VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SERVER PRO AUTOMATICKOU KONFIGURACI IPV6 TUNELU

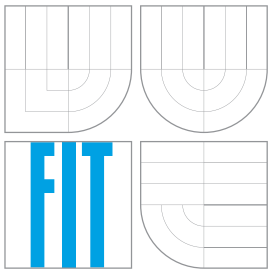
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. VOJTĚCH DRAHOŠ

BRNO 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SERVER PRO AUTOMATICKOU KONFIGURACI IPV6 TUNELU

SERVER FOR AUTOMATIC IPV6 TUNNEL CONNECTIVITY

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. VOJTĚCH DRAHOŠ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. PETR MATOUŠEK, Ph.D.

BRNO 2011

Abstrakt

Velkým problémem dnešního Internetu je blížící se vyčerpání adresního prostoru protokolu IPv4. Tato diplomová práce se zabývá studiem síťového protokolu IPv6 a možnostmi podporujícími rozšíření tohoto protokolu v síti Internetu. Dále studuje speciálně k tomu účelu určenou metodou tunelování AYIYA a způsoby řízení tunelů, obzvláště protokol TIC. Cílem této práce je navrhnut, implementovat, otestovat a nasadit do pilotního provozu serverovou část tunel broker systému, který bude využívat zmíněnou metodu tunelování a bude kompatibilní s volně šiřitelným klientem AICCU. Hlavním přínosem práce je vytvoření otevřeného tunel broker systému. Pilotní provoz v České republice přináší českým uživatelům tunelovaný IPv6 Internet s nízkou latencí.

Abstract

One of the major problems of modern Internet is the upcoming depletion of addressing space in IPv4 protocol. This master thesis focuses on the study of network protocol IPv6, the possibilities of expanding this protocol in the Internet using the appropriate tunelling method and ways of tunnel management, especially the TIC protocol. Major goal of this work is using knowledge of protocols in a design and implementation of the server part of tunnel broker system, which uses mentioned tunneling method and is compatible with the open source client AICCU. Second part of this work deals with testing and pilot deployment of this system. Main benefit consists in creation an open source tunel broker system. Real deployment of this system is particularly advantage for Czech users as a low latency tunnel IPv6 Internet.

Klíčová slova

IPv6, AYIYA, TIC, Tunel broker systém, Tunel server, Virtuální síťové rozhraní, Linux

Keywords

IPv6, AYIYA, TIC, Tunnel broker system, Tunnel server, Virtual network device, Linux

Citace

Vojtěch Drahoš: Server pro automatickou konfiguraci IPv6 tunelu, diplomová práce, Brno, FIT VUT v Brně, 2011

Server pro automatickou konfiguraci IPv6 tunelu

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Petra Matouška.

.....
Vojtěch Drahoš
19. května 2011

Poděkování

Chtěl bych poděkovat svému vedoucímu Petru Matouškovi za vedení a odborné rady. Dále pak Emanuelu Petrovi a Ondřeji Surému za zajištění serverového zázemí pro vývoj a provoz systému.

Obsah

Kapitola 1

Úvod

1.1 Motivace

Síť Internetu se s postupem času zvětšuje. Nynější hlavní směrovací protokol IPv4 již v některých ohledech nedostačuje současným potřebám. Nejkritičtější vlastností je nedostatek adres pro všechna zařízení připojená k sítí Internetu.

Následníkem IPv4, který problém řeší, je protokol IPv6. Jeho masivní rozšíření mezi běžné uživatele Internetu je ovšem stále problém. Tato práce se zabývá jednou z metod tunelování protokolu IPv6 přes stávající síťovou infrastrukturu a má napomoci zavádění protokolu IPv6 do širšího používání se zaměřením na Českou republiku.

Tato práce vznikla ze zadání správce české národní domény CZ.NIC. V současné době neexistuje svobodná implementace tunel broker systému podporující metodu tunelování AYIYA a právě CZ.NIC chce v rámci svých laboratoří takový systém provozovat. Na rozdíl od jiných podobných metod tunelování jako např. Teredo, AYIYA poskytuje klientům pevné IPv6 adresy a možnost budovat své IPv6 sítě. Další metody tunelování jako ISATAP, 6to4 nebo 6in4 vyžadují ke své činnosti veřejnou IPv4 adresu či jiné speciální podmínky. Metoda AYIYA je vhodná pro každého koncového uživatele a nevyžaduje žádné speciální podmínky.

1.2 Cíle projektu

Cílem této práce bylo nastudovat, navrhnout a implementovat tunel broker systém, který bude používat výše zmíněné protokoly a bude kompatibilní s klientem AICCU (Automatic IPv6 Connectivity Client Utility).

Výstupem práce bude fungující tunel broker systém, který bude nasazen v pilotním provozu v rámci výzkumných aktivit registrátora české národní domény, sdružení CZ.NIC.

1.3 Struktura práce

Informace o principu fungování Internetu, důvodu zavádění a vlastnostech protokolu IPv6 naleznete v kapitole 2. Kapitola obsahuje i část popisující možnosti zavádění nového protokolu do světa Internetu. Tato kapitola dále obsahuje teoretické informace o tunelovacím protokolu AYIYA (Anything In Anything) a metodách určených k sestavení a řízení tunelů TIC (Tunnel Information and Control Protocol) a TSP (Tunnel Setup Protocol).

Obsah kapitoly 3 se zabývá návrhem tunel broker systému. V text se věnuji jednotlivým částem systému a jejich vzájemné komunikaci jako fungující na venek celistvý systém. Tunel

broker systém obsahuje základní rozdělení na tunelovací server, server implementující službu potřebnou k domluvení parametrů tunelu a webový portál, který je použit jako rozhraní pro komunikaci s uživateli systému. Kapitola navrhuje databázový datový model používaný celým systémem a možným adresním plánem protokolu IPv6 pro tunelovací servery.

Kapitola 4 popisuje implementaci všech částí systému. Konkrétně se jedná o službu serveru TIC, tunelovacího serveru a rozebírá i některé části webového rozhraní. Popis programů a skriptů, které nesouvisí s přímou funkčností celého systému, ale hodí se při jeho nasazování, je popsán rovněž.

Bezpečností systému se zabývá kapitola 5. Můžete v ní najít různé teoretické situace cílených i nechtěných útoků na server TIC a tunelovací servery. Útoky jsou vždy rozebrány od motivace útočníka až po možné dopady.

Další kapitola 6 obsahuje souhrn testů, které jsem na systému prováděl. Jedná se o jednoduché testy, které ověřují správnou funkčnost systému až po testy, které měří propustnost tunelovacího serveru a zpoždění paketů při zpracování tunelovacím programem až po testy, jenž sledují vlastnosti systému, když obdrží podvržená data.

Kapitola 2

Teoretický úvod

Tato kapitola obsahuje teoretický základ protokolů IPv6, AYIYA a TIC. Z těchto informací vycházím v navazujících kapitolách.

Část popisující protokol IPv6 se zabývá uvedením čtenáře do problematiky. Text poskytuje základní informace o protokolu a věnuje se také koexistenci s protokolem IPv4 včetně přechodových metod z IPv4 na IPv6.

Metoda tunelování AYIYA je rozvedena v části 2.2. Naleznete zde detailní popis datové hlavičky a možností protokolu AYIYA.

Poslední část této kapitoly 2.3 se zabývá protokolem pro sestavení a řízení tunelů TIC. Tato část se také zmiňuje o protokolu TSP s podobnou funkcionalitou.

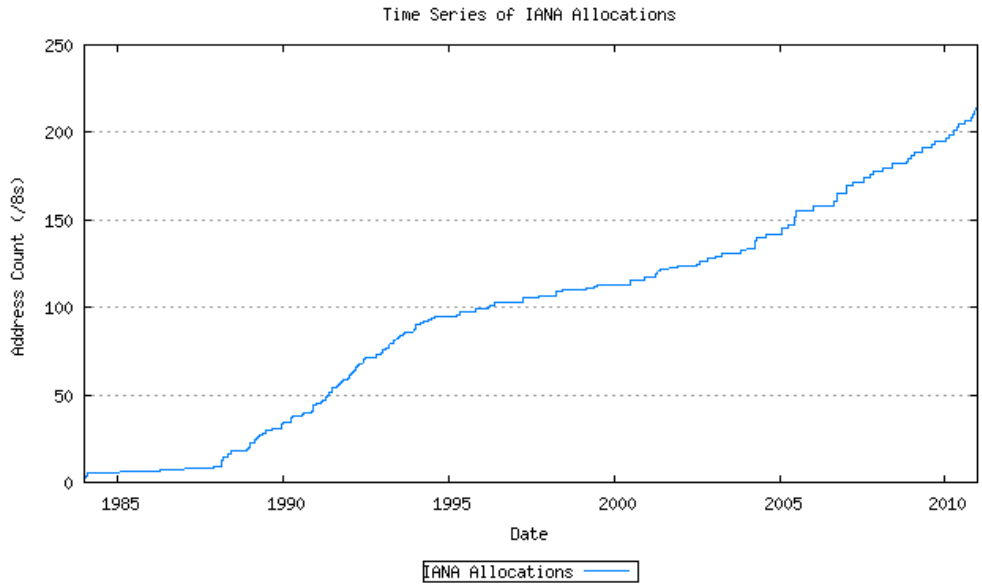
2.1 IPv6

Základním principem fungování celé sítě Internetu je zasílání zpráv mezi připojenými zařízeními. Aby zařízení mohla mezi sebou komunikovat, musí mít každé jednoznačnou adresu. Princip je stejný jako ten, který známe z reálného života. Chceme-li například poslat dopis známému, musíme pro poštu uvést přesnou adresu, kam dopis doručit. Poštovní adresa obsahuje jméno adresáta, stát, město, ulici s číslem popisným a směrovací číslo. Pro člověka zdánlivě lehce zapamatovatelný a logický formát ovšem příliš nevyhovuje strojům, jako jsou počítače.

V dobách vzniku Internetu jeho tvůrci definovali protokol, který se měl starat o adresaci zařízení (počítačů). Jedná se konkrétně o Internet Protokol verze 4 (IPv4), jenž měl pro adresování zařízení vyhrazen prostor 32 bitů. Na takovém paměťovém prostoru je možné rozlišit maximálně $2^{32} = 4\,294\,967\,296$ adres zařízení. Původní vlastností protokolu IPv4 bylo také třídní adresování. Adresy se dělily do tříd A, B, C, D a E. Některé měly speciální využití jako třída D pro multicast[1], další byly rezervovány a nesměly se používat (třída E).

Pro následující text předpokládám znalosti adresování protokolu IPv4. V počátcích Internetu se adresami značně hýřilo a velké firmy jako například HP nebo IBM dostávaly přiděleny adresní rozsahy třídy A, které obnáší 2^{24} adres. Již v počátku devadesátých let bylo zřejmé, že adresní rozsah bude relativně brzy vyčerpán. Tehdejší studie předpokládaly vyčerpání adresního prostoru do deseti let, což byl dostatek času na návrh nového protokolu, který by řešil více neduhů protokolu IPv4 než jen nedostatečně velký adresní prostor. Nový protokol byl označován jako IP Next Generation (IPng) nebo nověji Internet Protokol verze 6 (IPv6) [25]. Mezeru v číslování verzí IP protokolů tvoří experimentální Internet Stream

[1] Multicast je metoda přeposílání IP datagramů z jednoho zdroje skupině více koncových stanic.



Obrázek 2.1: Graf přidělování /8 adresních bloků vydávaných IANA čerpán z [4].

Protokol [27] (IPv5), který byl specifikován v roce 1979. Byl navržen pro proudový přenos dat jako connection-oriented protokol a dokonce garantoval Quality of Service (QoS). Do veřejného použití se ovšem nedostal.

Současně s vývojem nových protokolů se komunita IETF[2] snažila o zmírnění rychlosti ubývání současného adresního prostoru. Velkým snížením rychlosti čerpání adres bylo zavedení technologie CIDR, tedy Classless Inter−Domain Routing, která umožnila přidělovat adresní bloky o různé bitové délce. Obrázek 2.1 zobrazuje počet přidělených adresních bloků s prefixem /8 administrovaný organizací IANA[3]. Za zmírnění strmosti křivky přibližně v roce 1995 mohlo zavedení CIDR a následně dalších nástrojů jako NAT [16].

U zrodu protokolu IPv6 stála velká řada požadavku, jako například:

- Rozsáhlý adresní prostor, který vydrží nejlépe navždy.
- Optimalizace směrování zaměřená na rychlost.
- Zvýšení bezpečnosti pomocí šifrování a autentizace.
- Automatická konfigurace.
- Podpora mobility.

[2]Internet Engineering Task Force vyvíjí a podporuje Internetové standardy.

[3]Internet Assigned Numbers Authority volně přeloženo jako autorita pro přidělování čísel používaných na Internetu. Organizace dohlíží celosvětově na přidělování IP adres, správu kořenových zón DNS a další náležitosti Internetových protokolů.

- Plynulý přechod z předchozího protokolu.

Hlavním problémem protokolu IPv4 byl nedostatek adres. Proto se tvůrci nového protokolu rozhodli pro velmi razantní navýšení adresního prostoru na 128 bitů, jenž dává k dispozici nepředstavitelných $3.4 \cdot 10^{38}$ adres.

Myšlenka rychlejšího směrování se zakládá na více skutečnostech. Jedním ze základních kamenů úrazu je proměnlivá velikost IPv4 hlavičky. Rychlé směrování musí být akcelerováno pomocí speciálního hardware. Jak bude uvedeno v 2.1.1, protokol IPv6 má pevnou velikost hlavičky, a proto se lépe v hardware implementuje. Další důraz byl kladen na optimalizaci počtu směrovacích záznamů v páteřních směrovačích. Proces přidělování adresního prostoru je následující. IANA přiděluje adresní bloky Regionálním Internetovým registrátorům (RIR), kterých existuje pět a pokrývají celé území planety. Aby se adresy dostaly až ke koncovému zařízení, vystupuje zde ještě jeden druh organizace a to Lokální Internet Registrátor (LIR), což bývá většinou poskytovatel připojení k Internetu. Postup přidělování adres je logický, IANA spravuje celý adresní prostor a jednotlivým regionálním registrátorům přiděluje velké adresní bloky, které se dále rozdělují mezi lokální registrátory. Ti pak jednotlivé adresy nebo malé bloky přidělují zákazníkům. IPv6 na rozdíl od svého předchůdce disponuje obrovským množstvím adres, a tak lze jednotlivým Internet Service Provider (ISP) přidělit mnohem větší bloky, které pokryjí řádově více zákazníků než bloky přidělované u IPv4. Takto jednotliví ISP nebudou nuceni propagovat dlouhou řadu nenavazujících malých rozsahů, které v průběhu času dostávali přiděleny, a jejich počet se tak zmenší.

2.1.1 IPv6 hlavička

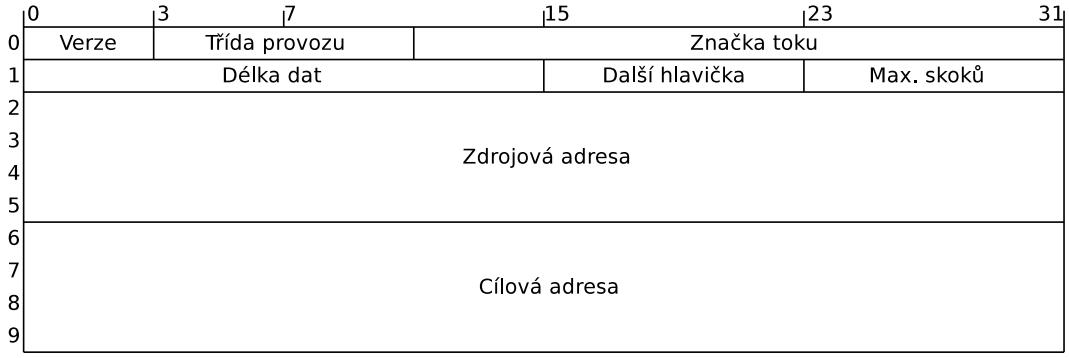
Jak je tomu běžné v počítačových sítích, pro přenos zprávy mezi počítači je zapotřebí vrstvového modelu protokolů, zajišťujících správný průchod sítí. Pro nás zajímavý síťový protokol je IPv6. Formát IPv6 datagramu obsahuje hlavičku protokolu, za kterou následují hlavičky výše postavených protokolů s daty. Obrázek 2.2 zobrazuje rozložení základních prvků IPv6 hlavičky. Z obrázku lze vyčíst velikost hlavičky a to čtyřicet bajtů.

V porovnání s IPv4 hlavičkou se čtyřikrát zvětšila velikost adresy příjemce a odesilatele, přesto však došlo k nárůstu celkové velikosti pouze na dvojnásobek. Základní hlavička IPv6 datagramu totiž neobsahuje některé informace z IPv4. Typickým příkladem je fragmentace nebo kontrolní součet. Veškeré doplňující, nepovinné či příležitostné informace byly přesunuty od rozšiřujících hlaviček. O jejich přítomnosti jsou uzly informovány polem *Další hlavička*. Pokud datagram žádné rozšiřující hlavičky neobsahuje, je v poli informace o hlavičce následující vrstvy typicky TCP nebo UDP.

Stejně jako u protokolu IPv4 zahajuje hlavičku položka *verze*. Jedná se o čtyřbitovou hodnotu obsahující číslo šest. Stejně tak IPv4 obsahovala hodnotu čtyři.

