru PostgresSQL.

- `dbuser`
  Výchozí hodnota: `confcall`
  Jméno uživatele pro připojení do databáze PostgresSQL.

- `dbpass`
  Výchozí hodnota: `confcall`
  Heslo uživatele pro připojení do databáze PostgresSQL.

- `name`
  Výchozí hodnota: `%A_%B-%H%M%S.wav`
  Formát pro název souborů, kam se budou ukládat dekódovaná zvuková data. Obsahuje znaky formátu používané funkcí `strftime()` a tyto znaky: `A` (nahrazeno zdrojovou IP adresou), `B` (cílová IP adresa), `f` (zdrojový port) a `t` (cílový port).

- `path`
  Výchozí hodnota: `./calls/%Y-%m-%d-%H`
  Formát pro název adresáře v souborovém systému, kam se budou ukládat soubory s dekódovanými zvukovými daty. Formát odpovídá formátu používanému funkcí `strftime()`.

- `len`
  Výchozí hodnota: `0`
  Maximální délka zvukového proudu v sekundách. Po dosažení tohoto limitu se zvukový proud považuje za ukončený. Hodnota 0 znamená, že délka hovoru nebude kontrolovaná.

- `logto`
  Výchozí hodnota: `stdout`
  Tato hodnota určuje, kam se budou zapisovat logovací zprávy. Hodnota `stdout` určuje standardní výstup, `stderr` standardní chybový výstup, `syslog` systémový log. Jiná hodnota určuje název souboru, do kterého se má zapisovat.

- `logfacl`
  Výchozí hodnota: `local0`
  Facilita pro systémový log, možné hodnoty jsou `local0`, `local1`, `local2`, `local3`, `local4`, `local5`, `local6` nebo `local7`.

- `loglvl`
  Výchozí hodnota: `info`
  Úroveň logovacích výstupů — čím vyšší úroveň, tím více textů se při běhu bude vypisovat. Možné hodnoty jsou (v pořadí od nejpodrobnějších po nejmíň podrobné): `debug`, `info`, `warning`, `error`.

- `logcolors`
  Výchozí hodnota: `1`, pokud je `logto` nastaven na `stdout` nebo `stderr`. V opačném případě `0`.
  Hodnota určující, jestli se pro výpis různých úrovní mají použít barvy pomocí ASCII sekvencí.

7.4 Možnosti spuštění

Chování aplikace lze ovlivnit těmito přepínači:

- `-c` — Cesta ke konfiguračnímu souboru, který se použije místo výchozího `confcall.conf`.

- `-h` — Program pouze vypíše možné argumenty a skončí.

- `-n` — Provede se pouze kontrola syntaxe konfiguračního souboru a program skončí.

- `-p` — Na standardní výstup se vypíšou direktivy a jejich hodnoty načtené z konfiguračního souboru, se kterým by aplikace běžela, a program se ukončí.

Kapitola 8

Testování

Pro účely testování byly vytvořeny soubory se zachycenými packety ve formátu pcap pro typické scénáře hovorů, které by mohly v reálném provozu nastat. Součástí těchto scénářů jsou různé situace během konferenčních hovorů, různé kombinace pořadí připojovaných a odpojovaných zařízení a podržených hovorů. Tyto soubory jsou součástí repozitáře se zdrojovými kódy. Soubory byly vytvořeny nástrojem tcpdump, jedná se tedy o standardní formát pro uložení zachycených packetů a je možné je otevřít a analyzovat například programem Wireshark. Nad každým souborem lze samozřejmě spustit popisovanou aplikaci, v tom případě dojde k extrakci informací o všech hovorech v daném souboru a dekódování a uložení zvukových dat do databáze a souborů. Stejným způsobem byla testována i původní implementace projektu.

Seznam souborů a popis scénářů, které obsahují:

- `call-9887-8783.pcap` Jednoduchý hovor mezi dvěma účastníky. Pouze vytočení čísla, vytvoření hovoru a zavěšení. Extrakcí by měly vzniknout dva páry, pro každý směr jeden. Kromě těchto párů by měly vzniknout i dva proudy — pro první pár bude jeden odchozí a druhý příchozí, pro druhý pár to bude obráceně.

- `call-9887-8783-mute.pcap` Stejný jednoduchý hovor jako v předchozím případě, pouze uprostřed hovoru je použita funkce MUTE na telefonu — pro ztlumení odchozího zvuku. Jak se později zjistilo, na signalizační data a zvukové proudy tato funkce nemá vliv.

- `call-9887-8783-hold.pcap` Stejný jednoduchý hovor jako v prvním případě, uprostřed hovoru je použita funkce HOLD — podržet hovor. Tím ze dvou původních proudů vznikne celkem proudů pět — dva před podržením hovoru, jeden během podržení hovoru (hudba posílaná z CCM na zařízení) a dva po podržení hovoru.

