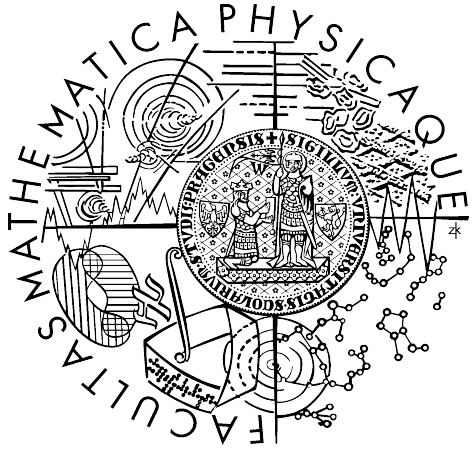
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Daniel Kratochvíl

H.323 Gatekeeper

Středisko informatické sítě a laboratoří

Vedoucí bakalářské práce: RNDr. Libor Forst

Studijní program: Informatika, Programování

2007

Děkuji svému vedoucímu RNDr. Liboru Forstovi za jeho pomoc a čas, který mi věnoval během vývoje bakalářské práce.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 7.8.2007 Daniel Kratochvíl

Obsah

Název práce: H.323 Gatekeeper
Autor: Daniel Kratochvíl
Katedra (ústav): Středisko informatické sítě a laboratoří
Vedoucí bakalářské práce: RNDr. Libor Forst
e-mail vedoucího: Libor.Forst@mff.cuni.cz

Abstrakt: Cílem práce je návrh a implementace služby Gatekeeper (serveru pro H.323 RAS protokol) s integrovanou H.323 proxy. Součástí práce je rovněž testovací aplikace protokolu H.323 RAS který může vystupovat jak jako server, tak klient.
Aplikace je naprogramována v jazyce C++ s použitím knihoven OpenH323, PWlib a readline a je primárně určena pro běh jako démon pod operační systém Linux. Díky striktnímu využívání přenositelných knihoven, je ale aplikaci možné provozovat na řadě jiných operačních systémů. Testovány byly Linux a FreeBSD, ale vhodné by měl být i jiné Unixy, Windows nebo MacOS X.

Klíčová slova: VoIP, H.323, server, Unix

Title: H.323 Gatekeeper
Author: Daniel Kratochvíl
Department: Network and Labs Management Center
Supervisor: RNDr. Libor Forst
Supervisor's e-mail address: Libor.Forst@mff.cuni.cz

Abstract: The purpose of this work is to design and implement a Gatekeeper service (a server for H.323 RAS protocol) with an integrated H.323 proxy. Thesis includes also a H.323 RAS protocol tester that can act both as a server and a client.
The application is implemented in the C++ language using OpenH323, PWlib and readline libraries and is designed to run as a daemon on the Linux operating system. Strict usage of the platform independent libraries provides the ability to run on other operating systems. Linux and FreeBSD were successfully tested but another Unices, Windows or MacOS X should be suitable as well.

Keywords: VoIP, H.323, server, Unix

Kapitola 1 - Cíl práce

Inspirací pro tuto práci byl veliký rozmach internetové telefonie v posledních letech a problematičnost využití současných open source řešení za překladem adres (NAT). Cílem bakalářské práce je nejen implementovat server pro H.323 RAS protokol (gatekeeper), který umožní uživatelům v malých sítích za NATem používat VoIP komunikaci pomocí protokolu H.323, ale také vytvořit aplikaci pro testování funkčnosti takového serveru.

Základním požadavkem je vytvořit serverovou aplikaci, která bude poskytovat základní funkce Gatekeeperu pro H.323 a bude umět vystupovat jako H.323 proxy. Pokud bude aplikace provozována na stejném stroji, který poskytuje NAT, bude umožněna snadná hlasová komunikace mezi uživateli ve vnitřní síti a v Internetu.

Aplikace bude navíc obsahovat HTTP rozhraní, které poskytne možnost konfigurace i když uživatel nebude mít přístup do operačního systému stroje, kde server běží. Bude zde i možnost zjistit, kteří uživatelé jsou nyní online, a tím poskytne informaci, komu se lze případně dovolat.

Druhou částí aplikace bude H.323 RAS tester, který přes readline rozhraní umožní testovat funkčnost komunikace pomocí protokolu RAS (Registration, Admission, Status). Umí vystupovat jako gatekeeper i jako endpoint a je uzpůsoben tomu, aby byl řízen pomocí skriptů.

1.1 Představení H.323

H.323 je standard od ITU-T (International Telecommunication Union - Telecommunication Standardization Sector), který definuje protokoly pro multimediální komunikaci po packet-switched sítích. Umožňují real-timeové audio, video a data přenosy. H. 323 bylo velice brzy dostupné a zároveň bylo prvním standardem, který používal RTP (Real-time Transport Protocol), což patřilo mezi jeho nesporné výhody. Díky tomu se rychle stal nejrozšířenějším standardem pro hlasovou a video komunikaci a tuto pozici si drží dodnes i přes stále rostoucí konkurenci protokolu SIP. H.323 je ovlivněno jak standardy pro veřejnou telefonní síť tak pro IP sítě, což umožňuje hladkou integraci s telefonní sítí i přenos dat po

Internetu.

H.323 definuje několik prvků, které jsou třeba pro multimediální komunikaci z nichž některé jsou povinné a některé volitelné. Mezi ty nejdůležitější patří: Terminál, Gateway, Multipoint Control Unit a Gatekeeper.

- **Terminál** je koncové zařízení (endpoint) v síti, které poskytuje realtimeovou obousměrnou komunikaci s jiným H.323 zařízením. Terminály mohou nabízet hlasové, datové nebo i video služby a většinou se jedné o VoIP telefony nebo softwarové telefony (např. Ohphone, Ekiga, Yast, ...).
- **Gateway** poskytuje konverzi protokolů mezi H.323 terminály a jinými terminály, které H.323 nepodporují. Gateway může na příklad spojovat H.323 síť s veřejnou telefonní sítí a tím umožnit hovory mezi softwarovými klienty a běžnými telefony.
- **Multipoint Control Unit (MCU)** je prvek, který umožňuje třem a více terminálům vytvořit konferenci. MCU obsahuje Multipoint Controler pro práci s call signaling a nepovinně může obsahovat Multipoint Processor pro přenos multimediálních dat.
- **Gatekeeper** je centrálním bodem celé H.323 sítě. Představuje sice její nepovinnou součást, ale poskytuje významné služby jako překlad aliasů na adresy endpointů nebo povolování (nebo zakazování) jednotlivých hovorů. Pokud je v síti přítomen, je povinností všech endpointů využívat jeho služeb.

H.323 spojuje řadu protokolů. Zde uvádím jen ty nejdůležitější z nich:

- **H.225.0 RAS** (Registration, Admission, Status) zajišťuje komunikaci endpointů s gatekeeperem (pokud je v síti přítomen). Jedná se o UDP pakety, které jsou zcela nezávislé na RTP i na call signaling. Standardně používané porty jsou 1719 pro RAS komunikaci a 1718 pro multicastové vyhledávání gatekeeperu. Existují tyto RAS zprávy:
  - **GRQ** (Gatekeeper Request)**, GCF** (Gatekeeper Confirm) a **GRJ** (Gatekeeper Reject) slouží pro vyhledávání gatekeeperu. Pokud endpoint zná adresu gatekeeperu, pošle mu unicastem GRQ; pokud ji nezná, pošle GRQ multicastem. Gatekeeper vždy odpovídá unicastem.
  - **RRQ** (Registration Request), **RCF** (Registration Confirm) a **RRJ**

(Registration Reject) slouží pro registraci. Endpoint odesílá RRQ obvykle ihned poté, co obdržel GCF.