Jako další následuje osmibitová *třída provozu*, která vypovídá o zařazení datagramu do určité přepravní třídy. Cílem položky je umožňovat služby se zaručenou kvalitou. V reálném nasazení ovšem slouží jen pro poskytování tzv. diferencovaných služeb, jejichž prostřednictvím mohou mít datagramy různé priority či způsoby zacházení, které vedou k priorizaci či naopak odkládání datagramů. Specifikace IPv6 nijak položku blíže neupřesňuje, pouze vyžaduje implicitní hodnotu nula.

Značka toku je novou položkou o délce dvacet bitů. Koncept datových toků má v podstatě označovat proud datagramů se společnými vlastnostmi jako odesilatel, adresát a požadavky na přenos. Podle značky toku by měl směrovač rychle identifikovat datagram a uplat-



Obrázek 2.2: Hlavička IPv6 datagramu.

nit na něj stejná pravidla jako na ostatní pakety téhož toku. V současné době se pole používá pouze pro experimentální účely.

Délka dat obsahuje počet bajtů následujících za standardní hlavičkou. Tedy délka případných rozšiřujících hlaviček, protokolů vyšších vrstev a samotných přenášených dat až do konce datagramu. Pro položku je vyhrazeno šestnáct bitů. Maximální délka datagramu je omezena na 64 KB. Pokud z nějakého důvodu potřebujeme vytvořit větší datagram, musíme použít rozšiřující hlavičku „Jumbo obsah“.

Pole *další hlavička* je osm bitů dlouhé a označuje, jaká data následují bezprostředně za základní hlavičkou. Může se jednat o rozšiřující hlavičky nebo hlavičky protokolů vyšších vrstev. Každá rozšiřující hlavička opět obsahuje pole typu následující hlavičky. Tímto způsobem lze rozšiřující hlavičky řetězit. Pořadí hlaviček při řetězení je logicky uspořádáno podle toho, která zařízení při přenosu je mohou využít. Hlavičky důležité pro koncovou stanici budou umístěny až za těmi, které mohou použít směrovače a obdobně. Každá rozšiřující hlavička může být zřetězena pouze jednou. Více se rozšiřujícími hlavičkami zabývat nebudu. Zájemce odkazuji na zdroj [13]. Odpovídající hodnoty jednotlivým protokolům přiděluje a udržuje organizace IANA. Aktuální seznam můžete nalézt na [1].

Maximum skoků zabírá posledních osm bitů před položkami adres odesilatele a adresáta. Je to obdoba položky Time To Live (TTL) z hlavičky IPv4 a slouží k tomu, aby datagramy neputovaly špatně nakonfigurovanou sítí do nekonečna.

2.1.2 Adresy v IPv6

Jak jsem již zmínil dříve, velikost adresy v protokolu IPv6 byla stanovena na 128 bitů. Autoři frustrováni rychlostí čerpání adres u IPv4 se rozhodli nový protokol značně naddimenzovat. Mohutnost adresního prostoru vyjádřená číslem má 38 nul. Rozpočteme-li adresy na povrch naší planety, vyjde nám, že na jeden čtvereční milimetr připadají milióny miliard adres. Doufám, že adresní protokol IPv6 nenarazí na limity svého adresního prostoru, i když, jak uvidíme později v části 2.1.2, se s adresami hodně plýtvá.

Formát pro zapisování IPv6 adresy je dán jako sekvence osmi částí čtyř hexadecimálních čísel oddělenými dvojtečkami, což dává dohromady oněch 128 bitů. Protože takto zapsané adresy jsou oproti adresám IPv4 velmi dlouhé a často obsahují dlouhou sekvenci nul, můžeme adresu zkrátit použitím dvou dvojteček „::“ za sebou, které v zápisu nahradí sekvenci nul. Zápis dvou dvojteček lze ovšem v adrese použít pouze jednou. Další povolené zkrácení adresy je možné provést v rámci každé čtveřice tehdy, pokud obsahuje nuly z levé

strany. Takové nuly není potřeba do zápisu udávat a lze je aplikovat na libovolnou čtveřici. Jako příklad uvedu adresu `2a01:0430:0011:0001:0000:0000:0000:0abc`.

Můžeme zkrátit v libovolných čtveřicích nuly z levé strany:

`2a01:0430:11:0001:0000:0:0:abc`

Jak je vidět, nuly v hexadecimálních číslech mohu, ale nemusím vynechat. Adresa z příkladu bude po zkrácení všech čtveřic vypadat:

`2a01:430:11:1:0:0:0:abc`

Tento tvar je ale stále příliš dlouhý. Použitím sekvence dvou dvojteček mohu zkrátit delší sekvenci nul.

`2a01:430:11:1::abc`

Jako další příklad uvedu adresu `0000:0000:0000:0000:0000:0000:0000:0001`, která se používá pro loopback rozhraní. Zkrácený zápis adresy je:

`::1`

Extrémním případem je nespecifikovaná adresa `0000:0000:0000:0000:0000:0000:0000:0000`, kterou lze zkrátit na pouhé dvě dvojtečky:

`::`

Druhy adres

Stejně jako v IPv4 se adresy přidělují jednotlivým rozhraním a ne celým počítačům. U IPv4 mohlo mít rozhraní více adres, ale ve většině případů mělo pouze jednu. Naproti tomu IPv6 dokonce přikazuje používat vícero adres na každém funkčním rozhraní. Existují tři druhy adres s odlišným chováním.

- **Individuální** (unicast) adresy se používají pro identifikaci rozhraní prakticky stejně jako u IPv4.
- **Skupinové** (multicast) adresy se používají pro adresování skupin zařízení. Pokud někdo odešle data na skupinovou adresu, musí být doručena všem členům skupiny.
- **Výběrové** (anycast) adresy jsou novinkou protokolu IPv6. Výběrové adresy označují skupinu stejně jako multicastové, ale data jsou doručena pouze jednomu členu skupiny - tomu nejbližšímu.

Je vidět, že proti IPv4 zmizely všesměrové (broadcast) adresy. Jejich funkci převzaly skupinové adresy. Protokol IPv6 definuje různé skupinové adresy, např. adresa `ff02::1` obsahuje všechna zařízení na lince a nahrazuje tak původní všesměrovou adresu lokální sítě.

Prefixy

Příslušnost adresy k dané síti nebo podsíti se vyjadřuje síťovým prefixem. Všechna rozhraní na jedné podsíti mají stejný prefix. Jak slovo prefix napovídá, jedná se o začátek adresy. Délka prefixu se zapisuje na konec adresy za znak lomítko jako odpovídající počet bitů prefixu. Délka IPv6 adresy je 128 bitů, prefix tedy může nabývat hodnot $0 - 128$. Prefix 0 se používá pro celou globální síť – `::/0`. Prefix 128 má také své použití pro síť která má pouze jednu adresu. Typickým použitím je adresa rozhraní loopback `::1/128`. Adresa přidělená rozhraní má většinou délku prefixu 64 bitů, například výše použitá adresa rozhraní s délkou 64 bitů bude vypadat takto:

`2a01:430:11:1::abc/64`

Adresa sítě, do které rozhraní spadá má adresu:

`2a01:430:11:1::/64`

Jak bude vidět v 2.1.4, směrování vychází z nejdelší shody prefixů směrovací tabulky a cílové adresy.

2.1.3 Automatická konfigurace IPv6 adres

Trendem dnešní doby je tzv. Plug and Play přístup, tedy připojit se a fungovat bez manuální konfigurace. Protokol IPv6 poskytuje dva přístupy, jak může rozhraní získat automaticky adresu, jedná se o stavový a nestavový přístup.

Stavová konfigurace

Podobně, jak tomu bylo u IPv4, stavový přístup zajišťuje Dynamic Host Configuration Protocol, tentokrát ve verzi 6 (DHCPv6) [14]. Základem je server spravující konfigurační parametry, kterého se klienti dotazují.

Přidělení adresy probíhá v následujících krocích. Klient zašle svůj požadavek na dobře známou skupinovou adresu, na níž naslouchají všechny DHCPv6 servery. Jako zdrojovou adresu použije linkovou adresu, kterou si sám přiřadil z rozsahu `fe80::/10`. Tato adresa má dostupnost pouze na lokální lince. Klientem požadovaná individuální adresa z DHCPv6 serveru má globální dosah. Všechny servery poslouchající na skupinové adrese klientovi zašlou své nabídky, ten si vybere jeho srdci nejbližší adresu a pomocí mechanismu objevování sousedů ověří, zda adresa není na síti již používána.

Tento popis byl velmi obecný, detailní informace o protokolu DHCPv6 najdete v [14]. Mechanismus objevování sousedů v této práci nevysvětluji. Zájemci se mohou dozvědět více v [22].

Bezstavová konfigurace

Novinkou protokolu IPv6 je bezstavová konfigurace. Zabývá se jí [26]. Principem je, že se v síti nachází směrovač, který pravidelně zasílá zprávy na skupinovou adresu všech zařízení obsahující informaci o prefixu sítě, kterou směrovač poskytuje. Tato informace stačí pro vytvoření adresy na rozhraní. Zařízení si pak ze své MAC adresy rozhraní pomocí formátu modifikovaného EUI-64 a oznámeného prefixu vytvoří celou adresu. Modifikované EUI-64 je popsáno v příloze A zdroje [19]. Například směrovač oznamuje prefix `2001:15c0:679b::/64` a dané rozhraní má MAC adresu `f0:4d:a2:8a:78:d1`. Výsledkem modifikovaného EUI-64 bude adresa `2001:15c0:679b:0:f24d:a2ff:fe8a:78d1/64`.

2.1.4 Směrování

Základní funkcí IP protokolů je snaha o doručení paketu od odesilatele k příjemci. Všechna zařízení podporující IPv6 protokol musí umět směrovat své pakety. Hlavními kameny jsou směrovače, které se starají o předávání paketů mezi sítěmi. Nejjednodušší směrování lze provádět na základě automatické konfigurace. Nicméně v praxi značně převažují směrovací tabulky i u jednoduchých zařízení.

Směrovací tabulka obsahuje minimálně údaje o cíli (adresa a prefix sítě), výstupním rozhraní a adrese směrovače. Jednotlivé záznamy v tabulce se definují pro různé cíle. Cíle jsou identifikovány prostřednictvím IPv6 adresy s prefixem.

Je-li cíl na stejné síti, stačí pro správné směrování uvést pouze jméno rozhraní. Naopak, pokud cíl není ve stejné síti, musí směrovací tabulka obsahovat adresu nejbližšího odpovídajícího směrovače. Takové chování se uplatňuje v případě broadcastové linkové technologie jako je například Ethernet. Máme-li jako linkovou technologii k odpovídajícímu směrovači dvoubodový spoj (P2P), stačí v záznamu uvést jméno rozhraní. Adresa směrovače není potřeba, protože P2P linková technologie zajišťuje, že co na rozhraní pošleme, to druhá strana obdrží.

Koncovým stanicím si stačí plnit svou směrovací tabulku přímo připojenými sítěmi, defaultní cestou naučenou z automatické konfigurace, nebo staticky konfigurovaných prefixů.

Velké směrovače na Internetu si ovšem s těmito principy nevystačí a aby mohly optimálně směrovat, musí se naučit cestu ke vzdáleným sítím. K tomuto účelu se používají dynamické směrovací protokoly. Výměnu směrovacích informací protokolu IPv6 podporují například RIPng, OSPFv3, BGP4+ nebo IS-IS.

2.1.5 Koexistence IPv4 a IPv6

Jedním z původních cílů návrhářů IPv6 byl plynulý přechod z IPv4. Ze současného pohledu je vidět, že to byl velmi těžký úkol. Existuje sada RFC dokumentů, které popisují jednotlivé fáze nasazování IPv6 protokolu i s doporučenými termíny realizace. Nicméně koncem roku 2010, kdy IANA vlastní posledních pět bloků IPv4 adres, jejichž rozdělení je již dané (každý RIR dostane jeden), je stále rozšíření IPv6 na pováženou.

Současná situace velmi zaostává za doporučeným průběhem nasazování protokolu IPv6. Koncoví uživatelé se nedožadují zavedení IPv6 svých poskytovatelů připojení k Internetu, protože na IPv6 je dostupných jen velmi málo služeb a kdo by na IPv6 své služby implementoval, když na něm nemá dostatek potencionálních zákazníků.

Vyčerpání IPv4 adres odhodlává více poskytovatelů Internetu k nabízení IPv6 konektivity. V současné době ovšem na trhu ještě nejsou žádné hotové nabídky, poskytovatelé zatím hovoří o fázi investic a vývoje. Na českém trhu majoritní dodavatel konektivity koncovým uživatelům O2 plánuje nabídnout IPv6 v druhé polovině roku 2011.

Internet v dnešní době tvoří převážně protokol IPv4. Nelze tedy jednoduše říct, že od určitého data zahodíme IPv4 a vše bude fungovat na IPv6. Jsou potřeba mechanismy, které umožní koexistenci obou protokolů. V následujícím textu se pokusím přiblížit různé možnosti koexistence obou protokolů.

Dvojí zásobník

Dvojí zásobník znamená, že zařízení je připojeno k oběma světům a je schopno komunikovat oběma protokoly. Tato varianta není příliš zajímavá, protože vyžaduje zachování IPv4. Nicméně další dvě metody vyžadují některá zařízení s dvojím zásobníkem.

Název mechanismu „dvojí zásobník“ je lehce zavádějící. Systémy mohou mít naimplementován pouze jeden hybridní zásobník s podporou obou protokolů. Důležité je pouze to, že je stanice schopna komunikovat oběma protokoly a má jak IPv4 tak i IPv6 adresu. IPv4 adresu získá statickou konfigurací nebo z DHCP, pro IPv6 může použít statickou konfiguraci nebo některou z automatických konfigurací (bezstavovou nebo DHCPv6). DNS klient na zařízení se musí orientovat jak v A tak i v AAAA záznamech pro IPv6.

Translátory

Translátory, neboli překladače, se snaží o vytvoření spojnice mezi IPv6 a IPv4 světem. Využití najde v případech, kdy máte vytvořit síť podporující pouze IPv6 protokol a klienti takové sítě chtějí spojit se službu dostupnou pouze v IPv4 Internetu. V takovém scénáři musí existovat zařízení s dvojím zásobníkem, které bude překládat čistě IPv6 pakety na IPv4 pakety.

Existuje několik služeb translátorů jako Stateless IP/ICMP Translation (SIIT) [23], Network Address Translation Protocol Translation (NAT-PT) [28], Transport Relay Translator (TRT) [18] nebo Bump In The Stack (BIS) [29]. Žádná z těchto technologií se nedočkala masivního nasazení. I nejnadějnější způsob NAT-PT bývá označován za historický kvůli důvodům popsaných zde [8]. Výhoda zařízení zprostředkující transparentní komunikaci mezi oběma světy byla natolik velká, že se začal specifikovat jeho nástupce NAT64 a DNS64, který je dnes již ve formě RFC dokumentů [10] a [9].

2.1.6 Tunelování

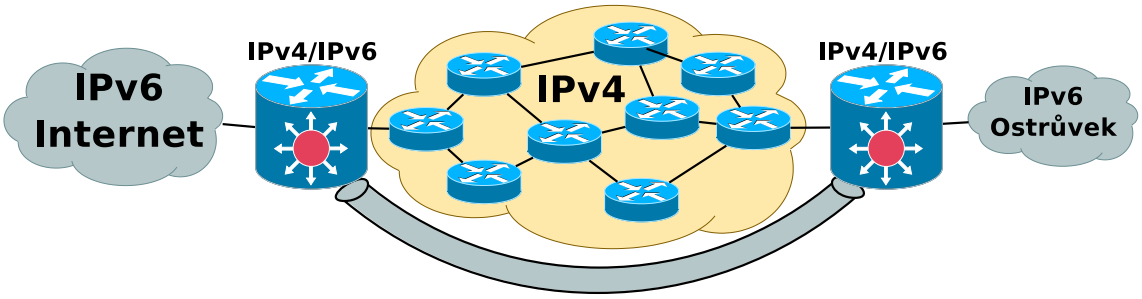
Protože tématem mé diplomové práce je software vytvářející tunelovací server, zkusím metodu tunelování blíže rozebrat.

Tunelování v počítačových sítích není nic nového. Obecně se jedná o zabalení jednoho protokolu do jiného. V počátečních fázích nasazování IPv6 je třeba tunelovat IPv6 datagramy tak, aby prošly IPv4 sítí.

Tunel má dva konce s IPv4 adresami. Pokud zařízení na jedné straně tunelu potřebuje doručit IPv6 datagram druhému, musí celý IPv6 datagram vložit do nově vytvořeného IPv4 datagramu. Cílovou adresu nového datagramu nastaví na IPv4 adresu druhého konce tunelu, zdrojovou nastaví na svou IPv4 adresu. Protože se jedná o tunelovaný IPv6 datagram, nastaví odesilatel hodnotu 41 do položky *Protokol* v IPv4 hlavičce. Tento způsob tunelu bývá označován jako *6in4*. Nový IPv4 datagram se běžně odešle. Druhá strana datagram přijme. Podle hodnoty pole *Protokol* 41 v hlavičce IPv4 příjemce pozná, že se jedná o tunelovaný datagram. Vybalí jej a naloží s ním jak s nově příchozím datagramem, tentokráte již IPv6.

Při přenášení IPv4 sítí vystupuje původní IPv6 paket jen jako přenášená data protokolem IPv4. Z toho vyplývá, že se po cestě jeho hlavička ani obsah IPv4 směrovače nemění. IPv6 datagram tedy vnímá cestu tunelem jako jeden skok a vybalující směrovač zmenší položku *Max. skoků* v IPv6 hlavičce o jedna. Obrázek 2.6 ukazuje typické použití tunelu, který připojuje IPv6 ostrůvek ke zbytku IPv6 světa přes IPv4 infrastrukturu. Takto lze tvořit spojení mezi různými částmi IPv6 světa.