- `call-9887-8783-8782.pcap` Jednoduchý konferenční hovor, po spojení dvou telefonů se pozve třetí do konferenčího hovoru. Ten následně konferenci opouští, původní dva pak ještě chvíli udržují hovor.
- `call-9887-8783-8782-abandon.pcap` Po spojení dvou telefonů se do konference přidává třetí účastník. Ten, který pozval třetího účastníka, zavěšuje, čímž ukončuje konferenční hovor i pro ostatní.
- `call-9887-8783-8782-referer.pcap` Stejný případ jako předchozí, ale třetího účastníka zve do konference volaný, nikoli volající. Konferenční hovor následně ukončuje původní volaný zavěšením.
- `call-9887-8783-8782-last.pcap` Třetího účastníka zve volaný, konferenci potom opouští tento třetí účastník. Obnovuje se původní hovor mezi původními dvěma účastníky.
- `call-9887-8783-8782-fourth.pcap` Tento záznam obsahuje odmítnutý pokus o přizvání čtvrtého účastníka do konference. K odmítnutí došlo ze strany CCM. Konferenční hovor zůstává pouze se třemi účastníky.
- `call-9887-8783-8785.pcap` Konferenční hovor se třemi účastníky, kde komunikace mezi CCM a třetím účastníkem není k dispozici.
- `call-8785-9887-stary.pcap` Podobná situace, jako předchozí případ. Rozdíl je v tom, že třetí účastník je mobilní telefon úplně mimo síť, ve které se nachází ostatní zařízení včetně CCM.
- `call-two-confs.pcap` Dva konferenční hovory v jeden časový okamžik.

8.1 Problémy a známá omezení

8.1.1 Jenom jedna vrstva VLAN

Kvůli omezení použitého packetového filtru (BPF) není možné jednoduchým způsobem vytvořit filtr pracující se vnořenými (více než jedna vrstva) VLAN. Každá další vrstva VLAN posouvá totiž binární data v packetu o počet bytů určujících, že se jedná o VLAN a identifikaci této VLAN. Implementace BPF vyžaduje totiž definovat pro každé VLAN zanoření filtr zvlášť. Například pro filtr zdrojové adresy `src host 192.169.0.1` je pro podporu VLAN nutné tento filtr zdvojit a podruhé použít s klíčovým slovem `vlan`, výsledkem je tedy tento filtr:

```
(src host 192.169.0.1) or (vlan and (src host 192.169.0.1))
```

Pro podporu zanořené VLAN je nutné tento zápis:

```
(src host 192.169.0.1) or (vlan and (src host 192.169.0.1)
or (vlan and vlan and (src host 192.169.0.1))
```

V zájmu jednodušší implementace jsem zvolil kompromis a aktuální verze podporuje (stejně jako původní projekt) pouze žádnou nebo jednu vrstvu VLAN.

8.1.2 Pouze dvě známé verze Skinny protokolu

Tento problém zůstal z původního projektu. Uvnitř zpráv Skinny protokolu je informace o použité verzi protokolu. Bohužel se mi ani pro tuto práci nepodařilo získat testovací data pro jinou verzi, než 0 nebo 22. Nelze tedy zaručit funkčnost programu i pro jiné verze.

8.1.3 Chybějící grafické rozhraní a mixování

Součástí původního projektu bylo i grafické rozhraní vytvořené jako webová aplikace v jazyce PHP. Pro potřeby podpory konferenčních hovorů a nového datového modelu bylo ale nutné změnit doménový model databáze, na kterém byla tato aplikace závislá. Pro efektivní použití programu by bylo potřeba upravit nebo přepracovat tuto webovou aplikaci tak, aby byla schopná s novým databázovým modelem pracovat. Pro potřeby zpětného přehrávání zachycených a dekódovaných zvukových souborů je potřeba také navrhnout a implementovat mixování zvuku a to tak, aby bylo možné pohodlně a uživatelsky přívětivě přehrávat zachycená zvuková data.

8.1.4 Zvukové formáty

V SCCP protokolu je podpora pro různé zvukové formáty. Například podle zdrojového kódu Wiresharku ([6]) jich může být několik desítek. Aktuální implementace vybírá pouze mezi několika typy formátů. Samotné parsování zvukových dat nechává na knihovně libsndfile. Nejsou tedy podporovány všechny možné formáty, takže je možné, že v některých případech odmítne aplikace hovor nahrát z důvodu neznámého formátu. V testovacím prostředí s tím ale problém nebyl.

Závěr

Výsledkem této práce je rozšíření, zobecnění a s tím spojené přepracování předchozího projektu pro zachytávání a nahrávání VoIP hovorů v protokolech SIP a Skinny. Nová funkcionalita je podpora konferenčních hovorů, díky tomu bylo nutné vytvořit nový návrh architektury a datového modelu, který není kompatibilní s původním. Proto součástí práce není podpora protokolu SIP. Pro vytvoření nového datového modelu bylo nutné pochopit, jakým způsobem zařízení a CCM signalizují vznik a změnu stavu konferenčního hovoru. Díky proprietárnímu charakteru Skinny protokolu jsem většinu potřebných informací získal analýzou zachycených packetů (pomocí programu Wireshark). Takto získané informace sloužily jako podklad pro návrh nové architektury, která je popisovaná v této práci.