- **URQ** (Unregistration Request), **UCF** (Unregistration Confirm) a **URJ** (Unregistration Reject) slouží pro odregistrovávání. Po odeslání URQ je endpoint vždy odregistrovaný. URJ informuje endpoint o tom, že před odesláním URQ zaregistrován nebyl.
- **ARQ** (Admission Request), **ACF** (Admission Confirm) a **ARJ** (Admission Reject) jsou zasílány, pokud chce endpoint začít hovor. Jestliže gatekeeper hovor povolí, tak v ACF sdělí cílovou adresu, na kterou má endpoint iniciovat call signaling. To umožňuje existenci proxy módu, kdy místo adresy volaného sdělí gatekeeper volajícímu adresu svojí.
- Pomocí **IRQ** (Information Request) zjišťuje gatekeeper status endpointu a pomocí **IRR** (Information Request Response) endpoint odpovídá. **IACK**(Information Request Acknowledge) a **INACK**(Information Request Negative Acknowledge) jsou potvrzením IRR od gatekeeperu.
- **BRQ** (Bandwidth Request), **BCF** (Bandwidth Confirm) a **BRJ** (Bandwidth Reject) slouží k žádostem o navýšení nebo snížení šířky pásma (bandwidth) přidělené danému hovoru.
- **LRQ** (Location Request), **LCF** (Location Confirm) a **LRJ** (Location Reject) se používají k dotazům na polohu gatekeeperu nebo endpointů a jsou využívány hlavně při inicializaci hovorů, kterých se účastní více než jeden gatekeeper. Pokud gatekeeper zná polohu dotazovaného endpointu a tento endpoint má dostatek zdrojů, sdělí všechny jeho známé adresy v LCF.
- **RAI** (Resource Availability Indicator) využívá gateway k informování gatekeeperu, zda je schopná pojmout ještě další hovory, a **RAC** (Resource Availability Confirm) je potvrzením této zprávy od gatekeeperu.

- **H.225.0 Call signaling**, jehož součástí je i Q.931, nastupuje na řadu, když je ukončena komunikace s gatekeeperem a slouží k nastavení a ukončení hovoru. Komunikace probíhá pomocí jednoho TCP spojení a standardně používaný port je 1720.
- **H.245 Media control**: Když se terminály dohodnou, že spolu chtějí komunikovat, tak se ještě potřebují shodnout na formátech a kodecích, které oba podporují. Právě k tomu slouží H.245.

Jak již bylo zmíněno, H.323 není jediný protokol využívaný pro VoIP. Vyvíjen a intenzivně podporován je ještě SIP (Session Initiation Protocol) za který je zodpovědný IETF (Internet Engineering Task Force). SIP zprávy jsou textové a jsou formátem podobné již zavedeným Internetovým protokolům (jako například HTTP). Tím pádem je jeho softwarová implementace snadnější a je vhodnější pro Internetové aplikace. H.323 má lepší spolupráci s veřejnou telefonní sítí a stále má podporu velkých společností, takže není zřejmé, jestli se některý z protokolů stane dominantním.

1.2 Obdobné produkty

1.2.1 OpenH323 Gatekeeper - The Gnu Gatekeeper

The GNU Gatekeeper představuje v dnešní době jediný stále vyvíjený open source gatekeeper, který je využíván ve většině VoIP řešení vycházejících z H.323. Je založen na knihovnách OpenH323 a PWlib, je funkční na všech platformách, kde lze tyto knihovny zkompilovat (Linux, Windows, MacOS X, Solaris, FreeBSD, OpenBSD a pravděpodobně i dalších), a má implementovánu řadu funkcí.

Veškerá konfigurace se provádí editací souboru /etc/gatekeeper.ini a získávání informací z běžícího gatekeeperu se uskutečňuje telnet připojením. Podporuje tři módy routování hovorů:

- žádné ... gatekeeper se hovoru vůbec neúčastní
- routované Call signaling ... RTP stále jde přímo mezi endpointy. Vhodné pro větší kontrolu nad hovorem. Například při zpoplatňování hovoru
- proxy ... Call signaling i RTP jdou přes gatekeeper

Za nevýhodu tohoto projektu lze považovat, že poměrně málo využívá podpory, kterou mu knihovny nabízejí. Například z OpenH323 téměř nevyužívá částečné implementace gatekeeperu, která již existuje.

1.2.2 OpenGK

Jedná se o jednoduchý gatekeeper, který je součástí knihovny OpenH323 a který přestavuje pouze příklad, jak v této knihovně psát. Projekt se již několik let nevyvíjí, jsou v něm prováděny jen drobné změny, aby byl stále zkompilovatelný i s nejnovějšími verzemi knihoven.

Hlavní výhodou je maximální využití knihoven OpenH323 a PWlib a za šťastné řešení lze považovat i HTTP konfiguraci a výpis statusu zaregistrovaných endpointů.

Bohužel značná část funkcí, které na konfigurační stránce nabízí, zatím není implementována (a zdá se, že ani nikdy nebude).

1.2.3 OpenGatekeeper

Projekt jehož kód téměř celý vznikl v roce 2000, je již od roku 2001 definitivně mrtvý. Zůstalo pouze u alfa verze. Byl také založen na knihovně OpenH323, ale se současnou cvs ani stabilní verzí knihovny již nejde zkompilovat.

1.2.4 Shrnutí

První dvě zmiňované aplikace ukazují cesty, kterými se lze při vývoji gatekeeperu ubírat. The GNU Gatekeeper je komplexní aplikací která nabízí téměř vše, co lze po gatekeeperu požadovat. OpenGK naznačuje cestu, jak vytvořit gatekeeper snadno konfigurovatelný a jak v maximální míře vyžívat možností, které knihovna OpenH323 nabízí.

Bakalářská práce je zaměřena na jejich nejdůležitější vlastnosti a ty dále rozpracovává, aby bylo umožněno snadné nasazení a údržba serveru pro správce malých domácích sítí a aby v budoucnu byla aplikace snadno rozšiřitelná o další funkce, pro které knihovna nabízí podporu.

Vzhledem k rozsahu zvoleného tématu a zaměření na domácí uživatele bude menší důraz kladen na některé rozšiřující funkce (například komunikace s dalšími gatekeepery nebo podpora pro zpoplatňování hovorů). Tyto možnosti jsou

samozřejmě důležité, ale díky snaze o co největší přizpůsobení se struktuře použitých knihoven, bude následné rozšiřování aplikace o další H.323 funkce bez větších překážek možné.

1.2.5 Tester

I při práci na testeru jsem se pokoušel hledat inspiraci u již existujících produktů, ale bohužel jsem nenalezl žádné open source řešení, které by umožňovalo H.323 RAS protokol testovat. Některé komerční projekty na svých internetových stránkách sice tvrdí, že tuto problematiku pokrývají, ale jejich volné stažení není možné a to ani formou demoverze nebo shareware.

Kapitola 2 - Návrh aplikace

Před samotnou implementací je třeba rozvrhnout funkcionalitu aplikace. Tuto fázi vývoje je třeba nepodcenit. Zde přijatá rozhodnutí ve velké míře ovlivňují výslednou podobu aplikace a veškeré nejasnosti nebo chyby v návrhu mohou vést k následnému předělávání částí aplikace.

2.1 Využití knihoven

Prvním a zároveň nejdůležitějším rozhodnutím byla míra využití knihoven. Samozřejmě existovala teoretická možnost knihovny nepoužívat vůbec, ale H.323 je rozsáhlý protokol (a to i když se omezíme pouze na tu část, kterou potřebuje gatekeeper), takže by jen jeho samotná implementace překračovala rozsah, který je od bakalářské práce očekáván. Tuto možnost jsem tedy zavrhl.

Jediná open source knihovna, která poskytuje H.323, je OpenH323, šlo tedy uvažovat pouze o ní. Jedná se o knihovnu napsanou v C++ a do značné míry založenou na dědičnosti. Aby byla OpenH323 platformově nezávislá, využívá služeb knihovny PWlib.

PWlib je poměrně velká knihovna, která vznikla před řadou let s cílem umožnit přenositelnost grafických aplikací mezi Microsoft Windows a X-Windows systémem. Od tohoto cíle bylo ale časem upuštěno a s ohledem na hlavního uživatele (projekt OpenH323) se knihovna zaměřila na podporu síťových aplikací a vláken. K těmto základním funkcím jsou přibaleny přídavky jako parser konfiguračního souboru, containery nebo jednoduchý web server.