Existují dva základní režimy tunelování – automatický a konfigurovatelný. Jak se dá očekávat, rozdíl mezi nimi je, že automatický tunel nevyžaduje konfiguraci a sestavuje se automaticky. Nejrozšířenější variantou automatického tunelování je metoda *6to4* s oficiálním názvem „Propojování IPv6 domén IPv4 sítěmi“. Zabývá se jí [12].



Obrázek 2.3: Mechanismus tunelování.

Cílem *6to4* tunelů je poskytnout možnost propojit IPv6 sítě přes IPv4 infrastrukturu s minimální konfigurací. Nutnou podmínkou pro sestavení *6to4* tunelu je, aby zařízení na obou koncích měly veřejné IPv4 adresy. Je běžnou praxí, že koncové IPv4 sítě mají k dispozici pouze jednu veřejnou adresu a zbytek sítě je připojen přes směrovač využívající dynamický překlad adres (NAT). Typické uspořádání pak vypadá tak, že *6to4* realizuje místní směrovač. Obecně může *6to4* realizovat kterékoli zařízení, ale je-li za NATem, musíme zajistit transparentnost NATu k protokolu 41.

Konfigurovatelný režim tunelování znamená, že na zařízení byly zadány příkazy vytvářející virtuální rozhraní. O odesílání datagramů tímto rozhraním rozhoduje směrovací tabulka v níž je uvedeno, pro které cíle se má tunel použít. Konfigurovatelné tunely se často používají pro permanentní propojování sítí nebo k zapojení počítače, který není k IPv6 světu nativně připojen.

Tunely nemají návaznost fyzické topologii IPv4 sítě, kterou procházejí. Doporučuje se ovšem k topologii přihlédnout a vybrat optimální bod připojení tunelu.

2.1.7 Tunelovací servery

Do kategorie konfigurovatelných tunelů spadají i tunel servery. Jedná se o nadstavbu nad explicitní konfigurací. V systému tunel serverů vystupují dva základní prvky. Tunel server a tunel broker server. Tunel server je zařízení, které je připojeno jak IPv4 tak i IPv6 a je ochotno posloužit jako konec tunelu pro kohokoli, kdo má zájem. Tunel broker server zastává roli registračního portálu uživatelů a může obsluhovat více tunel serverů. V takovém případě uživateli přidělí vzhledem k zadaným parametrům nejvhodnější tunel server.

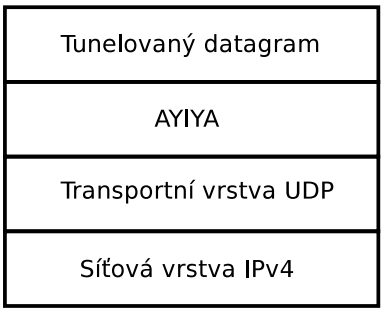
Výměnu informací potřebných na sestavení tunelu mezi klientem a serverem zajišťují speciální protokoly. Nejznámějším protokolem je Tunel Setup Protocol (TSP) nebo Tunnel Information and Control protocol (TIC), který je předmětem kapitoly 2.3.

Konkrétní návrh systému tunel serverů včetně uzlu tunel broker naleznete v kapitole 3.

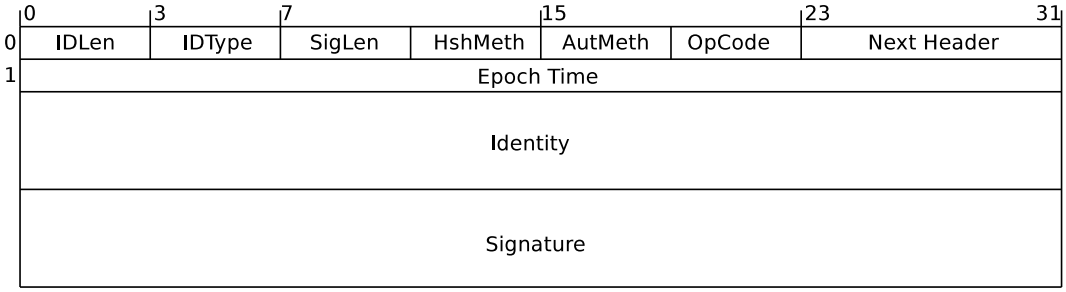
2.2 AYIYA

AYIYA je zkratka anglického názvu *Anything In Anything*, což lze přeložit jako „cokoli v čemkoli“. AYIYA je tunelovací protokol, který může být vložen jako payload (náklad) do jiného protokolu. Protokol ještě nedospěl do fáze RFC. Jeho návrh je dostupný na webu [20]. Oficiální číslo portu používané pro tento protokol je 5072.

V dnešní době je mnoho uživatelů Internetu situováno za technologií Network Address Translator (NAT), která umožňuje navazování spojení pouze zařízením z vnitřní sítě.



Obrázek 2.4: Vrstvový model tunelování IPv6 protokolu přes IP/UDP/AYIYA.



Obrázek 2.5: Hlavička protokolu AYIYA.

Spojení směrem do vnitřní sítě není možné. Takové chování je proti základnímu principu Internetu (kterékoli zařízení může navázat komunikaci s jakýmkoli jiným). V sekci 2.1 je popsán důvod, proč k došlo k nasazení NATu. Standardní tunelovací technika *proto-41* se v některých případech rozebraných v [17] nedá na zařízení umístěných za NATem navázat.

Problém NATu lze vyřešit použitím tunelování IPv6 protokoly transportní vrstvy jako UDP, TCP nebo SCTP. Budoucí transportní protokoly budou moci být také použity. AYIYA může být obsahem přímo IPv4 nebo IPv6 protokolu, což umožňuje tunelování s minimální režií.

Typické použití tunelování IPv6 protokolu přes IPv4 pomocí transportního protokolu UDP protokolem AYIYA je na obrázku 2.4. Nad síťovou vrstvou se nachází transportní vrstva jako například UDP. Další hlavičkou v pořadí je hlavička AYIYA, která jako data obsahuje tunelovaný datagram. Pomocí AYIYA můžeme přenášet jak IP pakety, tak i rámce linkové vrstvy.

2.2.1 Hlavička protokolu AYIYA

Obrázek 2.5 ukazuje formát AYIYA hlavičky. V následujícím popisu budu používat anglické označení jednotlivých položek, protože české alternativy nejsou ustáleny.

Všechny položky hlavičky uplatňují Big-endian[4] architekturu. Položky *Identity* a *Signature* nejsou povinné. Minimální velikost hlavičky může být pouze osm bajtů.

[4]Endianita řeší uspořádání více bajtových dat. Big-endian ukládá na nejnižší adresu nejvíce významný bajt a až za něj se ukládají ostatní bajty až po nejméně významný. Little-endian je přesně opačný.

IDLen

Položka *IDLen* (Identity Length) je dlouhá čtyři bity a uvádí délku položky *Identity* v mocninách čísla dvě v bajtech.

$$IDLen = 4, \text{ pak délka položky } \textit{Identity} \text{ bude } 2^4 = 16 \text{ bajtů.}$$

IDType

Identity Type specifikuje, jaký typ identity je v hlavičce použit. Položka zabírá čtyři bity. Aktuální návrh protokolu [20] definuje následující typy:

- **0x0** - žádný
- **0x1** - číslo (integer)
- **0x2** - ASCII řetězec

Specifikuje-li *IDType* typ „žádný“, pak položka *Signature* následuje přímo za *Epoch Time* a *IDLen* musí mít hodnotu 0.

Typ „číslo“ naznačuje, že pole *Identity* obsahuje číslo v binárním formátu. Může to být například IPv4 nebo IPv6 adresa či číslo označujícího konkrétního odesilatele.

Položka *Identity* typu „ASCII řetězec“ může obsahovat např. DNS název nebo nějaký specifický název odesilatele. ASCII řetězce jsou zarovnány znakem NULL do konce délky pole *Identity*.

SigLen

SigLen (Signature Length) určuje délku položky *Signature* (podpis) jako čtyřnásobek bajtů. Teoretická maximální délka pole *Signature* je 60 bajtů.

$$SigLen = 3, \text{ pak délka položky } \textit{Signature} \text{ bude } 3 \cdot 4 = 12 \text{ bajtů.}$$

HshMeth

Další z řady čtyřbitových položek je *Hashing Method*, která určuje typ metody kontrolního součtu. Ověřením kontrolního součtu celého paketu můžeme zjistit, zda se paket po cestě mezi odesilatelem a příjemcem nezměnil. Aktuální návrh definuje tyto:

- **0x0** - bez kontrolního součtu
- **0x1** - MD5 kontrolní součet
- **0x2** - SHA1 kontrolní součet

Protože jsou v současné době známy kolize metody kontrolního součtu MD5, je výchozí metodou SHA1.

Je-li použit typ „Bez kontrolního součtu“, položky *AuthMeth* a *SigLen* musí být nulové a hlavička nebude obsahovat pole *Signature*.

AuthMeth

Authentication method popisuje typ autentizační metody. Autentizací paketu můžeme ověřit, že odesilatel paketu je opravdu ten, za kterého se vydává. AYIYA nemá žádnou možnost šifrování tunelovaného obsahu. Návrh definuje následující autentifikační metody:

- **0x0** - bez autentizace
- **0x1** - kontrolní součet používající sdílené tajemství
- **0x2** - kontrolní součet používající metodu privátního/veřejného klíče

Pokud implementace nepodporuje přijatý typ identity nebo podpisu, musí přijatý paket zahodit a může upozornit odesilatele.

OpCode

OpCode neboli „Operační kód“ je poslední čtyřbitová položka, jenž označuje speciální význam zprávy. Jsou definovány tyto operační kódy:

- **0x0** - Bez operace / Heartbeat - paket slouží k aktualizování času posledního obdrženého paketu viz 2.2.2. Jedná se o režijní protokolu AYIYA a nemá žádný obsah (*Next Header* musí mít hodnotu 59. Viz [1]).
- **0x1** - Forward - za hlavičkou se nachází tunelovaný datagram.
- **0x2** - Echo Request - obsah paketu za hlavičkou musí být vrácen odesilateli jako typ Echo Response. Jedná se o testování živosti protistrany tunelu.
- **0x3** - Echo Request and Forward - navíc oproti předchozímu bodu je obsah za hlavičkou tunelovaný datagram a je potřeba jej vybalit a přeposlat.
- **0x4** - Echo Response - odpověď na Echo Request.
- **0x5** - MOTD - Message Of The Day dovoluje odesilateli zaslat zprávu příjemci, který ji může zobrazit uživateli. Stejně jako u kódu Heartbeat za hlavičkou nenásleduje žádná další hlavička, a tedy *Next Header* musí mít hodnotu 59. Obsah přenášené zprávy musí dodržovat přesný formát, který je popsán v [20].
- **0x6** - Query Request - pomocí Query Request a Query Response může odesilatel získat informace o protistraně jako například kontakt, jméno hostitele, verzi software a další.
- **0x7** - Query Response.

Next Header

Next Header je osmibitová a stejně jako u protokolu IPv6 obsahuje hodnotu identifikující následující hlavičku. Hodnota 59 se používá tehdy, když za AYIYA hlavičkou nenásleduje hlavička jiného protokolu. Kompletní seznam je na webu [1].

Epoch Time

Epoch Time je 32 bitů dlouhý, ne znaménkový integer a obsahuje počet sekund od počátku roku 1970 UTC. Aby tunel fungoval, nesmí mít strany rozdílný čas. Proto se doporučuje používat časovou synchronizaci pomocí protokolu NTP. Přijde-li příjemci paket, ve kterém se *Epoch Time* liší s lokálním časem o více, než je stanovené maximum (obyčejně 120 s), je paket zahozen. Časový limit je dostatečně velký, aby pokryl čas průchodu paketu sítí. Tato položka je zahrnuta jako ochrana proti replay útokům[5].

V roce 2038 dojde k přetečení tohoto čítače. Algoritmus zpracovávající *Epoch Time* údaj se musí s touto situací vyrovnat.

Identity

Položka *Identity* oznamuje příjemci zprávy, kdo je její původce. Identifikace odesilatele na základě adresy a portu není jistá, neboť vlivem NATu, DHCP nebo např. PPP protokolu se tyto údaje mohou v čase měnit. Obsah položky je domlouván mimo rámec tohoto protokolu.

Signature

Signature je nepovinná položka, která může obsahovat otisk paketu pro zajištění integrity. Pokud je zvolena i autentizační metoda, lze zajistit i autentizaci odesilatele.

2.2.2 Heartbeat

Obě strany tunelu si udržují povědomí o aktivitě protistrany a v případě delší nečinnosti tunel uzavírají. Funkce *Heartbeat* slouží k posílání režijních paketů v pravidelných intervalech, čímž dochází k aktualizaci času poslední aktivity druhé strany tunelu. Používají se zprávy s operačním kódem $0x0$. Typicky timeout bývá nastaven na 120 sekund a Heartbeat posílá každých 60 sekund jednu nebo více zpráv. Heartbeat komunikace se nemusí provádět, pokud jsou protokolem posílány data nebo jiné režijní zprávy.

Vedlejším pozitivním efektem této funkce je, že i při situaci, kdy se přes tunel nepřenášejí žádná data, pravidelně aktualizují záznamy v případných NAT tabulkách a nedochází tak ke ztrátě spojení. Režie komunikace je zanedbatelná.

2.2.3 Podepisování paketu

Pakety, které neobsahují v AYIYA hlavičce pole *Signature*, nemohou být podepsané. Pokud přijatý paket obsahuje příznak, že hlavička obsahuje kontrolní součet nebo autentizaci, pak příjemce musí verifikovat korektnost podpisu tak, že provede stejné operace jako odesilatel a výsledky porovná. Pokud výsledky odpovídají, může se paket dále zpracovat. V opačném případě jej musí zahodit a případně může informovat uživatele.

Kontrolní součet paketu

Pokud není vyžadována autentizace, vytváří se kontrolní součet paketu s inicializovaným polem *Signature* na NULL. Zbytek položek hlavičky a obsah odpovídají paketu při odeslání. Podpis je vytvořen nad kompletním paketem pomocí zvolené metody kontrolního součtu přes celou AYIYA hlavičku a obsah paketu. Výsledek je umístěn do pole *Signature*.

[5] Replay útok je formou síťového útoku, ve kterém jsou validní data vysílána podvodně, opakována nebo zpožďována.

Podepisování se sdíleným tajemstvím

Metoda je stejná jako čistý kontrolní součet paketu, ale místo inicializace pole *Signature* na NULL se inicializuje právě sdíleným tajemstvím. Tento přístup je jednoduchý na implementaci a vyžaduje sdílené tajemství určité délky tak, aby se vyplnila celá paměť vyhrazená pro *Signature*. Domlouvání sdíleného tajemství je mimo rozsah tohoto protokolu.

Podepisování umožňuje zjistit, zda byl paket po cestě mezi odesilatelem a příjemcem modifikován a zda odesilatel znal sdílené tajemství.

Obrázek 2.6: Příklad AYIYA komunikace.

2.3 TIC

Tunnel Information and Control protocol, zkráceně TIC, je textově orientovaný protokol, který dovoluje programům získávat parametry konfigurace AYIYA IPv6 tunelů. To značně ulehčuje život uživatelům, kteří takto nemusí mít znalost protokolu IPv6 nebo tunelovacích technik.

V současné době jediný oficiální dostupný zdroj informací o protokolu je [5]. Tvůrce protokolu je IPv6 Broker SixXS. IANA přidělila protokolu oficiální TCP port číslo 3874.

Protokol je typu klient/server, kde klient zadává příkazy a server odpovídá. Výsledek příkazu uvozuje stavový kód obdobně jako například u protokolu SMTP. Kódy z řady 200 pro úspěšný příkaz a 400 pro neúspěšný příkaz. Za stavovým číslem pokračuje zpráva popisující výsledek nebo chybu.

2.3.1 TSP protokol

Stejnou službu jako TIC poskytuje standardizovaný Tunnel Setup Protocol (TSP) definovaný v [11]. TSP je signalizační protokol k nastavení parametrů tunelu mezi jejich konci. Protokol TSP umožňuje domluvit mnoho druhů zapouzdření tunelů, jako například IPv6 v IPv4, IPv4 v IPv6 nebo tunely průchozí technologií NAT – IPv6 v UDP v IPv4 a další. TSP používá ke komunikaci XML zprávy a podporuje autentizaci pomocí SASL [21].

Zadaný tunelovací klient AICCU neobsahuje podporu TSP protokolu a umí zjistit parametry tunelu pouze protokolem TIC.

2.3.2 Příkazy TIC protokolu

Protokol TIC obsahuje řadu příkazů, které najdete na stránkách SixXS [5]. Pro naše potřeby uvedu pouze několik příkazů s popisem.

Příkaz	Popis
get unixtime	slouží ke zjištění času ve stejnojmenném formátu
starttls	snaží se zapnout šifrování komunikace pomocí TLS
client	s parametry TIC\ <version> <name> \ <version> <osname> \ <version> hlásí serveru informace o své verzi a operačním systému, na kterém běží
challenge	říká, jakým způsobem se bude klient autentizovat. Používaná varianta je md5. Odpovědí je náhodný řetězec.
authenticate	provádí autentizaci klienta, viz 2.3.3
tunnel list	po úspěšné autentizaci příkaz vypíše seznam dostupných tunelů
tunnel show	s parametrem *identifikátoru tunelu* vypíše podrobné informace o tunelu
quit	ukončuje relaci

2.3.3 Možnosti autentizace

Protokol podporuje autentizaci pomocí přenosu hesla v otevřené podobě nebo metodu *md5*. Přenos hesla v otevřené podobě je velmi nebezpečný, a proto je doporučeno tento způsob vůbec nepoužívat.