Možná rozšíření

Možná rozšíření se shodují (kromě podpory konferenčních hovorů) s těmi, která jsou popsána v mé bakalářské práci ([1]).

Mezi ně patří například podpora video hovorů — Skinny protokol umožňuje přenášet kromě zvukových dat i video data. Lze předpokládat, že rozdíl je pak vidět v tom, na jakém formátu dat se zařízení mezi sebou v průběhu vytváření datového spojení shodnou. Mezi zprávy Skinny protokolu patří i tyto: *StartMultiMediaTransmission*, *StopMultiMediaTransmission* a *StartMultiMediaTransmissionAck*. Jedná se nejspíš o alternativy ke zprávám *StartMediaTransmission*, *StopMediaTransmission* a *StartMediaTransmissionAck*. Pro podporu video hovorů by mělo nejspíš stačit umět extrahovat informace z těchto zpráv a potom dekódovat samotné datové proudy obsahující video.

Stejně jako v původním projektu, zůstává problém s prací se šifrovanou komunikací. Standardně jsou totiž všechny signalizační zprávy Skinny hovoru šifrované pomocí protokolu TLS na portu 2443. Pro testování této a předchozí práce bylo potřeba šifrování na straně CCM dočasně pro testované telefony

vypnout. Možné rozšíření tedy spočívá v umožnění detekovat a nahrávat i tyto šifrované hovory bez potřeby šifrování explicitně vypínat.

Literatura

[1] Polák, M.: *Signalizace a nahrávání VOIP hovorů*. Bakalářská práce, České vysoké učení technické, 2016.

[2] Šuster, F.: *Extrakce VOIP dat ze síťového provozu*. Bakalářská práce, České vysoké učení technické, 2015.

[3] Robejšek, V.: *Webové rozhraní k VOIP hovorům*. Bakalářská práce, České vysoké učení technické, 2015.

[4] Kučera, J.: *Extrakce a zpracování hlasu z VOIP paketů*. Bakalářská práce, České vysoké učení technické, 2015.

[5] Hartpence, B.: *Packet Guide to Voice over IP: A system administrator's guide to VoIP technologies*. O'Reilly Media, 2013, ISBN 978-1-449-33967-8.

[6] de Groot, D.: Dissector for the Skinny Client Control Protocol. Online, [vid. 2019-02-07]. Dostupné z: `https://code.wireshark.org/review/gitweb?p=wireshark.git;a=blob;f=epan/dissectors/packet-skinny.c;h=28db7a1f733d4b82896dbeac031794aafe600f3b;hb=HEAD`

[7] Git. Online, [vid. 2019-02-07]. Dostupné z: `https://git-scm.com/`

[8] *Network Protocols Handbook*. Javvin Technologies, 2005, ISBN 0974094528.

[9] Porter, T.; Jr., J. K.; Baskin, B.: *Practical VoIP Security*. Syngress, 2006, ISBN 1-59749-060-1.

[10] Cisco: Cisco ATA 186 and Cisco ATA 188 Analog Telephone Adaptor Administrator's Guide (SCCP). 2013, online, [vid. 2018-01-25]. Dostupné z: `http://www.cisco.com/c/en/us/td/docs/voice_ip_comm/cata/186_188/2_15_ms/english/administration/guide/sccp/sccp.pdf`

[11] VoIP Lecture Notes. Online, [vid. 2019-01-25]. Dostupné z: http://www.technologeeks.com/Courses/VoIP.pdf

[12] [obrázek] Skinny call flow diagram. Online, [vid. 2019-01-25]. Dostupné z: http://www.gl.com/skinny-protocol-emulation-using-maps.html

Příloha A

Seznam použitých zkratek

ASCII American Standard Code for Information Interchange

BPF Berkeley Packet Filter

CCM Cisco Call Manager

DHCP Dynamic Host Configuration Protocol

DNS Domain Name System

GSM Global System for Mobile Communications

CSS Cascade Style Sheets

HTML HyperText Markup Language

HTTP Hypertext Transport Protocol

IP Internet Protocol

LAN Local Area Network

VLAN Virtual Local Area Network

NTP Network Time Protocol

PoE Power over Ethernet

RTP Real-time Transport Protocol

SCCP Skinny Client Control Protocol

SIP Session Initiation Protocol

SQL Structured Query Language

TCP Transmission Control Protocol

TFTP Trivial File Transport Protocol

TLS Transport Layer Security

UDP User Datagram Protocol

VoIP Voice over Internet Protocol

Příloha **B**

Obsah přiložené SD karty

- `readme.txt` ..... stručný popis obsahu SD karty
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `db.sql` ..... SQL skript pro vytvoření tabulek v databázi
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
- `test` ..... Testovací pcap soubory