Rozhodl jsem se pro co největší využívání knihoven OpenH323 a PWlib. Je to „nejčistší“ řešení a umožňuje později relativně snadno doplňovat gatekeeper i tester o další funkce, které v knihovně jsou nebo které do ní budou časem přidány. Navíc mi to umožnilo vyzkoušet si práci s velkou a neustále vyvíjenou knihovnou, což považuji za cennou zkušenost. Protože jsou knihovny OpenH323 a PWlib psány v jazyce C++, tak tento jazyk je použit i v celé bakalářské práci.

Cenou za tuto volbu je nutnost podrobného nastudování částí knihovny, která je bohužel často nedostatečně zdokumentovaná a její chování se poměrně často odlišuje od výsledku očekávaného po přečtení dokumentace. Někdy nezbývala jiná možnost než řadu hodin číst zdrojové kódy, debugovat knihovní funkce a sniffovat pakety, které aplikace přijímala a vysílala, jen abych mohl do programu dopsat řádek kódu.

Alternativou bylo z knihovny použít jen tu nejnutnější část pro H.323 komunikaci a složitější třídy nad tím si navrhnout a implementovat sám. Tato varianta (kterou si například zvolil The GNU Gatekeeper) umožňuje větší kontrolu nad všemi funkcemi a tím pádem může při menším úsilí nabízet i funkce, na které knihovna nepamatuje. Následně se za to ale platí větším množstvím napsaného kódu (větší pravděpodobnost chyby) a ztrátou možnosti snadno využívat updaty a nově přidané vlastnosti knihovny.

2.2 Vztah gatekeeperu a testeru

V rámci bakalářské práce jsou vyvíjeny dvě aplikace: Gatekeeper a Tester, který navíc obsahuje dvě od sebe oddělené části: serverovou a klientskou. Bylo třeba definovat jejich vzájemný vztah. Naskytovaly se dvě možnosti:

1) Rozdělit projekt na dvě části: gatekeeper a tester
2) Rozdělit projekt na tři části: gatekeeper, tester a knihovnu, která by pokrývala společnou funkcionalitu

Z počátku bylo obtížné určit, kolik společné funkcionality bude gatekeeper a serverová část testeru moci mít. Rozhodl jsem se pro vytvoření knihovny a implementaci gatekeeperu pomocí ní. Hlavní výhody této varianty lze nalézt v zamezení opakování společného kódu jeho umístěním do knihovny a v úspoře paměti stroje při současném běhu gatekeeperu a testeru. Při testech jádra tohoto gatekeeperu se ale začalo ukazovat, že by bylo vhodné upravit některé aspekty jeho chování. Tyto změny ale znamenaly značné odchýlení od funkčnosti, kterou knihovna s ohledem na potřeby testeru nabízela. Po zvážení situace jsem tedy myšlenku knihovny opustil a přešel k variantě dvou oddělených aplikací. Původní obavy z opakování kódu se nakonec ukázali jako liché. Skutečně správná volba měla být již od začátku dvě samostatné aplikace.

2.3 NAT

Dále se bylo třeba rozhodnout, jakým způsobem udělat podporu pro klienty za překladem adres. Možnosti se nabízely tři: GnuGk NAT traversal, standardy H.460.17 až H.460.19 nebo H.323 proxy.

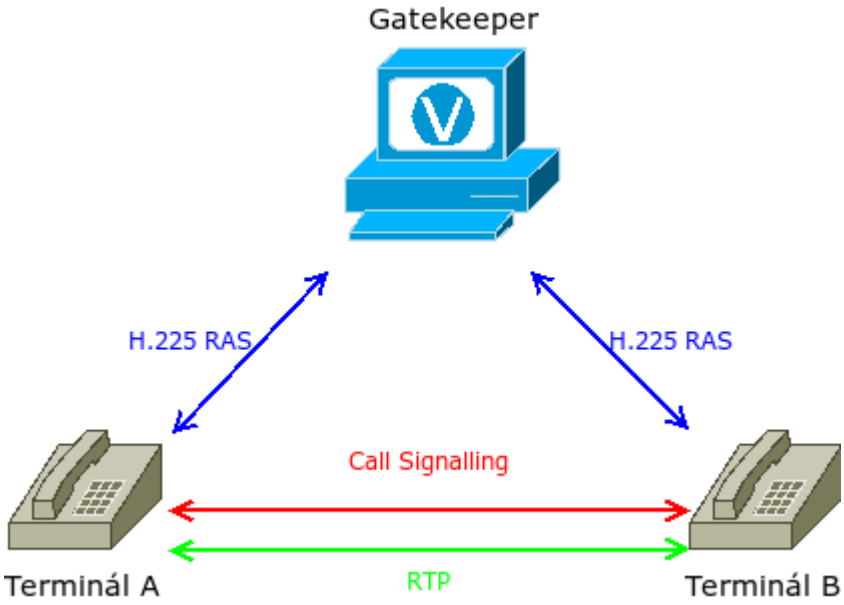
GnuGk NAT Traversal [1] je metoda, kterou zavedl The GNU Gatekeeper a která je podporovaná jen minimálním počtem klientů. Spočívá v tom, že klient otevře ke gatekeeperu nové TCP spojení, po kterém ale neposílá žádná data. V okamžik, kdy gatekeeper chce klienta kontaktovat přes call signaling, využije právě toto spojení. V tomto případě klient okamžitě vytvoří nové TCP spojení pro budoucí použití. Metoda bohužel pokrývá pouze call signaling, takže pokud volající k volanému nemůže vytvořit RTP spojení, pak hovor není možné uskutečnit. Pouze tato metoda tedy není pro klienty za NATem příliš prospěšná. Pro nedostatečnou funkčnost, nestandardnost a malou podporu jsem tuto možnost zavrhl.

V průběhu vývoje bakalářské práce (v roce 2006) rozšířilo ITU (International Telecommunication Union) H.323 o standardy H.460.17, H.460.18 a H.460.19 [2][3], které zavádějí tři metody pro NAT Traversal a které pracují jak s call signaling, tak s RTP. Standardy vypadají velice dobře a časem jistě půjde o preferované řešení. Bohužel ale nejsou zpětně kompatibilní a jejich komerční podpora je nyní (necelý rok po jejich vydání) minimální a open source podpora nulová.

Jediná plně funkční metoda je integrovat do gatekeeperu H.323 proxy. Její největší výhodou je, že nevyžaduje podporu ze strany klientů, takže i v dnešní době může být v plné míře používána, což také vedlo k jejímu výběru. Hlavní omezení představuje nutnost spustit gatekeeper na stroji, který poskytuje NAT, nebo na nějakém obdobném (musí být schopen přijímat spojení z obou sítí i do nich spojení vytvářet).

2.3.1 Běžný hovor za přítomnosti gatekeeperu

V případě, že je v síti přítomen gatekeeper a terminály jsou k němu zaregistrovány, vypadá hovor podle standardu H. 323 následovně (Obrázek 1). Přesně takto se chová i NGK (Nily's Gatekeeper), pokud je vypnutý proxy mód.

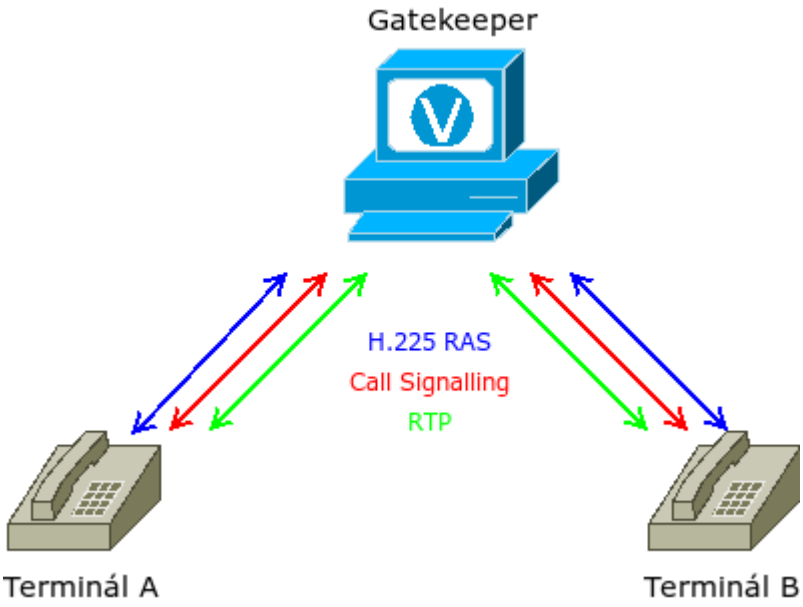


Obrázek 1 - Běžný hovor

1) Terminál A pošle pošle ARQ gatekeeperu
2) Pokud smí telefonát uskutečnit, gatekeeper odpoví ACF
3) Terminál A pošle Q.931 Call-Setup terminálu B
4) Terminál B pošle ARQ gatekeeperu
5) Gatekeeper odpoví ACF
6) Když terminál B přijímá hovor, tak pošle Q.931 Connect Terminálu A
2) Oba terminály pravidelně posílají IRR gatekeeperu
7) Během hovoru gatekeeper nezasahuje. Multimediální data jsou přenášena přímo mezi terminály po RTP spojeních
8) Pokud chce terminál A zavěsit, pošle DRQ gatekeeperu
9) Gatekeeper odpoví DCF
10) Terminál A pošle Q.931 Release Complete terminálu B
11) Terminál B pošle DRQ gatekeeperu
12) Gatekeeper odpoví DCF