Metoda *md5* porovnává kontrolní součet kontrolního součtu hesla spojeného s řetězcem z příkazu *challenge* na straně klienta a serveru. Autentizaci klienta provádí server.

2.3.4 Ukázka komunikace

Protože informací o protokolu TIC je k dispozici opravdu málo, odchytil jsem reálnou komunikaci AICCU klienta s serverem TIC SixXS. Modré zprávy zasílá server a červené klient.

```
200 SixXS TIC Service on noc.sixxs.net ready (http://www.sixxs.net)
client TIC/draft-00 AICCU/2007.01.15-console-linux Linux/2.6.34-gentoo-r6
200 Client Identity accepted
get unixtime
200 Client Identity accepted
get unixtime
200 1290112080
starttls
400 This service is not SSL enabled (yet)
username VDZ1-SIXXS/T19031
200 VDZ1-SIXXS/T19031 choose your authentication challenge please
challenge md5
200 94b7e0ababf3c2f5ad9d6bf57d482ee6
authenticate md5 d020cfdafd336aa8305a5a01a6c0bb9c
200 Successfully logged in using md5 as VDZ1-SIXXS
tunnel list
201 Listing tunnels
T19031 2001:15c0:65ff:1f0::2 ayiya simbx01
202 <tunnel_id> <ipv6_endpoint> <ipv4_endpoint> <pop_name>
tunnel show T19031
```

```
201 Showing tunnel information for T19031
TunnelId: T19031
Type: ayiya
IPv6 Endpoint: 2001:15c0:65ff:1f0::2
IPv6 POP: 2001:15c0:65ff:1f0::1
IPv6 PrefixLength: 64
Tunnel MTU: 1280
POP Id: simbx01
IPv4 Endpoint: ayiya
IPv4 POP: 212.18.63.73
UserState: enabled
AdminState: enabled
Password: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
Heartbeat_Interval: 60
202 Done
QUIT We just stopped breating
```

Shrnutí

Kapitola obsahuje přehled vlastností protokolu IPv6, přibližuje adresaci, směrování a možnosti automatické konfigurace adres IPv6. Ukázala možnosti koexistence obou protokolů v podobě zařízení s dvojím zásobníkem. Část 2.1.6 vysvětlila základní princip tunelovací architektury. Metoda tunelování AYIYA byla podrobně popsána v části 2.2. Poslední část této kapitoly 2.3 se věnovala protokolu TIC, který je určen k sestavování tunelů.

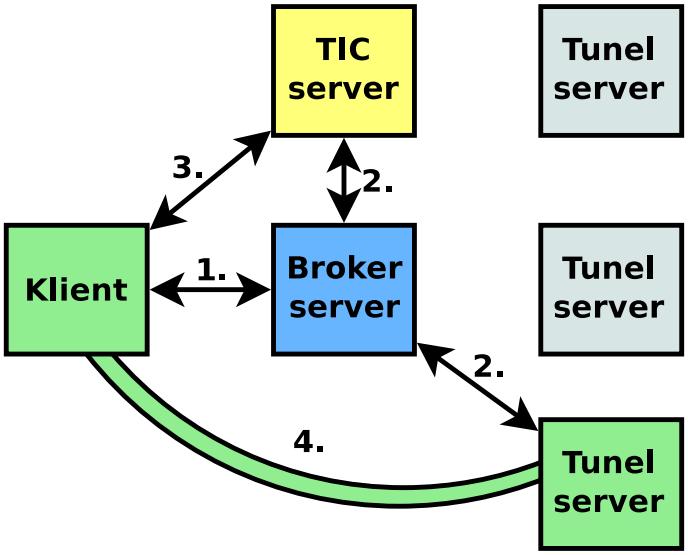
Kapitola 3

Návrh systému

Jedním z cílů tohoto projektu je navrhnout, a v rámci diplomové práce naimplementovat a uvést do pilotního provozu, brokerský tunelovací systém poskytující IPv6 konektivitu. Problematikou IPv6 tunel brokerů se zabývá [15] a poskytuje řadu doporučení pro tvorbu takových systémů.

Systém má být dle zadání vystavěn na míru klientům používající svobodný multiplatformní software Automatic IPv6 Connectivity Client Utility (AICCU), který pochází od tunel brokera SixXS [2].

Obecné schéma systému je na obrázku 3.1. Systém obsahuje tzv. *Broker server*, více *tunelovacích serverů* a *server TIC*. Všechny tyto servery jsou vysvětleny v následujícím textu.



Obrázek 3.1: Základní schéma Tunel Broker.

Systém nabízí pro své uživatele možnost připojení se do IPv6 světa. Jednoduchý seznam kroků, které proto musí udělat, je na obrázku 3.1 vyznačen čísly. Prvním krokem je registrace na Broker serveru a zažádání systému o přidělení tunelu.

Pokud systém uživatele přijme, musí kontaktovat vybraný tunel server a sdělit mu konfiguraci pro správné vytvoření tunelu s uživatelem. Na obrázku je to naznačeno šipkou

č.2. V tomto bodu se můj návrh od [15] liší. Literatura udává, že Broker server uživateli předá nastavovací skript, který uvede do provozu jeho stranu tunelu. Tento přístup zavádí mnohé problémy spojené s rozličností operačních systémů, které uživatelé používají. Navíc ne každý uživatel je natolik odborně znalý, aby instalaci v pořádku provedl. Místo toho bude navržený systém používat server TIC. Detaily jsou popsány v sekci server TIC.

Klient AICCU využívá protokol TIC k získání potřebných informací k nastavení tunelu. Šipka č.3 tedy označuje komunikaci klienta s serverem TIC. Aby server TIC byl schopný předat klientovi potřebné informace, které byly vygenerovány u registrace tunelu, musí sdílet stejné datové uložiště nebo si informace musí pravidelně předávat viz 3.4.

Posledním bodem je navázání samotného tunelu s konkrétním tunelovacím serverem. Následující text přiblíží důležité části systému.

3.1 Broker server

Broker server je místo, kde se mohou uživatelé registrovat, spravovat své tunely a jim přidělené prefixy. Systém bude umožňovat tunel vytvořit, povolit, zakázat, zrušit nebo modifikovat jeho MTU.

Nejsnadnější a nejdostupnější pro uživatele bude forma webové aplikace. Tato služba by měla být dostupná jak z IPv4, tak i z IPv6 Internetu.

3.2 Server TIC

Klient AICCU podporuje protokol TIC, který je popsán v části 2.3. TIC server klientovi sdělí všechny potřebné údaje k vzniku tunelu. Služba musí komunikovat pomocí protokolu IPv4, protože její klienti v čase domlouvání ještě nemají IPv6 konektivitu. Doporučuje se, aby adresa serveru měla pouze DNS *A* záznam. Komunikace mezi klientem a TIC serverem obsahuje informace, které by měly zůstat utajeny. Proto protokol TIC obsahuje podporu šifrování pomocí TLS[1].

Obecně může systém mít více TIC serverů, ale protože jde o velmi jednoduchou službu potřebnou pouze u fáze vzniku tunelu, nepředpokládám větší počet než jeden.

3.3 Tunelovací server

Tunel server je směrovač s dvojím zásobníkem připojen jak IPv4, tak i IPv6 konektivitou, který je ochotný poskytovat tunelovanou IPv6 konektivitu registrovaným uživatelům systému. Tunelovacích serverů může systém obsahovat více. Jejich rozmístění záleží na politice a možnostech poskytovatele. Je vhodné, aby tunelovací server byl co možná nejblíže klientům. Nemám na mysli vzdálenosti reálného světa, ale počítačových sítí. Počet tunelovacích serverů by se také měl odvíjet od vytížení již nasazených serverů.

3.4 Datový model

Protože všechny části systému potřebují přístup k datům, například tunel server k datům o identitě protistrany nebo TIC server ke kompletním informacím o validitě tunelu, uživatelským údajům atd., rozhodl jsem se pro používání MySQL databáze jako styčného datového

[1] Transport Security Layer je protokol poskytující zabezpečenou komunikaci pomocí asymetrické kryptografie.

bodu. Aby celý systém nebyl závislý jen na jedné kopii MySQL databáze, je vhodné využít systém MySQL Replikace nebo MySQL Cluster.

Zamýšlel jsem se také nad použitím SQLite databáze. Oproti MySQL ovšem nepodporuje síťový přístup a musí být mezi servery distribuována pomocí kopírování souborů. Pokud dojde ke změně v hlavní kopii na broker serveru, budou se muset změny rozšířit mezi všechny ostatní servery. Ke kopírování SQLite souborů bude docházet jen, pokud budou kopie databáze nekonzistentní, a to v pravidelných intervalech, protože propagování kopií při každé změně by mohlo servery u větší databáze omezovat. Navíc tento přístup vnáší do systému nepříjemný efekt zpoždění šíření informací.

MySQL databáze přináší do systému závislost na cizích balíčcích, ale výhody, které poskytuje nad SQLite, převažují. Proto chci primárně do systému naimplementovat podporu MySQL a jako rozšíření SQLite.

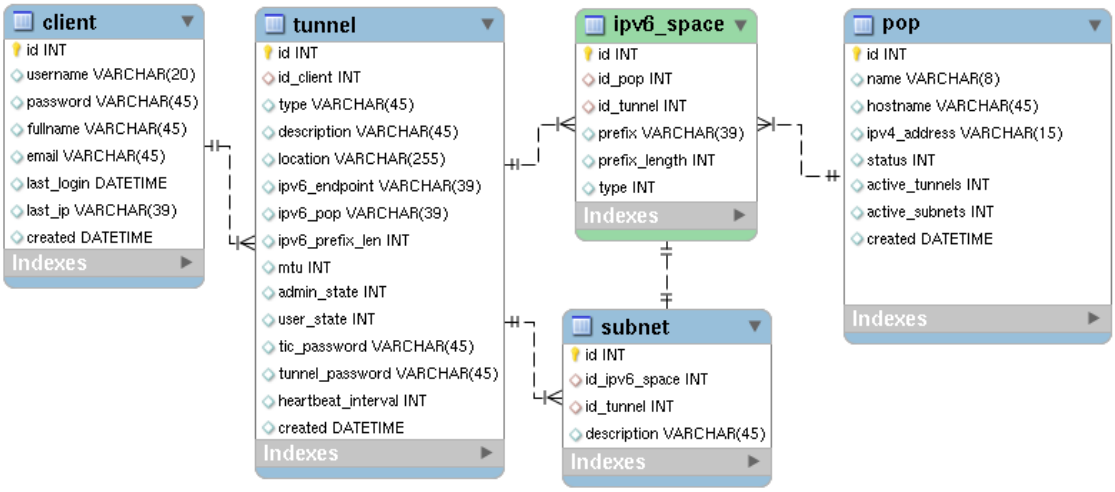
Relační model databáze je na obrázku 3.2. Tabulka `client` obsahuje základní informace o uživatelích jako přihlašovací jméno, heslo atd. Primárními klíči ve všech tabulkách jsou atributy *id*. Tabulka obsahující údaje o jednotlivých tunelech se jmenuje `tunnel`. Tabulka `subnet` je v relaci 1:N s tabulkou `tunnel`, protože přiřazené prefixy se váží na konkrétní tunel a jeden tunel může mít více nasměrovaných prefixů. Informace o jednotlivých tunelovacích serverech nese tabulka `pop`, jejíž název vychází z anglického „Point of presence“. Při instalaci nového tunelovacího serveru se naplní informace do tabulky `pop` a vloží se záznamy do tabulky `ipv6_space` obsahující prefixy náležící danému tunelovacímu serveru. Podle atributu *type* se rozlišuje význam prefixů uložených v tabulce `ipv6_space`. Systém je navrhnut na základní tři typy prefixů. Typ 0 se použije pro přiřazení adres konců tunelu, tedy síť mezi klientem a serverem. Na klientskou stranu každého vytvořeného tunelu systém automaticky směruje jeden prefix pro účely klientské sítě. Typ 1 je určen pro tyto automaticky přidělené prefixy. Posledním typem je typ 2, který se použije pro další typicky větší prefixy, které si uživatel systému může vytvořit, pokud mu pro svou síť za koncem tunelu nedostačuje automaticky přidělený prefix. Délky automaticky a manuálně přidělených prefixů lze velmi jednoduše nastavit již při vkládání záznamů do tabulky `ipv6_space`. Tato tabulka je v relaci s tabulkami `tunnel` a `subnet`. Pokud je prefix v tabulce ještě nepřidělen, nemá nastaven cizí klíč *id_tunnel*. Druhá relace s tabulkou `subnet` je typu 1:1, protože jeden prefix vždy odpovídá jednomu záznamu v `ipv6_space`. Význam ostatních atributů tabulek je patrný z jejich pojmenování a neslouží k relacím mezi tabulkami.

3.5 Adresní plán IPv6

V této části budu vycházet z plánovaného vytížení tunelovacích serverů a co možná nejjednoduššího externího směrování.

Hlavním kritérium ke zvážení je plánovač operačního systému. S každým aktivním tunelem bude zapotřebí vlákno, které bude provádět blokující čtení souborového deskriptoru reprezentujícího TUN rozhraní. Primárním operačním systémem na tunel serverech bude Linux, jehož standardní plánovač CFS považuje vlákno za samostatný plánovací prvek (schéma one-to-one). Více než 500 aktivních vláken znamená pro plánovač obtíže. Z pohledu plánovače bude ovšem většina vláken stále čekat na příchozí data, a proto nebudou v hlavní plánovací frontě, což umožní počet tunelů zvýšit.

Lze také předpokládat, že ne všechny tunely budou aktivní. Možným zmenšením počtu vláken je funkce `select()`, pomocí níž by jedno vlákno mohlo číst data z více souborových deskriptorů.



Obrázek 3.2: Relační schéma návrhu databáze.

Předpokládejme tedy, že kapacita jednoho tunel serveru je 1024 tunelů a pro celý systém máme vyhrazen IPv6 prefix `2001:430::/32`. Na každý tunel budeme potřebovat spojovací síť obdobnou u IPv4 prefixem `/30`. U IPv6 je dobrým zvykem pro propojovací sítě používat bloky s prefixem dlouhým `/64` bitů. Pro 1024 tunelů budeme tedy muset vyhradit dalších deset bitů, například prefix `2001:430::/54`.

Systém také počítá s přidělením vlastního prefixu uživateli, který bude nasměrován na druhou stranu tunelu. Podle [12] je vhodné koncovým zákazníkům přidělovat pro jejich směrovací potřeby prefixy délky `/48` bitů. Systém musí být navržen tak, aby mohl každému tunelu přiřadit i směrovaný prefix. Opět pro počet 1024 tunelů musíme vyhradit prefix s délkou `/38` bitů, například `2001:430:400::/38`.

Všechny tyto prefixy lze shrnout prefixem `2001:430::/37`. Z toho vyplývá, že z jednoho `/32` prefixu můžeme adresy rozdělit až $2^{37-32} = 2^5 = 32$ tunel serverům.

3.6 Implementované části systému

Na konec návrhu bych chtěl projít principy hlavních části, které budu implementovat. Jedná se o tři části: Tunel server, server TIC a Broker server.

Broker server - webové rozhraní budu implementovat jen v podobě základních formulářů určených k testování.

TIC server - software budu implementovat se zaměřením na klienta AICCU a tunelování pomocí protokolu AYIYA. Software bude podporovat šifrování komunikace pomocí TLS.

Tunel server - stejně jako pro server TIC je i u tunel serveru důležitá metoda tunelování AYIYA popsaná v kapitole 2.2. Protože protokol AYIYA je nestavový, software si bude muset uchovávat informace o stavu tunelu.

Inicializaci tunelu zahajuje vždy klient, pro server to tedy znamená přijetí zprávy AYIYA s operačním kódem Heartbeat na naslouchajícím soketu UDP.

Start tunelu znamená vytvořit zařízení TUN, nastavit mu adresu, MTU a případně vytvořit systémové směrování prefixů patřící danému tunelu.

Server musí vědět, jak dlouho z tunelu nepřišly žádné zprávy, a nečinný tunel uzavřít.

Provoz směrem do tunelu software bude načítat ze souborového deskriptoru zařízení TUN, zapouzdřovat do protokolu AYIYA a posílat danému klientovi. Pakety tekoucí opačný směrem bude software číst z soketu UDP, ověřovat správnost hlavičky AYIYA a identitu odesilatele. Dále je bude zapisovat do souborového deskriptoru příslušného rozhraní TUN.

O případnou fragmentaci IPv6 paketů či zasílání ICMPv6 zpráv se bude starat operační systém tunel serveru.

Shrnutí

Kapitola se zabývala návrhem tunel broker systému. Rozepsal jsem jednotlivé části systému a určil základní datové schéma, které bude systém využívat. Na konci jsem v textu shrnul informace, ze kterých vychází implementace systému.

Kapitola 4

Implementace

V této kapitole bych rád rozebral detaily implementace všech mnou vytvořených částí systému, tedy server služby protokolu TIC, tunelovací software protokolu AYIYA, napojení komponent na společnou databázi, další pomocné skripty a programy potřebné pro jednoduchou práci se systémem. Tunelovací AYIYA server rozeberu v rámci několika částí, včetně popisu, co se všechno děje při práci s pakety, údržbě tunelu a komunikaci s ostatními částmi systému.

4.1 Server TIC

Server TIC je klasická služba typu klient–server. Server implementuje stejnojmenný protokol, který je popsán v části 2.3. Hlavní výhodou proti implementaci společnosti SixXS je podpora šifrování komunikace pomocí asymetrické kryptografie. Protože se TIC protokolem přenášejí citlivá data, je šifrování důležité.

4.1.1 Konfigurační volby

Binární verze softwaru se jmenuje `ticd` a ke svému korektnímu spuštění potřebuje předat parametr `-c konfigurační_soubor`. Konfigurační soubor je složen z direktiv, jejichž parametry mohou být třech různých typů. Typ *int_t* znamená, že parametrem direktivy je celé číslo. Dalším typem je *string_t*, a označuje parametry typu řetězec bez bílých znaků. Posledním typem je *line_t*, který za parametr direktivy považuje všechny znaky do konce daného řádku.

Následuje výpis direktiv, které `ticd` podporuje:

- **welcome**: Direktiva typu *line_t* obsahuje řetězec, který server použije jako úvodní zprávu protokolu TIC.
- **port**: Direktiva typu *int_t*. Slouží k určení portu TCP, na kterém bude služba naslouchat. Parametr má výchozí hodnotu 3874.
- **tls_support**: Direktiva typu *string_t*. Parametrem může být pouze hodnota „yes“ nebo „no“. Direktiva zapíná podporu vrstvy TLS. Výchozí hodnota je „no“. Pokud je TLS zapnuto, musí být nastaveny direktivy **cacert_file**, **cert_file**, **key_file** a nepovinně **crl_file**.
- **cacert_file**: Direktiva typu *string_t*. Určuje cestu k souboru obsahující certifikát certifikační autority.

- **cert_file**: Direktiva typu *string_t*. Určuje cestu k souboru obsahující certifikát programu.
- **key_file**: Direktiva typu *string_t*. Určuje cestu k souboru obsahující privátní klíč certifikátu. Soubor by měl být měl mít nastavené oprávnění pouze pro svého vlastníka.
- **mysql_server**: Direktiva typu *string_t*. Obsahuje IP adresu, nebo DNS název MySQL serveru.
- **mysql_port**: Direktiva typu *int_t*. Obsahuje port, na kterém MySQL server naslouchá. Výchozí hodnota je 3306.
- **mysql_user**, **mysql_password**: Direktivy typu *string_t*. Obsahují uživatelské jméno a heslo MySQL účtu.
- **mysql_database**: Direktiva typu *string_t*. Obsahuje jméno databáze, která obsahuje tabulky systému.

Druhým možným parametrem programu je `--version`, který slouží ke získání aktuální zkompilované verze programu.

4.1.2 GnuTLS

GnuTLS je projekt, který vyvíjí knihovnu poskytující bezpečnou vrstvu nad spolehlivou transportní vrstvou. V současnosti knihovna GnuTLS implementuje standardy navržené pracovní skupinou IETF[1] a je popsána v [3].

V programu je tato knihovna využita pro šifrování komunikace mezi klientem a serverem služby TIC. Komunikace je z počátku nešifrovaná, tedy v otevřené textové podobě. Pomocí poslání příkazu `startls` klientem se obě strany domluví na šifrování. Veškerá další komunikace probíhá přes zabezpečený kanál.

Klienti nemají své certifikáty, kterými by se autorizovali serveru a neznají ani certifikát certifikační autority, pomocí kterého by šlo ověřit identitu serveru.

Možnosti útoku na TIC službu jsou rozepsány v kapitole 5.

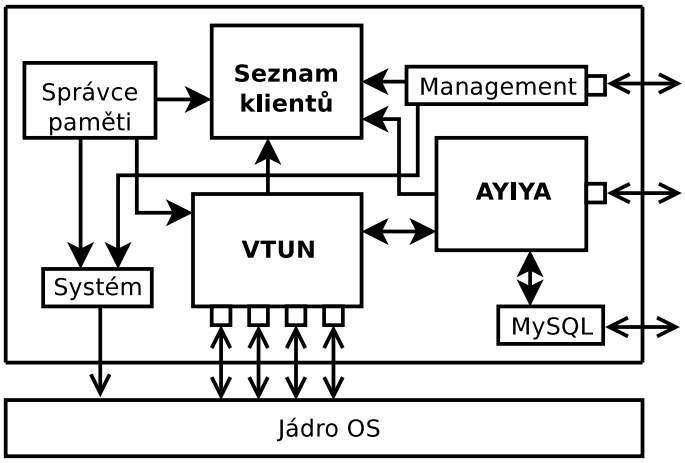
4.1.3 OpenSSL

Knihovnu OpenSSL v programech využívám jako zdroj funkcí počítajících kontrolní součty. Server TIC využívá kontrolní součet k ověření identity uživatele. Povoleny jsou metody MD5 a SHA1. Server AYIYA používá ke kontrolním součtu metodu SHA1, protože metoda MD5 není považována za bezpečnou. Knihovna OpenSSL nabízí následující funkce:

```
unsigned char *MD5(const unsigned char *d, unsigned long n,
                   unsigned char *md);

unsigned char *SHA1(const unsigned char *d, unsigned long n,
                    unsigned char *md);
```

[1] IETF je zkratka anglického názvu Internet Engineering Task Force, což je mezinárodní otevřená komunita síťových návrhářů, výrobců a výzkumníků zainteresovaných ve vývoji internetové architektury a hladkém běhu internetové sítě.



Obrázek 4.1: Blokové schéma tunelovacího AYIYA serveru.

První parametr funkce obsahuje ukazatel na začátek dat, nad kterým má být kontrolní součet proveden. Druhý parametr obsahuje délku dat v bajtech. Třetí parametr funkce se používá jako místo pro uložení výsledku. Pokud se jako třetí parametr uvede hodnota `NULL`, použije funkce svůj statický prostor pro výsledek. Ve vícevláknovém prostředí je ovšem nutné, aby každé vlákno mělo svůj vlastní prostor pro výsledek. Návratová hodnota obsahuje ukazatel na výsledný kontrolní součet.

4.2 Server AYIYA

Tunelovací server, jak bylo napsáno v části 3.3, je směrovač, který poskytuje tunelovanou konektivitu IPv6 klientům připojených přes IPv4 protokol. Základní princip činnosti serveru je obsluhovat pakety přicházející zapouzdřené protokolem AYIYA od klientů a také vytvářet pakety AYIYA, které putují zpět ke klientům. Blokové schéma programu je na obrázku 4.1. Modul AYIYA se stará o přijímání a odesílání tunelovaných paketů klientům. Modul VTUN se stará o komunikaci s jádrem OS, kterému se předávají pakety ke směrování. Informace o všech aktivních tunelech se udržuje v seznamu klientů. Správce paměti vyřazuje neaktivní tunely a modul Managemet slouží k předávání aktuálních změn parametrů aktivních tunelů. Modul MySQL zpracovává komunikaci s centrální databází všech klientů.

4.2.1 Konfigurační volby

Stejně jako u serveru TIC i server AYIYA po zkompilování vytváří binární soubor `tunserv`, který ke korektnímu spuštění vyžaduje jeden parametr `-c konfigurační_soubor`. V konfiguračním souboru jsou obsaženy všechny potřebné volby pro nastavení programu.

Následuje výpis direktiv, které `tunserv` podporuje:

- **read_threads**: Direktiva typu *int_t*. Obsahuje počet vláken, které bude společně číst UDP soket. Výchozí hodnota je 1.
- **port**: Direktiva typu *int_t*. Slouží k určení portu UDP, na kterém bude server AYIYA naslouchat. Parametr má výchozí hodnotu 5072.

- **pop_name**: Direktiva typu *string_t*. Určuje jméno tunelovacího serveru v rámci celého systému. Musí korespondovat s pojmenováním tunelovacího serveru v databázi.
- **client_timeout**: Direktiva typu *int_t*. Určuje čas v sekundách, po kterém se tunel při neaktivitě může ukončit. Výchozí hodnota je 120.
- **mem_thread_sleep**: Direktiva typu *int_t*. Určuje časový interval v sekundách, ve kterém probíhá údržba neaktivních tunelů. Výchozí hodnota je 60.
- **management_port**: Direktiva typu *int_t*. Obsahuje číslo portu TCP, na kterém bude naslouchat řídící modul. Výchozí hodnota je 1234.
- **mysql_server**: Direktiva typu *string_t*. Obsahuje IP adresu, nebo DNS název serveru MySQL.
- **mysql_port**: Direktiva typu *int_t*. Obsahuje port, na kterém server MySQL naslouchá. Výchozí hodnota je 3306.
- **mysql_user**, **mysql_password**: Direktivy typu *string_t*. Obsahují uživatelské jméno a heslo k účtu MySQL.
- **mysql_database**: Direktiva typu *string_t*. Obsahuje název databáze, která obsahuje tabulky systému.

4.2.2 Modul MySQL

Modul MySQL slouží k zajištění komunikace mezi softwarem a stejnojmennou databází. Modul obsahuje několik funkcí, které zapouzdřují standardní rozhraní MySQL a definice dotazů SQL. Tyto dotazy jsou použity ve funkcích, které se využívají pro získání informací z databáze.

Hlavní místo, kde jsou uloženy všechny permanentní data z celého systému, je právě databáze. Z databáze čerpají informace pomocí tohoto modulu tunelovací server `tunserv` i server TIC `ticd`. To zaručuje aktuálnost dat v celém systému.

Modul se umí vyrovnat i s chvilkovým výpadkem služby MySQL nebo i přepojením databáze. U každého dotazu kontroluje dostupnost databázové služby a v případě, kdy služba není dostupná, se ji snaží znovu navázat.

Zdrojový kód tohoto modulu obsahuje řadu funkcí, které z databáze zjišťují určitou hodnotu podle zadaných kritérií. Pokud je potřeba z databáze zjistit neznámé množství dat, jako například několik řádků tabulky, modul nabízí rozhraní, kde samotný výsledek dotazu nezpracovávají funkce modulu, ale předávají jej volající funkci. Zpracování takových dotazů by bylo možné realizovat i interně v rámci modulu, ale docházelo byt tak ke dvojímu kopírování dat v paměti, což by znamenalo zpomalení běhu aplikace.

S vícevláknovým prostředím se modul MySQL vyrovnává chráněním svých proměnných mutexy. To zajišťuje, že pouze jedno vlákno může vykonávat dotaz MySQL v jednom čase. Tento přístup je možný, protože vyhledávání dat v databázi je používané velmi málo vzhledem k celkovému běhu programu.

4.2.3 Virtuální rozhraní VTUN

Funkce směrování paketů z rozhraní na rozhraní jsou naimplementovány v jádře systému Linux. Aby server AYIYA běžící v uživatelském prostoru „*user space*“ mohl zprostředkovávat

komunikaci klientů, musel jsem použít rozhraní mezi jaderným a uživatelským prostorem. Toto rozhraní se jmenuje VTUN a jeho dokumentace je k nalezení ve zdroji [7].

Virtuální rozhraní se operačnímu systému jeví jako jakékoli jiné síťové zařízení s tím rozdílem, že místo fyzické síťové karty jsou data zapisována nebo čtena z uživatelského prostoru.

Program běžící v uživatelském prostoru požádá jádro o vytvoření standardního popisovače funkcí `open()` na speciální soubor */dev/net/tun*. Tomuto popisovači lze nastavit funkcí `ioctl()` parametry jako například jméno rozhraní. Software serveru AYIYA vytváří jméno virtuálních rozhraní spojením prefixu „tun“ a jednoznačného databázového identifikátoru tunelu v podobě kladného čísla. Software nastaví IPv6 adresu již pojmenovaného rozhraní voláním systémového příkazu `ip -6 address add` s příslušnými parametry. Obdobně program nastaví i systémové směrování příkazem `ip -6 route add` a MTU pomocí `ip link set mtu`.

Operace čtení a zápisu

V programu používám pro čtení a zápis tunelovacího rozhraní funkce standardu POSIX, `readv()` a `writev()`. Načítací funkce se od své jednodušší varianty liší tím, že načítaná data dokáže uložit do nesouvislé paměti. Funkce k popisu nesouvislé paměti využívá pole struktur *struct iovec*, kde každá obsahuje ukazatel někam do paměti a počet bajtů, které lze od adresy zapsat. Obdobně je tomu i u funkce pro zápis, která je schopná z několika míst v paměti zapisovat data do popisovače souboru, tedy virtuálního rozhraní. Obě funkce jako svůj parametr obsahují pole struktur *iovec* a velikost tohoto pole.

Vlastnosti funkcí `readv()` a `writev()` potřebuji pro korektní zaslání zprávy jádru systému. Jádro potřebuje znát typ obsahu zasílané zprávy, tedy zda se jedná o IPv4 paket, IPv6 paket, nebo jde například o rámec linkové vrstvy. Tento údaj jádro systému hledá na začátku předávaných dat, kde je umístěna struktura *struct tun_pi*, která říká, jaká hlavička v datech následuje. Názorný kód používající funkci `writev()` naleznete v příloze A.1.

4.2.4 Organizace paměti programu

Tunelovací server AYIYA potřebuje ke svému korektnímu běhu znát informace o svých klientech. Komunikace mezi klientem a serverem je realizována pomocí protokolu UDP, který nevytváří stavové spojení jak, to dělá protokol TCP. Z tohoto důvodu si musí server o klientovi udržovat svůj interní záznam o stavu a aktivitě klienta. Mimo stavu si musí server udržovat informace, na jaké IP adrese a portu UDP klient naslouchá, kdy naposledy komunikoval, a jaký souborový popisovač používá pro předávání paketů operačnímu systému.

Data, která server o klientech potřebuje znát (např. identifikátor klienta AYIYA), jsou uloženy v databázi. Aby server nemusel pro každou tunelovanou zprávu hledat data v databázi, vytváří si pro každého klienta datovou strukturu. Některé důležité prvky struktury naleznete na obrázku 4.2. Prvek `reading_thread` je typu *pthread_t* a určuje vlákno, které obsluhuje čtení z virtuálního rozhraní. Položka `addr` je struktura *sockaddr_in* a obsahuje IPv4 adresu a port, na kterém klient naslouchá. Další v řadě jsou `server_identity` a `client_identity`. Tyto prvky obsahují IPv6 adresy serveru a klienta, které se používají k identifikaci komunikující protistrany. Stejně tak server musí znát sdílený tajný klíč, lépe řečeno kontrolní součet tajného klíče, pomocí kterého se ověřuje integrita přijatých zpráv a vypočítává se kontrolní součet zasílaných zpráv.

```
typedef struct client {
    pthread_t reading_thread;
    struct sockaddr_in  addr;
    struct in6_addr  server_identity;
    struct in6_addr  client_identity;
    sha_hash signature;
    int prefix_len;
    int tunnel_id;
    time_t last_active_time;
    int tun_fd;
    int ayiya_fd;
} client_t;
```

Obrázek 4.2: Datová struktura popisující klienta.

Struktury popisující jednotlivé klienty program uspořádává do jednosměrně vázaného seznamu, s nímž provádí operace přidání, odebrání, vyhledávání, atd. Operace nad lineárním seznamem jsou zabezpečeny pro použití ve vícevláknovém prostředí.

Synchronizace přístupu pomocí modelu čtenáři – písaři

Seznam struktur se záznamy o klientech je chráněn systémem mutexů, které řeší problém tzn. „čtenářů a písařů“. Je to problém, kdy nad společnou proměnnou pracují dvě množiny procesů, jedni proměnnou čtou a druzí do ní zapisují. Počet operací čtení je výrazně vyšší než operace zápisu. Aby nedocházelo u proměnné ke kolizím, je možné v jedné chvíli buď zapisovat jedním písařem, nebo číst i více čtenáři současně.

Do systému jsem naimplementoval variantu s preferováním písařů, která zajišťuje písařům jistý přístup k proměnné i při zahlcení čtenáři. Operace vyhledávání v seznamu je právě operací čtení a operace přidání a odebrání znamenají změny v seznamu. Tento přístup umožňuje, aby sdílenou proměnnou četlo více vláken současně a operace vytváření a mazání, které nejsou příliš časté, zbytečně dlouho neotálely s provedením.

4.2.5 Vícevláknové prostředí

Tunelovací server musí umět obsluhovat souběžně více klientů, kteří chtějí využívat jeho služby. Z protokolu AYIYA, popsaného v části 2.2, vyplývá, že tunelovací server musí poslouchat na definovaném portu UDP. Protokol posílá datagramy UDP, které obsahují tunelované zprávy. Pokud by se o přicházející zprávy starala pouze jedna výpočetní jednotka, tedy jedno vlákno nebo jeden proces, další datagramy by byly obslouženy až po kompletním vyřízení prvního datagramu. Program pro svou paralelizaci používá vlákna, která vytváří pomocí knihovny `pthread.h`. Konfigurační parametr *read_threads* určuje, kolik vláken bude současně číst tunelované pakety z soketu UDP. Záleží na administrátorovi, kolik současných čtecích vláken zvolí.

Každému aktivnímu tunelu program vytváří virtuální rozhraní pro komunikaci s jádrem operačního systému. Zápisem do virtuálního rozhraní lze systému předat paket bez čekání. O čtení každého virtuálního rozhraní se stará vlákno, které se vytvoří při vzniku tunelu a zaniká s koncem tunelu. Se zvyšujícím se počtem tunelů se zvyšuje i počet vláken. Protože pakety na virtuální rozhraní nechodí z pohledu procesu příliš často a čtení z rozhraní je

blokující operace, budou čtecí vlákna většinu výpočetního času v čekajícím stavu a nebudou tedy zbytečně zatěžovat plánovač procesů operačního systému.