2.3.2 Proxy mód

V krocích 3) a 8) běžného hovoru (kapitola 2.3.1) navazují terminály spojení přímo mezi sebou, což při NATu nemusí být povoleno. Toto lze vyřešit přepnutím gatekeeperu do Proxy módu. Celý hovor poté probíhá podle Obrázku 2:



Obrázek 2 - Proxy mód

1) Terminál A pošle pošle ARQ gatekeeperu
3) Gatekeeper odpoví ACF, ale jako cílovou IP adresu uvede adresu vlastní
4) Terminál A pošle Q.931 Call-Setup gatekeeperu
5) Gatekeeper pošle Q.931 Call-Setup terminálu B
6) Terminál B pošle ARQ gatekeeperu
7) Gatekeeper odpoví ACF
8) Když terminál B přijímá hovor, pošle Q.931 Connect Gatekeeperu
9) Gatekeeper pošle terminálu A tentýž typ zprávy, který dostal od terminálu B.
10) Odpověď, kterou gatekeeper dostane od terminálu B pošle terminálu A
11) Oba terminály pravidelně posílají IRR gatekeeperu
12) Oba terminály otevřou dva RTP kanály ke gatekeeperu (příchozí a odchozí audio). Gatekeeper přeposílá všechna data z výstupního kanálu jednoho terminálu na vstupní kanál druhého terminálu
13) Když chce terminál A zavěsit, pošle DRQ gatekeeperu
14) Gatekeeper odpoví DCF
15) Terminál A pošle Q.931 Release Complete gatekeeperu
16) Gatekeeper pošle terminálu B Q.931 Release Complete terminálu A
17) Terminál B pošle DRQ gatekeeperu
18) Gatekeeper odpoví DCF

Nutnou a postačující podmínkou k funkčnosti tohoto řešení je schopnost gatekeeperu otevírat TCP spojení a posílat UDP pakety k oběma terminálům a aby oba terminály měly stejná práva směrem ke gatekeeperu. To je v případě NATu splněno, pokud gatekeeper běží na stejném stroji, který poskytuje NAT.

Proxy ale nemusí být užitečná pouze když se jedná o NAT. Vhodná může být i v případě, že je gatekeeper připojen do dvou sítí, mezi kterými neexistuje routing. Například bylo úspěšně vyzkoušeno spojení mezi IPv6 a IPv4 sítí.

2.4 Správa gatekeeperu

Další důležitou věcí, kterou bylo třeba rozhodnout je konfigurace a správa gatekeeperu. Jako každý daemon na unixu je i NGK konfigurovatelný pomocí plain textového souboru. Editace konfiguračního souboru ale nemusí být vždy dostatečná. Správce může potřebovat zjistit vzdáleně současný stav aplikace nebo pozměnit její konfiguraci. Řešením tohoto problému by mohla být možnost telnet připojení k aplikaci nebo poskytnutí HTTP rozhraní. Pro splnění snahy o co největší využití již přilinkovaných knihoven, byla zvolena integrace web serveru, který obsahuje knihovna PWlib.

2.5 Řízení testeru

Ještě před samotným začátkem práce na testeru bylo nutné stanovit, zda má tester v sobě obsahovat logiku pro řízení testu, nebo zda bude řízen externím skriptem. Externí skript nabízí řádově větší možnosti, takže může být uzpůsoben požadavkům uživatele. Toto byl hlavní důvod jeho výběru. Rozhraní aplikace je řešeno přes standardní vstup a výstup, což představuje pro skripty ideální řešení, a aby bylo ovládání snadné i pro uživatele, je vstup načítán přes knihovnu readline.

Kapitola 3 - Uživatelská dokumentace

3.1 Instalace

Před samotnou instalací je třeba mít nainstalované knihovny a hlavičkové soubory knihoven PWlib, OpenH323 a readline. Pro kompilaci je nutné mít k dispozici GNU make a překladač gcc. Byly testovány verze 3.4, 4.0, 4.1 a 4.2 a kompilace proběhne vždy v pořádku, pouze verze 4.2 dává warning v knihovně PWlib. Vše je funkční i se současnými cvs verzemi knihoven, přesto ale doporučuji používat verze stabilní:

Knihovna	Verze	Download
PWlib	1.10.0	http://www.voxgratia.org/releases/pwlib-v1_10_0-src-tar.gz
OpenH323	1.18.0	http://www.voxgratia.org/releases/openh323-v1_18_0-src-tar.gz
Readline	5.2	ftp://ftp.cwru.edu/pub/bash/readline-5.2.tar.gz

Na počátku nejdříve spustíme skript **./configure**, který se pokusí najít knihovny a hlavičkové soubory (na Linuxu i FreeBSD funkční). V případě, že se mu nic nalézt nepodaří, lze potřebné cesty sdělit v systémových proměnných PWLIBPATH, PWLIBLIB, OPENH323PATH, OPENH323LIB, READLINEPATH a READLINELIB. Druhý krok představuje spuštění **make** a třetím krokem je s právy roota spustit **make install**.

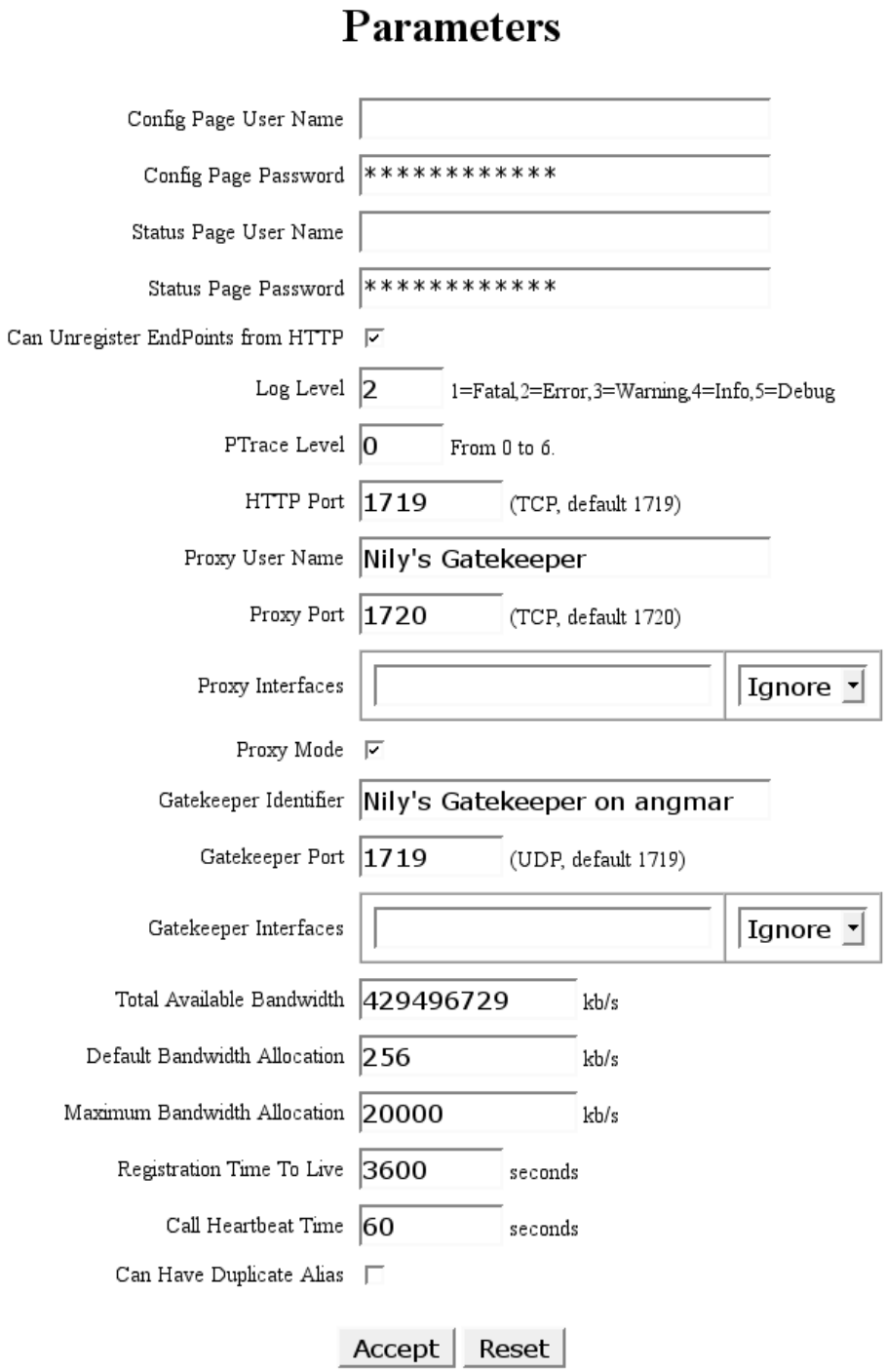
3.2 Gatekeeper (NGK, Nily's Gatekeeper)