4.2.6 Zpracování paketů

V této části bych rád popsal, při jaké události dojde k virtuálnímu sestavení tunelu, co se přesně stane, když přijde zabalený IPv6 paket tunelem a naopak co se děje, když míří odpověď směrem do tunelu. Jakým způsobem je kontrolována příchozí hlavička AYIYA a jak se při odesílání hlavička AYIYA vytváří.

První paket

Tunel vždy inicializuje klient. Je to způsobeno tím, že server nezná IP adresu a port UDP klienta. V prostředí IPv4 Internetu, kde se vyskytují technologie NAT, je možné pouze jednosměrná inicializace datových spojení.

Klient po své inicializaci, tedy obdržení údajů z serveru TIC, vytvoří virtuální rozhraní TUN a odešle první tři zprávy s operačním kódem *heartbeat*. Operační kód *heartbeat* se používá pro režijní zprávy, které nenesou žádný další obsah a nejsou směrovány. Množství zpráv vychází z fungování protokolu UDP, který negarantuje doručení zpráv. Vyšším počtem zaslaných zpráv se zvyšuje pravděpodobnost, že protější strana některou ze zpráv obdrží.

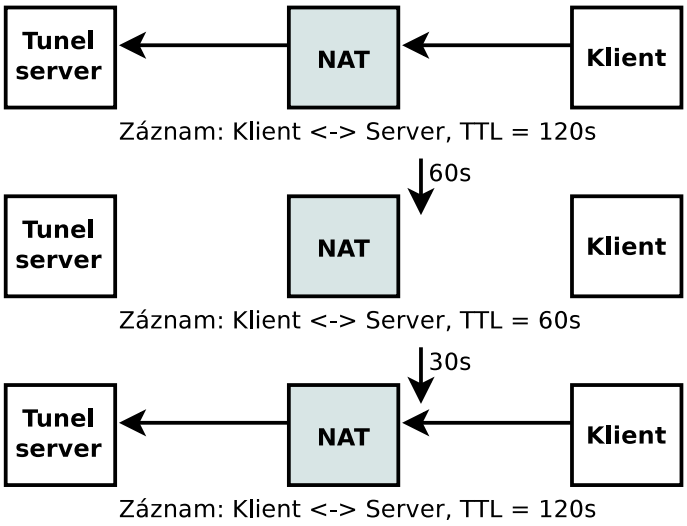
Na straně serveru odbavuje příchozí pakety několik vláken. Pokud paket obsahuje validní hlavičku AYIYA, obslužné vlákno zkontroluje, zda již zná identitu klienta tím, že porovná obsah položky hlavičky *Identity* se všemi svými známými klienty, které má v paměti. Pokud položku nenalezne, jedná se patrně o nového klienta a je nutno jeho identitu ověřit v centrální databázi. Najde–li program v databázi identitu klienta, načte z ní i tajné sdílené heslo, IPv6 adresu, délku prefixu, MTU tunelu a databázové ID tunelu. Pomocí tajného sdíleného hesla ověří pravost paketu a pokusí se vytvořit virtuální rozhraní TUN, jak bylo popsáno v části 4.2.3.

Režijní zprávy

Protokol UDP nevytváří trvalé spojení mezi klientem a serverem, ale pouze při procházení možnou technologií překladu adres NAT vytváří záznam pro směrování paketů putujících sítí v přesně opačném směru. Tedy pro odpovědi. Záznamy v NAT mají omezenou platnost a jsou časem vymazány. Pokud by byl tunel delší dobu neaktivní (neproudily by přes něj žádá data), mohl by se překlad adres rozpadnout a příchozí zprávy pro klienta v rámci tunelované konektivity by nebyly klientovi doručeny.

Takovému chování zabraňuje použití režijních zpráv typu *heartbeat*. Tyto zprávy klient posílá serveru v pravidelných intervalech nezávisle na datovém provozu v tunelu. Zajišťuje tím udržování záznamů v tabulkách NAT po cestě k serveru. Celý princip je znázorněn na obrázku 4.3, kde je zprovozněn tunel, přes který neproudí žádná data. Na obrázku je překlad adres nastaven tak, aby záznamy udržoval 120 sekund. Po 90 sekundách nečinnosti klient posílá *heartbeat* zprávu a tím aktualizuje překladový záznam.

Pokud by se přece jen adresa klienta změnila, například změnou překladu adres, server si aktualizuje odpovídající adresu klienta hned s první příchozí zprávou od klienta. Server nerozlišuje klienty dle příchozí adresy, ale podle položky *Identity* v hlavičce AYIYA. Adresu klienta pak vždy porovná s minulou zapamatovanou adresou a případně provede její



Obrázek 4.3: Vliv režijních zpráv na překlad adres (NAT).

aktualizaci. Děje se tak i v případě, kdy klient tunelem neodesílá žádná užitečná data, ale odesílá jen režijní zprávy.

Pakety od klienta do Internetu

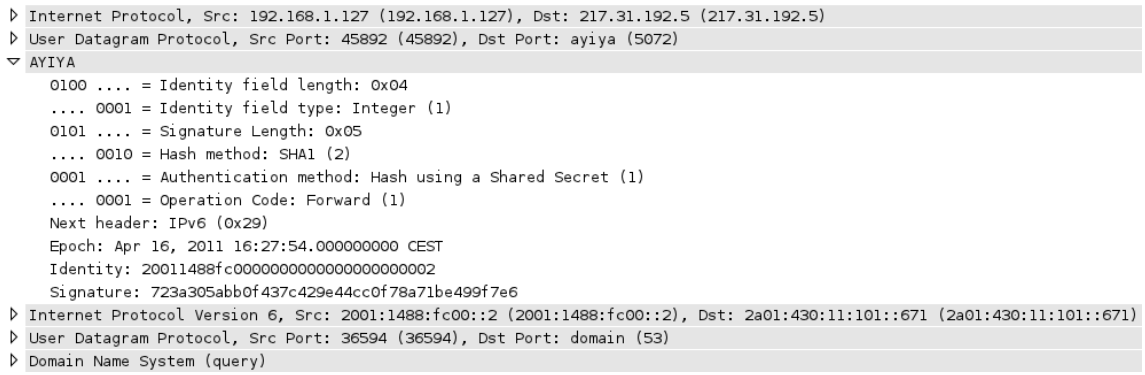
Zprávy, které jsou určeny k průchodu tunelovacím serverem, tedy k rozbalení a přesměrování, jsou označeny v hlavičce AYIYA operačním kódem *forward* nebo *echo request and forward*. V obou případech dojde k přesměrování tunelovaného paketu, ale v případě *echo request and forward* se paket musí ještě odeslat zpět klientovi jako potvrzení, že byl opravdu doručen a přesměrován. Implementace klienta AICCU používá výhradně pouze operační kód *forward*, a proto možnost vrácení obsahu paketu zpět klientovi není v tunelovacím programu implementována.

Typický příklad paketu, který míří od klienta k serveru, je ukázán na obrázku 4.4. Z obrázku je vidět poskládání jednotlivých protokolových vrstev. Fyzickou a linkovou vrstvu obrázek neobsahuje. První viditelná vrstva IPv4 je nosným síťovým protokolem celého paketu. Na ni navazuje protokol UDP, jenž tvoří transportní vrstvu a obsahuje hlavičku AYIYA, která je na obrázku zobrazena podrobně. Jednotlivé položky AYIYA protokolu jsou popsány v části 2.2.1. Klient, kterému paket náleží, má IPv6 adresu shodnou s hodnotou pole *identity*. Protože se jedná o tunelovaný paket, je operační kód nastaven na hodnotu *forward*. Jak je vidět, obsahem tunelovaného paketu je IPv6 paket, který přenáší dotaz protokolu DNS.

Pro úspěšné přesměrování tunelované zprávy musí paket vyhovět různým podmínkám. První test, který tunelovací software provádí po načtení dat z naslouchajícího soketu UDP je kontrola délky načtených dat. Pokud je délka přijatých dat menší než je minimální velikost hlavičky AYIYA, je jasné, že se nemůže jednat o validní tunelovanou ani režijní zprávu, a paket je zahozen.

Druhý test přijatých dat zjistí, zda jeho položky vyhovují implementovaným funkcím tunelovacího programu. Ten podporuje všechny zprávy odesílané klientem AICCU. Konkrétně otestuje délku a metodu pole kontrolního součtu, operační kód a metodu autentizace.

Dále tunelovací program v hlavičce AYIYA ověřuje, zda zná identitu odesilatele. Identitu



Obrázek 4.4: Struktura AYIYA tunelované zprávy.

může nalézt na více různých místech – v paměti programu nebo v centrální databázi. Rychlost zpracování zprávy závisí na rychlosti vyhledání identity. Proto se nejprve vyhledává v paměti programu, kde jsou umístěny identity všech aktivních klientů. Pokud identita není nalezena v paměti programu, musí program vznést dotaz na centrální databázi, která obsahuje identity všech registrovaných tunelů. Obsahuje-li databáze hledanou identitu, je vytvořeno nové virtuální rozhraní a zpracování zprávy pokračuje.

Ve chvíli, kdy je známá identita odesilatele, musí program ověřit integritu zprávy. K tomu potřebuje znát sdílené tajemství, pomocí kterého vypočítá kontrolní součet zprávy a porovná jej s přijatým polem *Signature*. V případě shody je zpráva považována za ověřenou a je dále zpracovávána. Jestliže se kontrolní součty neshodují, je zpráva zahozena.

Poslední kontrolu, kterou musí zpráva podstoupit, je kontrola času. Ta porovná položku hlavičky *epochtime* s aktuálním časem. V případě, že se časy rozcházejí o více než 120 sekund, je paket zahozen.

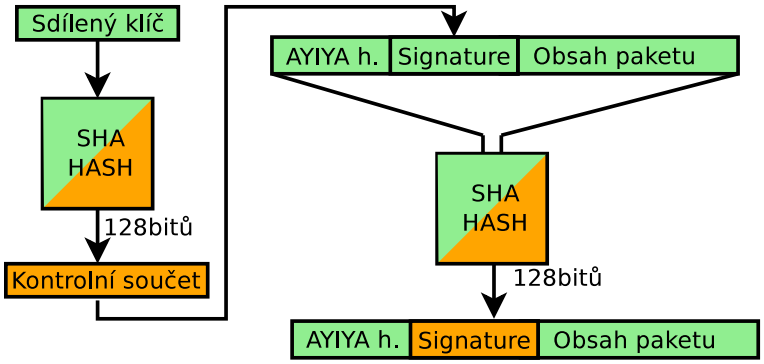
Nyní je zpráva připravena k vybalení a odeslání jádru operačního systému k přesměrování. Protože program zná identitu klienta, porovná zdrojovou IPv4 adresu a UDP port příchozí zprávy se svým záznamem v paměti a v případě neshody provede aktualizaci svého záznamu. Takto se udržuje stále aktuální vazba na IPv4 adresu klienta.

Rychlost zpracování zprávy již navázaného tunelu, kdy záznam o klientovi je uložen v paměti programu, se odvíjí od rychlosti procházení jednosměrného seznamu klientů v paměti. V současné verzi jsem vyhledávání naimplementoval postupným procházením seznamu. Rychlost vyhledávání je dostatečná, protože program předpokládá počet aktivních klientů v řádech stovek až několika málo tisíců.

Pakety z Internetu pro klienta

Značně jednodušší situace nastává ve chvíli, kdy program přečte zprávu putující směrem ke klientovi. Jádro operačního systému má k dispozici směrovací tabulku, podle které ví, jak pakety směrovat. S každým novým aktivním tunelem se do směrovací tabulky přidají potřebné záznamy pro prefix na tunelovém rozhraní a staticky směrované prefixy na druhý konec tunelu.

Pro každý tunel, potažmo virtuální rozhraní, má program svůj vlastní popisovač souborů, přes který komunikuje s jádrem operačního systému. Načítání dat z popisovače obstarává funkce `tun_read()`, která běží jako obslužná funkce vlákna pro každý aktivní tunel. Podle toho, ze kterého popisovače data načte, ví přesně, do kterého tunelu směřují. Takto



Obrázek 4.5: Proces podepisování AYIYA paketu.

načtená data jsou předána funkci `ayiya_write()`, kde jsou opatřena hlavičkou AYIYA. V hlavičce se vyplní základní údaje o délce pole identity, její typ, typ kontrolního součtu, identita odesílatele a vypočítá se kontrolní součet s použitím tajného hesla.

Vytvoření podpisu paketu je znázorněno na obrázku 4.5. Sdílený klíč může mít libovolnou velikost. Pomocí metody kontrolního součtu SHA1 bude jeho otisk o přesné délce 128 bitů. Tento otisk se umístí do pole *Signature* AYIYA hlavičky. Celý paket poté znovu projde kontrolním součtem a jeho výsledek bude přepsán od pole *Signature*, takto podepsaný paket bude odeslán přes standardní soketové rozhraní na poslední známou IPv4 adresu a port.

4.2.7 Ukončení tunelu

Udržovat informace o neaktivních tunelech je pro tunelovací software přítěží. Proto program obsahuje speciální vlákno, které v daném časovém intervalu prochází celý seznam tunelů v paměti programu a kontroluje, zda nejsou splněny podmínky pro zrušení a vymazání tunelu z paměti. Hlavní podmínkou živosti tunelu je komunikativnost protější strany. Klient AICCU posílá v pravidelných intervalech režijní *heartbeat* zprávy nezávisle na datovém provozu. Tunelovací server má ve struktuře pro každého klienta proměnnou `last_active_time` typu *time_t.*, ve které udržuje čas poslední aktivity klienta. Za aktivitu je považována jakákoli validní příchozí zpráva od klienta. Výše zmíněné vlákno kontroluje parametr `last_active_time` a pokud je tunel neaktivní po určitou dobu, kterou lze nastavit v konfiguračním souboru, vlákno tunel ukončuje.

Tunelovací server je složen z více kooperujících vláken. Každý tunel má své vlákno, které čte příchozí data od jádra operačního systému z popisovače souboru. Pokud se má tunel korektně ukončit, je potřeba přerušit blokující čtení vlákna, uzavřít popisovač souborů a vymazat datovou strukturu klienta v paměti programu. Činnost ukončení tunelu inicializuje vlákno, jež udržuje paměť programu. Toto vlákno vyjme datovou strukturu klienta ze seznamu, nastaví mu ukončovací příznak, pomocí systémových příkazů `ip link set down` a `ip link delete` smaže virtuální rozhraní ze systému a začne čekat na signál čtecího vlákna signalizující jeho konečnou fázi. Čtecí vlákno, které se nachází v blokujícím čtení popisovače souborů, dostane po zrušení rozhraní chybový stav `Input/output error` a vynoří se ze čtení. K dalšímu opakování čtení nedojde, protože se čtecí cyklus ukončí pomocí nastaveného ukončovacího příznaku. Čtecí vlákno uzavře popisovač souborů, vyšle signál a ukončí se. Udržovací vlákno obdrží signál, převezme návratovou hodnotu ukončeného

čtecího vlákna a vymaže datovou strukturu klienta z paměti programu.

4.2.8 Management rozhraní

Tunelovací software disponuje i rozhraním pro správu tunelů. Toto rozhraní se využívá hlavně z registračního portálu, kde lze tunelům měnit MTU a měnit jejich stav z povoleného na zakázaný a naopak. Tyto změny se zaznamenají do centrální databáze, odkud se použijí při nově navázaném tunelu pro jeho konfiguraci. Co ale dělat v případě, kdy je tunel na serveru aktivní a jeho parametry se změní? Použitím Management rozhraní lze měnit konfiguraci aktivních tunelů. Další vlastnost, kterou managementovací rozhraní podporuje, je možnost změnit IPv6 směrování pro aktivní tunely.

Jedná se o jednoduchou službu typu klient-server založenou na protokolu TCP. Služba implementuje speciální aplikační protokol, který zavádí následující aplikační zprávy:

```
subnet_add    tunnel_id prefix prefix_len
subnet_del    tunnel_id prefix prefix_len
mtu_change    tunnel_id
status_change tunnel_id
```

Příkazem `subnet_add` nebo `subnet_del` se přidá nebo ubere tunelu s *tunnel_id* specifikovaný IPv6 prefix. Příkaz `mtu_change` slouží k oznámení změny MTU tunelu. Program po obdržení této zprávy načte aktuální hodnotu MTU z databáze a v případě, že současná a nová hodnota nejsou stejné, provede změnu přímo v systémové konfiguraci pomocí systémového příkazu `ip`. Poslední z aplikačních příkazů `status_change` slouží k aktualizování stavu tunelu.