Pro testovací účely je možné gatekeeper spouštět jako běžnou aplikaci pomocí **ngk -x**, ale pro běžný provoz doporučuji nechat aplikaci běžet jako démon s právy roota. Pro takové spouštění doporučuji použít skript **/etc/init.d/ngk start**, a to buď ručně nebo vytvořením linků v /etc/rc?.d/. Na platformách, které používají jiný systém spouštění aplikací (např FreeBSD), doporučuji pří startu systému volat **ngk -d** a při ukončení **ngk -k**.

Veškerou konfiguraci lze provádět z HTTP rozhraní, kde lze také nalézt informace o online uživatelích. HTTP server běží defaultně na portu 1719. Různé chybové, informativní a debugovací hlášky jsou zaznamenávány do systémových logů (/var/log/syslog, /var/log/debug, ...).

3.2.1 HTTP konfigurace

Konfigurace gatekeeperu nabízí tyto možnost:



Obrázek 3 - Parametry gatekeeperu

- **Config Page User Name, Config Page Password**: Jméno a heslo ke konfigurační stránce
- **Status Page User Name, Status Page Password**: Jméno a heslo

ke konfigurační stránce

- **Can Ungeregister EndPoint from HTTP**: Určuje jestli na Status stránce nabízet možnost k ručnímu odregistrování endpointů a ukončení hovorů.
- **Log Level**: Jak významné zprávy mají být logovány. 1 ... fatální chyby, 2 ... chyby, 3 ... varování, 4 ... informativní zprávy, 5 ... debugovací zprávy
- **PTrace Level**: Jak významné PTrace zprávy mají být logovány. Čím vyšší číslo, tím podrobnější zprávy. (PTrace je debugovací výpis knihoven PWlib a OpenH323.)
- **HTTP Port**: Na jakém portu má běžet HTTP konfigurace
- **Proxy User Name**: Pod jakým názvem vystupuje v telefonátech proxy
- **Proxy Port**: Port, na kterém lze přímo kontaktovat proxy
- **Proxy Interface**: Na jakých adresách má poslouchat proxy endpoint
- **Proxy Mode**: Zapnout nebo vypnout proxy mód
- **Gatekeeper Identifier**: Název gatekeeperu, jak ho vidí k němu připojené endpointy
- **Gatekeeper Port**: Port, na kterém běží gatekeeper
- **Gatekeeper Interfaces**: Na jakých adresách má poslouchat gatekeeper
- **Total Available Bandwidth**: Celkový bandwidth, který může gatekeeper přidělovat hovorům
- **Default Bandwidth Allocation**: Defaultní bandwidth přidělený novému hovoru
- **Maximum Bandwidth Allocation**: Maximální bandwidth, o který smí endpoint požádat.
- **Registration Time to Live**: Doba, na kterou je přidělená registrace platná. Po jejím uplynutí se musí endpoint znovu registrovat.
- **Call Heartbeat Time**: Jak často musí endpointy informovat gatekeeper o stavu hovoru.
- **Can Have Duplicate Alias**: Určuje, zda je možné, aby se více endpointů zaregistrovalo pod stejným jménem.

3.2.2 HTTP Status

Status

End Point Identifier	Call Signal Addresses	Aliases	Application	Active Calls	
46bb0792:1	ip$192.168.0.202:1720 ip$172.16.217.1:1720 ip$172.16.194.1:1720	userA	*Name:* Open H323 Project OhPhone *Version:* 1.4.5 (OpenH323 v1.18.0) *Vendor:* 9/61	1	Unregister
46bb0792:2	ip$192.168.0.239:1720	userB	*Name:* Open H323 Project OhPhone *Version:* 1.3.7 (OpenH323 v1.11.7) *Vendor:* 9/61	1	Unregister

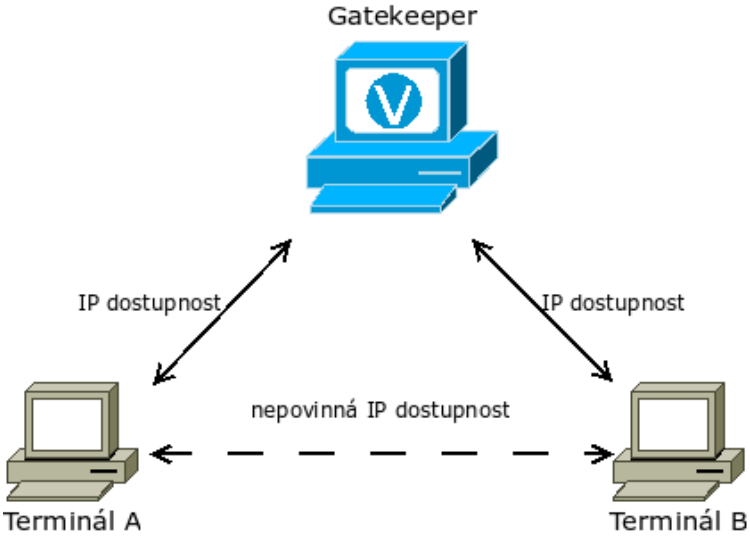
Call Identifier	End Point	Source/Destination Signalling Addresse	Last IRR	Connected	
10aaf586-e244-dc11-9e44-000c2969c980	46bb0792:2	userA [192.168.0.202] [192.168.0.80]@ip$192.168.0.80:4407 userB@ip$192.168.0.239:1720	14:35:27	14:34:32	Clear
de7adffc-e344-dc11-943d-0013d4b6bf54	46bb0792:1	userA@ip$192.168.0.202:33753 userB@ip$*:1720	14:35:01	14:34:26	Clear

Obrázek 4 - Status gatekeeperu

Stránka status obsahuje dvě tabulky. První vypisuje informace o všech zaregistrovaných endpointech a druhá informace o všech hovorech, ke kterým gatekeeper vydal povolení. Spojení, jejichž oba účastníci jsou u gatekeeperu zaregistrováni, se zobrazují dvakrát, jelikož oba museli gatekeeper požádat o svolení k hovoru (pomocí paketu ARQ).

3.2.3 Provoz

Pro plné otestování funkcí gatekeeperu je vhodné mít k dispozici tři osobní počítače, kde alespoň mezi dvěma páry existují síťová propojení (viz obrázek 5). Tyto sítě nemusejí nutně být na stejném protokolu. Například překlad hovoru mezi IPv4 a IPv6 sítí je funkční.