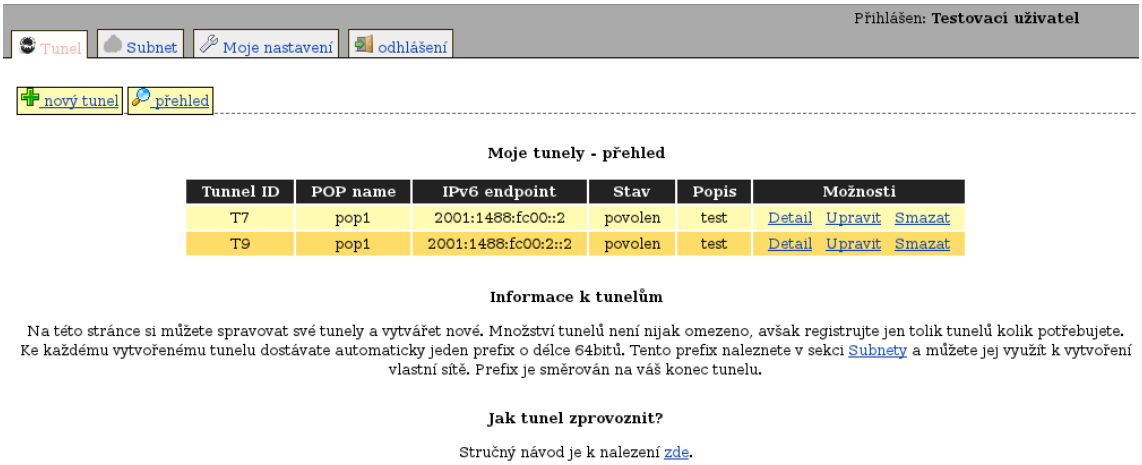
Příkazy managementovacího rozhraní se provádějí pouze v případě, že je specifikovaný tunel aktivní. Například u příkazu `status_change` je zbytečné vykonávat změnu ze zakázaného stavu na povolený, protože zakázaný tunel nemůže být aktivní. Naopak v případě, kdy se tunel příkazem zakazuje, tedy načítá se z databáze zakázaný stav, se aktivnímu tunelu nastaví příznak zakázaného stavu. Takový tunel bude ukončen s příští iterací vlákna, které maže neaktivní nebo takto zakázané tunely.

4.3 Registrační portál

Registrační portál je tvořen webovým rozhraním, které je napsáno v jazyce PHP. Portál umožňuje registraci nových uživatelů a přihlášení zaregistrovaných. Po přihlášení si mohou uživatelé spravovat své tunely a jim přidělené prefixy, uživatele si také mohou změnit své kontaktní údaje a heslo k registračnímu portálu.

4.3.1 Prostředí portálu

Webové rozhraní nabízí uživateli základní možnost vybrat si ze záložek *Tunel*, *Subnet*, *Moje nastavení* a *odhlášení*. Dále je v pravém horním rohu vidět identita přihlášeného uživatele. Jména záložek souvisí s činnostmi, které mohou v systému provádět. Záložka *Tunel* ukazuje přehled registrovaných tunelů. Ke každému tunelu nabízí možnost podívat se na jeho detaily, upravit nastavení tunelu a tunel smazat. Pomocí tlačítka *nový tunel* se uživateli zobrazí formulář, jehož vyplněním lze vytvořit v systému nový tunel. Obrázek 4.6 ukazuje výřez záložky *Tunel*.



Obrázek 4.6: Část registračního portálu – sekce *Tunel*.

Tabulka přehledu tunelů zobrazuje na každém řádku informace o jednom tunelu. *ID tunelu* je spojeno s databázovým ID daného tunelu. *POP name* je označení asociovaného tunelovacího serveru. Na tunelovací server jsou pevně vázány IPv6 adresy a nelze jej po dobu trvání tunelu změnit. Sloupec *IPv6 endpoint* zobrazuje IPv6 adresu klientova konce tunelu. Další položky jako *Stav* a *Popis* odpovídají svému pojmenování. Dále si lze o každém tunelu zobrazit podrobnosti, ve kterých se uživatel dozví uživatelské jméno, sdílené heslo a adresu serveru TIC. Každému tunelu uživatel může změnit nastavení, jako např. MTU tunelu.

Záložka *Subnet* obsahuje podobnou strukturu jako *Tunel*, ale zobrazuje informace o IPv6 prefixech přidělených jednotlivým vytvořeným tunelům. Každý tunel při svém vytvoření dostává automaticky nasměrován jeden IPv6 prefix o délce 64 bitů. Pokud by uživatel chtěl na konci tunelu vytvářet více sítí, může si přes záložku *Subnet* přiřadit každému tunelu navíc jeden IPv6 prefix o délce 56 bitů. Takto uživatel získá prostor 8 bitů pro vytváření své IPv6 podsítě, což znamená, že může vytvořit až 256 nezávislých IPv6 sítí o délce prefixu 64 bitů.

4.3.2 Činnosti na pozadí portálu

Údaje o uživatelích, tunelech a prefixech se ukládají do centrální databáze, která shromažďuje veškeré informace. Z této databáze berou data všechny části systému. Od serveru TIC, přes tunelovací server až po webový portál. Druhým způsobem, kterým webové rozhraní komunikuje s tunelovacími servery, je jejich managementovací rozhraní.

Vytvoření tunelu a přidělení prefixu

Než je uživatel schopen vytvořit tunel, musí provést proces registrace, což obnáší vyplnění jednoduchého formuláře a kontroly, zda již zadaný login neexistuje.

Vytvoření tunelu probíhá mírně komplikovanějším způsobem. Systém musí prvně prověřit, zda má pro vytvoření nového tunelu dostupné zdroje. Za zdroje se považují záznamy v tabulce `ipv6_space`, které nemají nastavenu asociaci s již existujícím tunelem a jejich atribut *type* má hodnotu 1. Hodnota signalizuje, že tento prefix je určen pro propojovací

síť virtuálního tunelu. Jsou-li zdroje dostupné, PHP skript pokračuje vytvořením záznamu v tabulce `tunnel` a asociací jednoho ze zdrojů s nově vzniklým tunelem. Systém ke každému vzniklému tunelu automaticky přiděluje prefix o délce 64 bitů. Záznamy v tabulce `ipv6_space` určeny k tomuto účelu jsou označeny hodnotou atributu *type* 2. Opět skript provádí kontrolu na dostatek zdrojů a vytváří nový záznam v tabulce *subnet* asociovaný cizím klíčem tunelu. Pokud zdroje nejsou dostupné, skript ruší veškeré změny, které provedl a vrací uživateli chybu.

Systém umožňuje každému tunelu přidělit ještě jeden větší IPv6 prefix o délce 56 bitů. Formulář pro přiřazení prefixu lze vyvolat pomocí tlačítka *nový subnet*. Uživatel si vybere jeden ze svých tunelů a odesláním formuláře nastartuje proces přiřazování. Proces opět zkontroluje dostupnost zdrojů, tentokrát s hodnotou atributu *type* rovnou třem, vytvoří záznam v tabulce subnetů asociovaný s vybraným tunelem a kontaktuje managementovací rozhraní příslušného tunelovacího serveru, aby mu směrování nového IPv6 prefixu oznámil.

Změna parametrů tunelu

Webový portál umožňuje uživatelům měnit nastavení tunelů. Například při změně popisu tunelu se pouze aktualizuje záznam v databázi. Naopak změny typu zvětšení či zmenšení MTU tunelu systém přenáší do běžící konfigurace tunelovacích serverů. K tomu využívá managementovací rozhraní jednotlivých tunelovacích serverů. PHP skript podle druhu změny komunikuje s příslušným tunelovacím serverem. Jeho adresu zjistí dotazem SQL.

4.4 Pomocné programy a skripty

V rámci vývoje systému a následnému pilotnímu projektu jsem napsal několik pomocných skriptů, které jsou určeny ke zjednodušení nasazení a provozu systému.

4.4.1 Přidělování bloků IPv6 adres tunelovacím serverům

K nasazení nového tunelovacího serveru je nutné, aby na něj byl nasměrován IPv6 prefix, který server bude dále distribuovat svým přiřazeným klientům. IPv6 prefix nebo více prefixů musí být dostatečně velké, aby je bylo možné rozdělit na více prefixů různé délky. Implementace systému je navržena na tři různé typy prefixů. Prefixy, které se používají na dvoubodové spojení virtuálního tunelu, dále automaticky přidělené prefixy a větší prefixy pro vytváření rozlehlejších IPv6 topologií.

Délku prefixů si může zvolit každý provozovatel systému individuálně. Já se v tomto textu budu držet délek 64 bitů pro první dva typy a 56 bitů pro třetí typ. K rozdělení velkých prefixů na více prefixů s menší délkou masky jsem napsal program `create_subnets`.

Program přebírá vstupní parametry a vypisuje výsledek na standardní výstup. Při spuštění program očekává čtyři parametry: *vstupní prefix*, *délku vstupního prefixu*, *délku výstupního prefixu* a *počet výstupních prefixů*. Jako příklad uvedu rozdělení vstupního IPv6 prefixu 2001:1234:5678::/48 na 200 výstupních prefixů s délkou 56 bitů:

```
$ ./create_subnets 2001:1234:5678:: 48 56 200
    2001:1234:5678::
    2001:1234:5678:100::
    2001:1234:5678:200::
    ...
```

```
2001:1234:5678:c600::
2001:1234:5678:c700::
```

Program obsahuje ošetření vstupních parametrů pro případ, kdy vstupní prefix nelze rozdělit na požadovaný počet výstupních prefixů se zadanou délkou. Program umí pracovat pouze s horními 64 bity IPv6 adresy, takže nejmenší výstupní prefix, na který je schopen dělit, je dlouhý 64 bitů. Tato vlastnost nevadí reálnému používání, protože dle zdroje [24] je doporučená délka IPv6 adres i pro dvoubodové spoje 64 bitů.

Tento program lze využít ve skriptech jako tvůrce invariantu pro inicializační plnění tabulky `ipv6_space`. Následující skript psaný v jazyce BASH slouží k naplní tabulky `ipv6_space` První čtyři parametry předává programu `create_subnets`, další parametr slouží k určení databázového ID tunelovacího serveru. Poslední parametr určuje hodnotu vloženého atributu *type*.

```
#!/bin/bash

BIN="cesta_k_adresari_programu"

HOST="mysql_server"
PORT="mysql_port"
USER="databazovy_uzivatel"
PASS="databazove_heslo"
DB="jmeno_databaze"

for prefix in `${BIN}/create_subnets $1 $2 $3 $4`; do
    echo $prefix;
    mysql -h $HOST -P $PORT -u $USER -p$PASS $DB -e \
        "INSERT INTO ipv6_space (id_pop, prefix, prefix_length, type) \
         VALUES ( $5, \"$prefix\", $3, $6)"
done
```

Shrnutí

Tato kapitola rozebrala implementaci jednotlivých částí systému. Hlavně se zaměřila na tunelovací server AYIYA 4.2, kde byly popsány detailně jeho funkční bloky. Také zde bylo uvedeno, jaké postupy tunelovací program vykonává při standardních situacích, jako např. ověření integrity zprávy.

Kapitola 5

Bezpečnost

V této kapitole rozeberu několik druhů útoků na některé části systému a ukážu, jak systém reaguje.

První část bude věnována službě TIC, kde se podívám na možnost oklamání klienta TIC ke komunikaci s nepravým serverem TIC a možné dopady odposlouchávání komunikace TIC.

Druhá část popisuje teoretické útoky na tunelovací sever. Konkrétně se jedná o modifikaci komunikace AYIYA útočníkem a útoky na zahlcení databázového spojení.

5.1 Podvržení serveru TIC

Jedním z klasických útoků na klienta je podvržení služby. Útočník musí klienta oklamat a donutit jej použít své podvržené služby. Možností, jak klienta donutit používat podvrženou službu, je více. Útočník může mít k dispozici prostředky, kterými změní překlad DNS požadavků klienta ve svůj prospěch nebo může ovlivnit komunikaci mezi klientem a serverem tak, aby provoz procházel přes či končil na jím kontrolovaném zařízení. Existuje více možností, jak může útočník podvržení serveru dosáhnout.

Příklad komunikace TIC je uveden v části 2.3.4. Šifrování komunikace pomocí zabezpečené vrstvy TLS je v tomto případě zbytečné, protože útočník donutí klienta komunikovat se svým softwarem a tudíž pokud dojde k aktivaci TLS, útočník bude implicitně rozšifrovávat veškerou komunikaci.

5.1.1 Útok na podvržení vlastního tunelovacího serveru

Klient a server sdílejí pro autentizaci protokolu TIC tajné heslo. Autentizace je jednosměrná, a to autentizace klienta. Celý proces funguje tak, že server vygeneruje náhodný řetězec, který pošle klientovi. Klient k němu připojí kontrolní součet sdíleného hesla, vypočítá kontrolní součet nově vzniklého řetězce a pošle jej zpátky serveru. Server stejným způsobem vypočítá svou verzi kontrolního součtu a mezi sebou je porovná. Pokud jsou zprávy shodné, server klienta autentizoval.

Útočníkova verze služby ovšem může autentizovat klienta i bez znalosti sdíleného tajemství. Prostě přijme jakoukoli klientovu odpověď za správnou. Klient se nachází ve stavu, kdy zjišťuje provozní informace o tunelu od serveru. Útočník je tímto způsobem schopen klientovi předat podvržené informace a navést jej k používání svého tunelovacího serveru. Tento druh útoku se útočníkovi vyplatí provádět pouze pokud chce získat přístup k datovým tokům uživatele. V tomto příkladu je to situace, kdy útočník podvrhl svůj server pomocí

špatného překladu DNS. V situaci, kdy se může útočník dostat ke klientovým datům, je pro něj tento útok bez užitku.

5.1.2 Útok na získání tajného hesla

Užitečnější pro útočníka by bylo zjištění sdíleného tajemství k komunikaci protokolu TIC. Pak by mohl útočník vystupovat za legitimního klienta a měl by možnost modifikovat jeho tunelované zprávy. Zde by se dal útok založit na dnes známé slabině metody kontrolního součtu MD5. Jen nastíním situaci, kdy útočníkův server pošle klientovi řetězec náchylný na MD5 kolize jako autentizační zprávu. Tomuto riziku se lze vyhnout použitím jiného algoritmu pro výpočet kontrolních součtů. Protokol TIC podporuje zatím bezpečnou metodu SHA1. Problémem je, že klient AICCU používá standardně pro autentizaci TIC protokolu metodu MD5 a nelze jeho chování ovlivnit žádnou konfigurační direktivou. Řešením by bylo přepracovat klienta AICCU tak, aby ke kontrolnímu součtu používal metodu SHA1.

5.2 Odposlouchávání komunikace protokolu TIC

Ve chvíli, kdy má útočník možnost pasivně odposlouchávat klientovu komunikaci, může z komunikace TIC vyčíst tajné heslo k ověření integrity přenášených tunelovaných paketů a může se vydávat za klienta. Navíc pokud by útočník posílal validní tunelované zprávy AYIYA tunelovacímu serveru z jiné adresy a portu než legitimní klient, server by si je přenastavil a tím validního klienta odstřihl od paketů proudících k němu. Komunikující klient i útočník by se tímto způsobem mohli hádat, komu budou chodit pakety z Internetu. Útočník může touto cestou tunel klientovi téměř znepřístupnit.

Jednoduchou obranou proti odposlouchávání komunikace je zapnout šifrování vrstvou TLS. Výchozí konfigurace AICCU klienta podporu TLS na server nevyžaduje, ale pokud je dostupná, aktivuje ji. Má implementace služby TIC má na rozdíl od implementace SixXS podporu pro TLS šifrování a klienti ji mohou používat.

5.3 Podvržení paketů AYIYA

V minulé části jsem popsal některé útoky týkají se serveru TIC. Nyní se podívám na možnosti útoků a zabezpečení tunelovacího serveru. Nejedná se o útoky, kde útočník zná tajné heslo k tunelu.

5.4 Přetížení databázového serveru

V situaci, kdy by útočník chtěl poškodit tunelovací server, mohl by na něj posílat zprávy AYIYA s neexistující identitou. Pokud by útočník používal jednu identitu, umí se s tím server vyrovnat velmi dobře. Procedura, která ověřuje pravost identity, je nejnáročnější část zpracování zprávy AYIYA. Proto je systém navržen tak, aby si již jednou ověřené identity pamatoval po zvolenou dobu i v případě, kdy se jedná o neexistující identity.

Tunelovací server tímto způsobem zabraňuje přetížení databázového serveru například v případech, kdy na server přichází velké množství zpráv AYIYA s jednou nebo několika neexistujícími identitami. Serverová implementace je vhodná i v případě, kdy dojde například k zakázání velmi aktivního tunelu, kterým teče proud dat UDP směrem do Internetu.

Zakázaný tunel se na serveru po chvíli smaže, avšak klient svůj stav nezmění do restartování klientského programu AICCU. Tunelovaný proud dat považuje server za nepovolený a zahazuje ho pouze s jedním ověřením identity za daný časový interval.

5.5 DOS útok na tunelovací server

Tunelovací protokol AYIYA používá k přenosu zpráv nestavový transportní protokol UDP. Proto na tunelovací server nelze provádět útoky typu *SYN flood*, které jsou možné pouze u protokolu TCP. Podobnou logiku jako u zmíněného útoku lze ovšem použít na aplikační vrstvě. Útočník může zasílat na server nepřeberné množství paketů AYIYA pokaždé s jinou identitou. Takové počínání by mohlo skončit přetížením tunelovacího serveru, který by se snažil ověřit každou novou identitu. Pro všechny identity by navíc program vytvářel záznam v lineárním seznamu v paměti. Nároky na jeden záznam nejsou velké a činí necelých 300 bajtů. Záznam bude v paměti programu jen po určitou dobu, takže k z hlediska paměti může server odolávat velmi dlouho.

Čím více bude záznamů v lineárním seznamu v paměti, tím delší dobu bude trvat jeho prohledání. Horší situace nastává u ověřování nových identit v databázi, kdy na jeden paket AYIYA s unikátní identitou je nutné komunikovat s databází ve více než jedné zprávě. V případě, že nemáme omezenu rychlost toku dat, které mohou na tunelovací server mířit, může dojít k přetížení komunikace s centrální databází. Z důvodu přetížení databáze by se nedostupnost tunelovacího serveru ovšem neměla projevit pro již připojené klienty, pro jejichž pakety není potřeba do databáze přistupovat.

DOS útok s různými identitami jsem nezkoušel, takže je možné, že dojde k nedostupnosti tunelovacího serveru i pro připojené klienty. Například v případě, kdy všechna vlákna, která obsluhují příchozí pakety AYIYA, budou čekat na zatíženou databázi.