Obrázek 5 - síť

Na počítači gatekeeper poběží NGK a doporučuji pro něj operační systém Debian GNU/Linux. Může jít ale o jakýkoliv systém, kde jsou k dispozici (nebo kde lze zkompilovat) potřebné knihovny.

Terminály mohou běžet na jakémkoliv operačním systému, ale doporučuji Debian GNU/Linux, Ubuntu GNU/Linux, FreeBSD nebo Windows. Důvodem je dostupnost H.323 softwarového telefonu Ohphone (viz dále) na těchto operačních systémech.

Ohphone je textový H.323 telefon založený na knihovně OpenH323. Je dobře ovladatelný a má funkce, které jsem u žádného jiného telefonu nenašel (například hledání gatekeeperu multicastem). V Debianu a Ubuntu je Ohphone možné nainstalovat příkazem „apt-get install ohphone“ a ve FreeBSD příkazem „portmanager net/ohphone -l“. Pro Windows jsou k dispozici pouze několik let staré binární soubory, ale stále jsou funkční. Nalézt je lze na přiloženém CD. V balíčkovacím systému Gentoo tento program není a byla by pravděpodobně nutná ruční kompilace.

Testován byl ještě H.323 telefon Yate (obsahuje i GTK GUI) a zdá se být funkční. V současné době asi nejrozšířenější softwarový telefon Ekiga příliš nedoporučuji pro jeho tendenci k zamrzání a pádům.

Jednoduchý scénář telefonátu:

1) gatekeeper: spustit NGK příkazem „/etc/init.d/ngk start“ a na adrese „http://<adresa_počítače_gatekeeper>:1719/Parameters“ ho přepnout do Proxy módu
2) terminál A: spustit klient příkazem „ohphone -l -u userA“
3) terminál B: spustit klient příkazem „ohphone -l -u userB“
4) terminál B: vytvořit hovor příkazem „c userA“
5) terminál A: přijmout hovor příkazem „y“
6) gatekeeper: na adrese „http://<adresa_počítače_gatekeeper>:1719/Status“ se lze přesvědčit, zda vše proběhlo v pořádku
7) terminál A: ukončit hovor příkazem „h“

Na závěr kapitoly bych ještě chtěl varovat před snahou kterýkoliv ze tří zúčastněných počítačů virtualizovat pomocí programu vmware. Běh času ve virtuálním stroji může být značně odlišný od reálného času a navíc může docházet k nepravidelným změnám této rychlosti. Zažil jsem již desetkrát rychlejší či pětkrát pomalejší běh než u času reálného. Takové prostředí nesvědčí realtimeovým aplikacím jako jsou například VoIP telefony nebo proxy. Někdy se hovor může podařit, ale často není slyšet nic nebo jen nepříjemné praskání.

3.3 Tester (NGT, Nily's Gatekeeper Tester)

Druhou část aplikace tvoří tester, který je určen pro testování H.323 RAS (Registration, Admission, Status) protokolu.. Původním záměrem aplikace bylo pouze testovat, zda jednotlivé zprávy projdou přes firewall. Aplikace se od té doby ale rozšířila a nyní může být použita i pro testování gatekeeperu a klientů (ale pouze RAS). Umí vystupovat jako jednoduchý gatekeeper nebo endpoint a na základě uživatelových příkazů posílat různé požadavky druhé straně.

Ovládání je řešeno přes standardní vstup a výstup. Díky tomu by bylo proto možné napsat skripty, které celé testování zautomatizují. Aby aplikaci mohli snadno ovládat i běžní uživatelé, jde veškerý vstup přes knihovnu readline. Vstup se tedy chová obdobně jako například v programech bash, lftp nebo sqlite. Fungují kurzory doleva a doprava pro editaci již napsané řádky, šipky nahoru a dolů pro listování historií nebo C-r pro vyhledávání v historii.

Klientská část funguje jako endpoint a na základě příkazů uživatele posílá požadavky ke gatekeeperu a zobrazuje obdržené odpovědi. Řada zpráv, které H323 RAS protokol umožňuje posílat, je závislá na existenci hovoru mezi dvěma koncovými body, a proto je možné vytvořit i velmi jednoduchý „hovor“.

Serverová část funguje jako jednoduchý gatekeeper. Jejím úkolem je informovat o všech zprávách, které klienti posílají a odpovídat na ně. Umožňuje také na základě požadavku vypsat různé statistické informace o dění na serveru a informace o registrovaných klientech.

3.3.1 Vstup

Příkazy jsou čteny ze standardního vstupu a mají pevně daný formát:

<příkaz> [<parametr>=<hodnota>]...

Pokud má příkaz jen jeden jediný parametr, není třeba uvádět jeho název. Vstup pak může vypadat i takto:

<příkaz> <hodnota>

Příkaz a jednotlivé parametry jsou od sebe odděleny bílými znaky (' ', '\t'). Jestliže by je chtěl uživatel použít v jiném významu. je možné jejich oddělující význam zrušit znakem backslash ('\'). Příkazy jsou od sebe navzájem odděleny koncem řádky ('\n'). Parametry mohou být několika typů: celé číslo, řetězec znaků, pravdivostní hodnota, IP adresa a connection token a mají definovány defaultní hodnoty, které jsou použity, pokud uživatel daný parametr neuvede.

3.3.2 Výstup

Veškerý výstup je vypisován na standardní výstup a má pevně daný formát:

<čas> <kategorie> <hodnota>

Jednotlivé položky jsou odděleny mezerou. Čas je uváděn v milisekundách od 1.1.1970.

Kategorie je jeden znak, který určuje povahu vypisované události.

V případě, že zpráva obsahuje ještě nějaká přiložená data, začíná následující řádek levou složenou závorkou ('{'), pak následují data a blok je ukončen samostatnou pravou složenou závorkou ('}'). To se týká kategorií P a R.

A (Odpověď na příkaz)
Je vypsáno po vykonání jakéhokoliv příkazu a říká jeho návratovou hodnotu. Zpráva má tvar:

<čas> A <návratová_hodnota> <název_příkazu>

Návratová hodnota může nabývat hodnot:

- OK: Příkaz byl zadán správně a proběhl v pořádku
- Fail: Příkaz selhal. Důvod je většinou upřesněn pomocí chybové hlášky

I (Informace)

Výpis informací. Většinou se jedná o informace, které si uživatel před okamžikem vyžádal nějakým příkazem. Například nápověda, nastavení programu, detaily o registrovaných koncových bodech ...

S (Syntaktická nebo sémantická chyba uživatele)

Varování, že uživatel zadal příkaz chybně. Buď vstup neodpovídá syntaxi příkazu, nebo zadaný příkaz neproběhl správně.

E (Chyba v programu)

Nastala chyba, která je pravděpodobně důsledkem interní chyby aplikace, nebo selhala volaná knihovní funkce.

P (Odeslaný paket)

Za touto zprávou vždy následuje blok s dekódovaným obsahem odesílaného H323 RAS paketu.

R (Přijatý paket)

Za touto zprávou vždy následuje blok s dekódovaným obsahem přijatého H323 RAS paketu.

D (Debugovací výstup)

Tato zpráva se v jiných než testovacích verzích aplikace nezobrazuje.

3.3.3 Příkazy

Obecné příkazy

- **help**: Vypíše nápovědu k příkazu nebo příkazům
  - *command:* Název příkazu, ke kterému se má vypsat nápověda. Není-li uveden, vypíše seznam všech nyní dostupných příkazů
- **q**: Ukončí program
- **quit**: Ukončí program
- **ptrace**: Nastaví novou úroveň PTrace a soubor, kam se má PTrace výstup zapisovat

    - *file:* Soubor, do kterého se má zapisovat PTrace výstup
    - *level:* Nová úroveň PTracingu
- **client**: Přepne program do klient módu
    - *iface:* IP adresa zařízení, na kterém poslouchat
    - *port:* Port, na kterém poslouchat
- **server**: Přepne program do server módu
- **o_buffer**: Přesměruje výstupy do bufferu
    - *types:* Typy výstupů které mají být přesměrovány
- **o_file**: Přesměruje výstup do souboru
    - *file:* Soubor, kam se má zapisovat
    - *type:* Typ výstupu, který přesměrovat
- **o_stdout**: Přesměruje výstupy zpět na standardní výstup
    - *types:* Typy výstupů, které mají být přesměrovány
- **o_flush**: Vypíše nové záznamy z bufferů a souborů na standardní výstup
    - *types:* Typy výstupů, které mají být vypsány
- **o_status**: Vypíše současné nasměrování výstupů (stdout, buffer nebo file) a počet bajtů uložených v bufferech a souborech
    - *types:* Typy výstupů, které mají být vypsány

Klientské příkazy

- **gk_use**: Připojí se ke gatekeeperu. Pokud je uvedena adresa, je GRQ odesláno unicastem přímo gatekeeperu. Pokud uvedena není, je použit multicast.
    - *addr:* Adresa gatekeeperu
    - *id:* Identifikátor gatekeeperu
- **gk_rm**: Odpojí se od gatekeeperu
- **c_add**: Připojí se k jinému koncovému bodu (klient). Vzniklý hovor není funkční, ale gatekeeper to nepozná
    - *remote:* Klient, ke kterému se připojit. Adresa se udává ve formátu [<alias>@][ip$]<adresa>[:<port>]
- **c_ls**: Vypíše všechny tokeny k navázaným spojením
- **c_rm**: Odpojí se od koncového bodu
    - *token:* Token ke spojení, které chceme ukončit. Defaultně je používána

odpověď od příkazu c_token

- **c_token**: Vypíše token posledního navázaného spojení. Toto spojení je používáno jako defaultní u všech příkazů, které vyžadují parametr token
- **BRQ**: Pošle gatekeeperu požadavek, aby změnil rezervovaný bandwidth pro dané spojení
  - *bandwidth:* Nově požadovaný bandwidth
  - *token:* Token ke spojení, které chceme ukončit. Defaultně je používána odpověď od příkazu c_token
- **LRQ**: Pošle gatekeeperu požadavek o adresu registrovaného koncového bodu
  - *alias:* Alias uživatele, na kterého se ptám
- **set**: Nastaví některé parametry klienta
  - *access_token:* Nastaví nové iNow Gatekeeper Access Token OID
  - *bandwidth:* Nastaví nový defaultní bandwidth
  - *user:* Nastaví nové uživatelské jméno, které zároveň slouží i jako hlavní alias. Nastavením tohoto jména jsou všechny ostatní aliasy zahozeny
- **get**: Vypíše aktuální konfiguraci klienta. Jednotlivé parametry jsou boolovské hodnoty, které určují, zda se má daná hodnota vypisovat. Defaultně je vše nastaveno na true
  - *access_token:* Vrátí současné iNow Gatekeeper Access Token OID
  - *bandwidth:* Vrátí hodnotu defaultního bandwidth
  - *user:* Vrátí aktuální uživatelské jméno
- **a_add**: Přidá jeden alias. Nastavování aliasů má efekt pouze před navázáním spojení ke gatekeeperu. Pak již nejsou změny reflektovány
  - *alias:* Alias, který se má přidat
- **a_ls**: Vypíše všechny aliasy
- **a_rm**: Odebere alias
  - *alias:* Alias, který se má odebrat

Serverové příkazy

- **l_add**: Přidá listener
  - *iface:* IP adresa rozhraní, na kterém poslouchat
  - *port:* Port, na kterém poslouchat
- **l_clean**: Odebere všechny listenery
- **l_rm**: Odebere listener. Parametry musejí být stejné jako při přidávání listeneru
  - *iface:* IP adresa rozhraní, na kterém se poslouchá

    - *port:* Port, na kterém se poslouchá
- **set**: Nastaví některé parametry serveru
    - *bandwidth:* Celkový dostupný bandwidth ve stovkách bitů za sekundu
    - *id:* Identifikátor tohoto gatekeeperu
    - *irr:* Interval, jak často jsou monitorovány telefonáty mezi klienty. (IRR = InfoResponseRate)
    - *ttl:* Time to live pro nově registrované endpointy
- **get**: Vypíše aktuální konfiguraci serveru. Jednotlivé parametry jsou boolovské hodnoty, které určují, zda se má daná hodnota vypisovat. Defaultně je vše nastaveno na true.
    - *bandwidth_available:* Dostupný bandwidth
    - *bandwidth_default:* Defaultní bandwidth pro nová volání
    - *bandwidth_used:* Celkový využitý bandwidth
    - *calls_active:* Počet aktivních volání
    - *calls_peak:* Maximální počet aktivních volání v jeden okamžik
    - *calls_rejected:* Počet odmítnutých volání
    - *calls_total:* Celkový počet volání od nastartování gatekeeperu
    - *id:* Identifikátor tohoto gatekeeperu
    - *irr:* Interval, jak často jsou monitorovány telefonáty mezi klienty. (IRR = InfoResponseRate)
    - *registrations_active:* Počet aktuálně registrovaných koncových bodů
    - *registrations_peak:* Maximální počet registrovaných koncových bodů v jeden okamžik
    - *registrations_rejected:* Počet odmítnutých pokusů o registraci
    - *registrations_total:* Celkový počet registrací od nastartování gatekeeperu
    - *ttl:* Time to live pro nově registrované koncové body
- **registered**: Vypíše informace o všech aktuálně registrovaných koncových bodech. Jednotlivé parametry jsou boolovské hodnoty, které určují, zda se má daná hodnota vypisovat. Defaultně je vše nastaveno na true
    - *Alias:* Aliasy
    - *ApplicationInfo:* Verze a název používané aplikace
    - *Authenticators:* Informace o zabezpečení tohoto RAS spojení
    - *Descriptor:* přiřazený deskriptor
    - *IsBehindNAT:* Myslí si gatekeeper, že je klient za překladem adres?

- *Prefix:* Prefix, který tento klient akceptuje
- *ProtocolVersion:* Verze protokolu, kterým se klient registroval
- *RASAddress:* Adresy, na kterých lze tohoto klienta kontaktovat pomocí RAS protokolu
- *SignalAddress:* Adresy, na kterých lze tohoto klienta kontaktovat pomocí H.225/Q.931 protokolu (call signaling)

3.3.4 Příklady použití

Začít doporučuji zadáním příkazu **help**, který vypíše všechny dostupné příkazy. a pro výpis parametrů zvoleného příkazu zadat **help <název_příkazu>**.

Bufferování výstupu:

1) **o_buffer PR**: pro bufferování všech příchozích i odchozích paketů
2) **o_status**: pro výpis stavu bufferů
3) **o_flush PR**: pro výpis všech nových paketů v bufferech
4) **o_stdout PR**: pro vypnutí bufferů pro pakety

Spuštění testovacího gatekeeperu:

1) **server**: pro přepnutí do serverového módu
2) **l_add**: pro spuštění jednoho gatekeeper listeneru na všech síťových zařízeních a defaultním portu

Pro získávání informací o stavu serveru:

1) **get**: pro výpis současného stavu serveru
2) **registered**: pro získání informací o zaregistrovaných endpointech

Pro spuštění testovacího klienta:

1) **client port=<port>**: pro nastartování klienta na portu <port>. Doporučuji nepoužívat standardní port (1720) pokud na stejném stroji chcete provozovat

i NGK. (V defaultním nastavení na něm běží H.323 proxy.)

2) **set user=<jméno>**: nastavení jména, pod kterým se zaregistruji ke gatekeeperu
3) **gk_use**: multicastové vyhledání gatekeeperu a připojení se k němu
4) **gk_use addr=<adresa>**: připojení ke gatekeeperu na zadané adrese. Muže být užitečné, pokud selže multicastové vyhledávání gatekeeperu.

Vytvoření jednoduchého hovoru:

1) **LRQ <alias>**: pro dotaz ke gatekeeperu na polohu uživatele <alias>
2) **c_add <alias>@<adresa_vrácená_od_LRQ>**: vytvoří jednoduchou connection. Pokud je cílem jiná instance aplikace NGT, tak hovor zůstane aktivní, ale nikdy nedojde k otevření kanálů pro zvuk. V případě, že je cílem funkční telefon, tak se hovor během chvíle ukončí.
3) **BRQ bandwidth=<číslo>**: požadavek na změnu přiděleného bandwidth poslednímu otevřenému hovoru
4) **c_rm**: ukončení posledního hovoru

Kapitola 4 - Programátorská dokumentace

Projekt je psán v programovacím jazyce C++ a v co největší míře využívá možností knihoven PWlib a OpenH323. Testování proběhlo úspěšně na operačních systémech Debian GNU/Linux a FreeBSD. Kompilace by ale měla být možná i na dalších Unixech nebo i na Windows a MacOS X.

Podrobnou programátorskou dokumentaci si lze vygenerovat Doxygenem ze zdrojových souborů nebo nalézt na přiloženém cd. Zde uvádím pouze krátký popis důležitých tříd a jejich funkcí.

4.1 Gatekeeper (NGK, Nily's Gatekeeper)

Gatekeeper je z velké části tvořen třídami poděděnými z knihoven, jejichž chování je ale značně pozměněno díky novým implementacím některých virtuálních metod.

Obrázek 6 - Spárované hovory přes proxy

Telefonátu přes proxy (viz Obrázek 6) se účastní tyto třídy:

- **NGKEndPoint**: Endpoint, přes který gatekeeper komunikuje a pomocí kterého je implementována proxy. Hovory, které jsou k němu otevírány, nepřijímá, ale snaží se je přeposlat dál. Pokud zjistí dotazem ke gatekeeperu adresu volaného uživatele, tak k němu vytvoří nový hovor, do kterého jsou směrována všechna data z příchozího hovoru.
- **NGKConnection**: Reprezentuje jeden probíhající H.323 hovor.
- **NGKGatekeeperServer**: Implementace vlastního gatekeeperu.
- **NGKChannel**: Kanál, který přesouvá data mezi dvěma hovory při proxy módu. Jeden hovor ho má vždy otevřený pro čtení a druhý pro zápis.
- **Monitor**: Třída, která uchovává společná data pro dvojici hovorů v proxy módu a pomáhá při synchronizaci vláken, která se účastní telefonátu.

Průběh hovoru (viz Obrázek 6):

1) Když chce volající vytvořit hovor a pošle ARQ gatekeeperu, tak mu NGKGatekeeperServer v odpovědi (ACF) sdělí adresu NGKEndPointu.
2) Volající vytvoří hovor A k NGKEndPointu, který je zavolán knihovnou přes call back metodu OnIncomingCall a zareaguje vytvořením hovoru B k volanému. Zároveň vytvoří i novou instanci Monitoru, která bude mít na starost tento pár hovorů.
3) V call back metodě NGKEndPoint::OnAnswerCall sdělíme volajícímu, že čekáme na přijmutí hovoru uživatelem.
4) Hovor B mezitím zvoní u volaného a jakmile je přijat, je zavolána call back metoda NGKEndPoint::OnOutgoingCall. Zde je sděleno volajícímu, že přijímáme hovor A.
5) Nyní jsou vytvořena čtyři RTP spojení (dva audio kanály pro každý hovor) a každé z nich při svém vzniku zavolá NGKEndPoint::OpenAudioChannel. Za pomoci Monitoru jsou tato spojení spárována a oběma vzniklým párům je přiřazeno po jednom NGKChannelu. Ten mezi svými RTP spojeními přeposílá audio data.
6) Když jeden z hovorů skončí, metoda NGKEndPoint::OnConnectionCleared informuje o této události Monitor a ten ukončí druhý hovor.
7) Konec druhého hovoru opět zavolá NGKEndPoint::OnConnectionCleared. Tentokrát je v této metodě zničen již nepotřebný Monitor a celý telefonát tím končí.

Druhou důležitou funkcí gatekeeperu je jeho HTTP server, který je součástí třídy **NGK**. Tato třída je vstupním bodem celé aplikace a při startu nebo změně konfigurace inicializuje jak HTTP server, tak celý gatekeeper. Status stránku vytváří třída **StatusPage** a konfigurační stránku vytvářejí společně při své inicializaci třídy **NGK**, **NGKEndPoint** a **NGKGatekeeperServer**. Toto řešení bylo zvoleno z důvodu, aby zúčastněné třídy mohly do stránky vložit aktuální hodnoty svých parametrů.

4.2 Tester (NGT, Nily's Gatekeeper Tester)

Vstupním bodem celé aplikace je třída **NGT** a její jediná činnost spočívá v inicializaci třídy Space a spuštění smyčky pro načítání příkazů.

Třída **Space** spravuje všechny aktivní příkazy podporované aplikaci. Ty jsou přidávány metodou AddCommand, jsou uchovávány v containeru map, kde jsou indexovány podle svých názvů. Třída Space zároveň implementuje hlavní smyčku programu v metodě MainLoop. Zde jsou spouštěny příkazy, které uživatel zadal na vstup. Smyčka běží tak dlouho, dokud neskončí vstup nebo není zavolán příkaz quit.

Abstraktní třída **Command** je předkem všech příkazů a její nejvýznamnější virtuální metodou je Run, kterou musí každý její potomek implementovat. Každý příkaz může podporovat řadu parametrů (viz třída Parameter), které uchovává v containeru map. Hodnoty jsou parametrům předávány ve formě pole řetězců, který již před tím naparsovala třída Input.

Abstraktní třída **Parameter** je předkem všech parametrů, které příkazy mohou mít. Abstraktní virtuální metody, které každý parametr musí implementovat, jsou Load a Reset. Metoda Load načte hodnotu parametru z řetězce, který na vstupu zadal uživatel a metoda Reset vrátí parametr zpět na jeho defaultní hodnotu. Hodnota parametru je uchovávána jako void pointer, což umožňuje, aby si zde potomek uložil jakýkoliv typ bude potřebovat.

Třída **Input** načítá a parsuje vstup uživatele. Čtení probíhá metodou ReadCommand, která využije služeb knihovny readline a poté načtený řádek rozdělí podle bílých znaků. První prvek typicky obsahuje název příkazu a další prvky mají formu <název_parametru>=<hodnota>.

O výstup aplikace se stará třída **Output**. Výstup je rozdělen do řady kategorií (viz kapitola 3.3.2) a Output si pro každou z nich udržuje v poli outputs jeden ostream pointer. To umožňuje jednotlivé kategorie přesměrovávat na standardní výstup (cout), do bufferu (ostringstream) nebo do souboru (ofstream).

Třídy **NGTEndPointServer** a **NGTGatekeeperServer** implementují gatekeeper a třída **NGTEndPointClient** je jednoduchý koncový bod, který slouží jako gatekeeper client.

K vypisování příchozích a odchozích paketů složí v klientském módu **NGTTransport** a v serverovém módu **NGTGatekeeperListener**. Tyto třídy mají přístup k veškerému síťovému provozu, což využívají k dekódování a vypisování všech paketů.

Kapitola 5 - Možnosti budoucího rozšiřování aplikace

Pří analýze a vývoji aplikace byla nalezena inspirace pro možná budoucí rozšíření aplikace.

- Tester by mohl být rozšířen o testování call signaling a RTP spojení.
- Gatekeeper by mohl komunikovat s dalšími gatekeepery v sítí a tím umožnit volání přes několik proxy zároveň. Potom by byly možné i telefonáty mezi klienty, kterým v cestě stojí několik NATů nebo firewallů.
- Gatekeeper by mohl umět autentizovat uživatele jménem a heslem a tím omezit, kdo všechno smí proxy používat.

Obsah přiloženého CD

Součástí bakalářské práce je CD, které obsahuje:

- Zdrojové kódy programu
- Dokumentaci programu
- Text bakalářské práce ve formátech pdf a odt (pro OpenOffice)
- Obrázky a diagramy použité v bakalářské práci ve formátech png a dia (pro Dia Diagram Editor)
- Zdrojové kódy použitých knihoven
- Dokumentaci použitých knihoven
- Zdrojové kódy programu Ohphone
- Program Ohphone v binární podobě pro Microsoft Windows

Literatura

[1] GnuGk NAT Traversal,
http://www.gnugk.org/nat-traversal.html

[2] H.323 standards,
http://www.packetizer.com/voip/h323/standards.html

[3] H.323 Firewall/NAT Traversal White Paper,
http://www.h323forum.org/papers/301005_Firewall_NAT_Traversal_White_Paper.pdf