Proti DOS útoku nepomůže ani omezení počtu identit na jednu zdrojovou IP adresu, protože u nestavového UDP protokolu může útočník jednoduše měnit zdrojovou adresu, a tak oklamat server.

Omezením příchozí rychlosti tunelovaných paketů by se také situace příliš nezlepšila. Nemohlo by sice docházet k velkému přetěžování databázové či jiné části systému, ale klienti využívající tunelovacího serveru by stejně měli službu značně omezenou či téměř nepoužitelnou v důsledku omezení rychlosti a zahazování paketů.

Shrnutí

Tato kapitola popsala několik druhů útoků na služby tunel broker systému včetně útoků na monitorování datových toků klientů. U jednotlivých částí jsou popsány teoretické předpoklady, za kterých by byl útok realizovatelný. Dále text řeší otázku vylepšení vlastností systému tak, aby k bezpečnostním incidentům nedocházelo. Naopak, např. u DOS útoku dnes nelze bezpečně oddělit legitimní provoz od podvrhů.

Kapitola 6

Testování

Důležitou částí v rámci vývoje systému bylo testování jednotlivých částí systému i jejich vzájemná kooperace. Tato kapitola blíže rozebírá testy týkající se propustnosti, směrování, nastavení parametru tunelu MTU. Testování se zaměří i na zpoždění komunikace přes tunelovací server a podívá se i na interakci webového rozhraní s uživatelem.

6.1 Testování funkčnosti tunelovaného připojení

Popis testu: Test se skládal z vyzkoušení dostupnosti různých služeb v síti IPv6. Jako reprezentativní služby jsem si vybral webové servery www.nic.cz a www.nix.cz. Dále jsem testoval protokol FTP listováním a stahováním dat ze serveru ftp.fit.vutbr.cz. Abych do testu zahrnul i jiný transportní protokol než TCP, zkoušel jsem posílat ICMPv6 zprávy na uvedené servery.

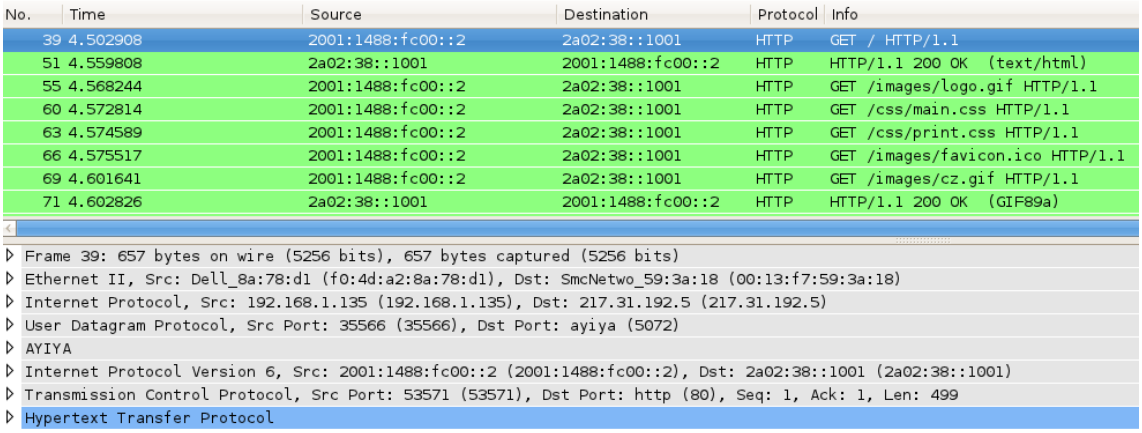
Cíl testu: Ověření, zda tunelované připojení umožňuje přenášet různé transportní protokoly a aplikační data síťovým protokolem IPv6.

Výsledky testu: Všechny zkoušené přenosy protokolem IPv6 proběhly v pořádku. Pomocí programu Wireshark jsem datové přenosy kontroloval. Obrázek 6.1 zachycuje část komunikace s webovým serverem. Detailně je vidět vrstvení protokolů od linkových, přes nosný protokol IPv4, tunelovací protokol AYIYA a tunelovaný přenosný protokol TCP/IPv6 a aplikační protokol HTTP. Horní část obrázku ukazuje několik požadavků HTTP a odpovědí realizovaných přes protokol IPv6.

Zhodnocení: Test dopadl úspěšně.

6.2 Rozmezí velikosti MTU tunelu

Popis testu: Velikost MTU tunelu ovlivňuje propustnost a v případě přenosu větších bloků dat se data fragmentují do paketů o velikosti v závislosti na MTU. Test zjišťoval minimální a maximální hranici, kterou je možno tunelu nastavit. Minimální velikost vychází z [13], jenž definuje pro linkovou vrstvu minimální MTU na 1280 Bajtů. Horní hranice se odvozuje od MTU protokolu Ethernet (1500 B) odečtením režie tunelu. Režie tunelu se skládá z velikosti hlaviček protokolu IP (20 B), UDP (8 B) a AYIYA (44 B). Maximální teoretická MTU tunelu AYIYA je tedy $1500 - 72 = 1428$ Bajtů.



Obrázek 6.1: Vrstvový model reálné HTTP komunikace s serverem IPv6.

Cíl testu: Cílem testu bylo ověřit, zda tunel bude fungovat s různými hodnotami MTU od minimální až po maximální.

Výsledky testu: Při nastavení minimálního MTU tunel funguje bez problémů. U maximálního možného MTU jsem narazil na problém na lince typu ADSL, která má hodnotu MTU sníženou o režii protokolu PPP (2 B) a PPPoE (6 B). Stabilního přenosu jsem docílil až nastavení MTU na hodnotu 1420 Bajtů.

Zhodnocení: Změna MTU tunelu funguje v pořádku. Doporučení pro uživatele systému je následující. Pokud uživatel neví přesně, co dělá, je vhodné nechat hodnotu MTU na výchozích 1280 Bajtech.

6.3 Směrování subnetů

Popis testu: Klient od systému dostává ke každému vytvořenému tunelu prefix IPv6 pro tvorbu vlastní sítě. Aby klient mohl prefix směrovat, musí svůj systém, na kterém má nainstalován klienta AICCU zprovoznit jako router IPv6. V případě Linuxového systému stačí zapsat hodnotu 1 do souboru `/proc/sys/net/ipv6/conf/all/forwarding`. Adresu z prefixu pak nastaví na některé rozhraní a na stejné síti nastaví adresu a výchozí bránu na ostatních počítačích nebo síťových zařízeních. Zařízení by s touto konfigurací by měla být schopna komunikovat s IPv6 Internetem.

Cíl testu: Cílem testu bylo ověřit směrování prefixů IPv6 na tunelovacím serveru.

Výsledky testu: Podle jednoduchého postupu uvedeného v popisu testu jsem připojil a ověřil funkčnost směrování prefixů IPv6. Navíc jsem pomocí programu `radvd` zprovoznil bezstavovou konfiguraci klientů a vše fungovalo, jak mělo.

Zhodnocení: Test dopadl úspěšně.

6.4 Testování více tunelů současně

Popis testu: Systém umožňuje uživatelům se bez omezení zaregistrovat a vytvářet tunely. Tunelovací server musí umět všechny tyto tunely provozovat současně. Pro tento test

jsem si v systému vytvořil 5 instancí tunelů a nainstaloval je na pět různých počítačů a síťových zařízení. Při testu jsem se snažil použít více operačních systémů, jako systémy Windows, Linux a speciální distribuci OpenWRT, která je určena pro malé domácí routery.

Cíl testu: Ověření, zda je tunelovací server schopen provozovat více tunelů současně. Dalším cílem bylo vyzkoušet vzájemnou komunikaci připojených klientů k jednomu tunelovacímu serveru.

Výsledky testu: Klienti byli schopni současně provozovat své tunely. Propustnost tunelů jsem prověřoval pomocí zpráv ICMPv6, které klienti posílali na libovolné IPv6 cíle. Navíc jsem na tunelovacím serveru kontroloval výpis virtuálních rozhraní a sledoval logovací zprávy tunelovacího programu.

Zhodnocení: Test dopadl úspěšně. Klienti byli schopni komunikace nejen se servery na IPv6 Internetu, ale byli schopni komunikovat i mezi sebou.

6.5 Testování propustnosti tunelovacího serveru

Popis testu: Propustnost tunelovacího serveru je zajímavý údaj, který provozovatel může vzít do úvahy v plánování kapacit celého systému.

Konfiguraci klientů z minulého testu jsem využil i pro testování propustnosti. Tunelovací server byl k Internetu připojen pomocí technologie Ethernet s rychlostí 1Gbps a byl provozován jako virtuální server s výkonem jednoho procesoru AMD na frekvenci 1800MHz a 512MB paměti RAM. Klienti byli připojeni pomalejšími linkami jako ADSL. Dva klienti dosahovali rychlosti připojení 100Mbps a 1Gbps. Abych mohl propustnost prověřit, zvolil jsem pro přenos dat obraz DVD Linuxové distribuce Knoppix, který mi při velikosti 3.6 GB poskytoval dostatečný čas stahování pro pozorování vytížení serveru.

Cíl testu: Zjistit propustnost tunelovacího programu na virtuálním stroji v laboratoři při konfiguraci uvedené v popisu testu.

Výsledky testu: Propustnost specifikovaného tunelovacího serveru byla přibližně 150 Mbps při 90% vytížení procesoru serveru. Hodnoty jsem zjistil pomocí programů `top` a `iptraf-ng`.

Zhodnocení: Tento výsledek považuji za velice dobrý, protože z vlastností tunelovacího protokolu AYIYA plyne nutnost vypočítat kontrolní součet každého paketu, což z ní činí výpočetně náročný proces. Bohužel, nemám možnost srovnání s jinými tunelovacími metodami, protože jsem žádnou jinou neimplementoval, ani jsem obdobný systém nenasazoval.

6.6 Latence tunelovaného připojení

Popis testu: Používání tunelu obecně zvedá latenci komunikace proti nativní konektivitě. Princip je jednoduchý – né vždy jdou tunelovaná data stejným směrem, jako by šla nativní cestou. Se vzdáleností tunelovacího serveru od klienta se tato latence zvedá. Test jsem provedl ve třech lokalitách. První je v Brně a disponuje nativní IPv4 i IPv6

Destinace	Klient1			Klient2		Klient3	
	T6	**N6**	**N4**	**T6**	**N4**	**T6**	**N4**
Klient1	—	—	—	15,024	2,980	62,501	54,634
Klient2	15,010	14,512	2,720	—	—	71,608	89,572
Klient3	84,958	107,079	44,822	74,572	45,731	—	—
Server1	4,603	4,088	4,308	10,428	10,275	55,243	89,154
Server2	5,590	3,800	4,255	11,200	10,288	51,862	43,925
Server3	172,703	173,807	180,370	180,148	190,710	221,058	192,075

Tabulka 6.1: Tabulka s průměrnými naměřenými časy. Zkratka **T6** označuje tunelovanou IPv6, **N6** označuje nativní IPv6 a **N4** označuje nativní IPv4 konektivitu. **Klient1** se nachází v Brně, **Klient2** v Bratislavě a **Klient3** v Prištině. **Server1** je označení pro tunelovací server v Praze, **Server2** pro portál seznam.cz taktéž v Praze a **Server3** pro službu whatismyipv6.com, která sídlí v USA.

konektivitou. Další dvě lokality nedisponují nativní IPv6 konektivitou a jsou umístěny v Bratislavě a Prištině (hlavní město Kosova). Kosovskou lokalitu jsem zvolil z důvodu relativně vzdálené polohy od tunelovacího serveru, který je v Praze.

Cíl testu: Vyzkoušet, jak se zvedne latence tunelované komunikace s různými servery oproti nativní konektivitě IPv6 a IPv4 v závislosti na vzdálenosti klienta od tunelovacího serveru a cílového serveru. Všechny zvolené servery jsou dostupné jak z IPv4, tak i IPv6 cestou.

Výsledky testu: Výsledek testu je zanesen v tabulce 6.1, kde jsou průměrné hodnoty 20 ICMP zpráv. Z tabulky je vidět, že tunelovací server přidává k cestě mezi dvěma klienty přibližně 0.5 až 1.5 ms zpoždění. Podle vzdálenosti a nastavení směrovacích protokolů na Internetu se řídí i zpoždění mezi jednotlivými destinacemi. V případě spojení Klienta1 a Klienta3, je vidět, že tunelované připojení je rychlejší. Může za to vhodnější vyústění tunelu v IPv6 síti z pohledu globálního směrování.

Zhodnocení: Test potvrdil předpokládané výsledky. Zpoždění tunelovacím serverem se pohybuje mezi 0,5 až 1,5 ms. Jeden z testů ukázal, že tunelovaná konektivita může odezvu i zlepšit. Nicméně, takové zlepšení latence může nastat pouze v ojedinělých případech.

6.7 Testování kontroly integrity tunelovaných paketů

Popis testu: Tunelovací server má za úkol zkontrolovat integritu každého přijatého tunelovaného paketu. Kontrolu integrity samozřejmě provádí jen v případě, pokud zná klienta, který paket odeslal. Test jsem realizoval pomocí odposlechu komunikace pravého klienta, upravením hodnoty jednoho bajtu v těle zprávy a odesláním na server. Zkoušel jsem také u pozměněného paketu změnit odesílací IP adresu a port.

Cíl testu: Ověřit, zda tunelovací server korektně kontroluje integritu paketů a zda nepropustí podvržený paket či neovlivní další komunikaci se skutečným klientem.

Výsledky testu: U pozměněného paketu tunelovací server správně rozpoznal, že byl paket modifikován a zahodil jej. Změna adresy odesilatele pozměněného paketu také

neovlivnila žádné datové struktury tunelovacího programu, protože byl paket zahozen dříve, než k nim mohlo dojít.

Zhodnocení: Tunelovací server se chová správně, podle specifikace.

6.8 Test vstupů webového rozhraní

Popis testu: Uživatelé webového rozhraní jsou velice vynalézaví co se týče používání webových formulářů určených pro interakci uživatelů se systémem.

Cíl testu: Odhalit nedostatky formulářů webového rozhraní. Test probíhal v rámci pilotního provozu systému v laboratořích CZ.NIC.

Výsledky testu: V průběhu testu se odhalila chyba při vyhodnocování formulářů v SQL dotazech. Pokud vyplněné textové pole obsahovalo znak apostrof nebo uvozovky, byl tento řetězec špatně interpretován a způsobil chybu SQL dotazu. Z tohoto chování vznikaly v systému neinicializované tunely. Problém vyřešilo použití funkce PHP `mysql_real_escape_string()`, která nevhodné znaky v SQL dotazech vloží jako escape sekvenci.

Zhodnocení: Během testu se odhalila chyba zpracování textových polí formulářů. Tato chyba byla následně opravena.

Shrnutí

Tato kapitola rozebrala několik testů, které jsem na systému prováděl. Konkrétně se jednalo o testy funkčnosti tunelovaného připojení a o testy zjišťující latenci a propustnost tunelovacího serveru. Nakonec jsem testoval odolnost webového rozhraní vůči zadávaným hodnotám od uživatelů systému.

Výsledky byly velmi uspokojivé. Všechny testy, které testovaly funkčnost tunelovaného připojení potvrdily správnou činnost systému včetně směrování prefixů IPv6 na klienta. Propustnost jednoho tunelovacího serveru v konfiguraci virtuálního stroje s 512 MB paměti a procesorem s frekvencí 1800 MHz byla experimentálně určena na 150 Mbps. Latence zpracování tunelovacím programem na stejném hardware se pohybuje mezi 0,5 a 1,5 ms. Testy webového rozhraní objevily chybu zpracování formulářů, kterou jsem následně opravil.

Kapitola 7

Závěr

Výsledkem práce je přehled protokolů IPv6, AYIYA, TSP a TIC, které jsem využil jako vědomostní základ pro návrh a implementaci serverové části tunelovacího systému kompatibilního s klienty AICCU.

Implementovaný tunel broker systém jsem otestoval a je reálně nasazen v pilotním provozu v rámci výzkumných aktivit laboratoří registrátora české národní domény, CZ.NIC. Webové rozhraní broker serveru můžete nalézt na adrese [6].

Tato diplomová práce získala 2. místo na studenské konferenci a soutěži EEICT 2011 v kategorii Počítačové systémy.

V průběhu zpracovávání jsem si zopakoval a rozšířil znalosti o síťovém protokolu IPv6, podrobně jsem se seznámil s metodou tunelování AYIYA a prostudoval jsem protokol TIC. V návrhu tunel broker systému jsem zohlednil znalosti získané studiem výše zmíněných protokolů. V implementační části jsem se naučil mimo jiné používat virtuální síťová rozhraní operačního systému Linux. Testovací fáze se částečně překrývala s vývojem jednotlivých částí systému. Oživil jsem si používání aplikací pro sledování zatížení systému. Promyslel jsem a popsal aspekty některých druhů útoku. Také jsem některé útoky reálně vyzkoušel.

Budoucí rozšíření systému spatřuji v integraci dalších metod tunelování jako například proto-41 a ve vytvoření univerzálního rozhraní pro komunikaci s různými klienty pomocí protokolu TSP pro řízení tunelu.

Literatura

[1] Assigned Internet Protocol Numbers.
URL http://www.iana.org/assignments/protocol-numbers

[2] Automatic IPv6 Connectivity Client Utility.
URL http://www.sixxs.net/tools/aiccu

[3] GnuTLS Manual.
URL http://www.gnu.org/software/gnutls/manual

[4] IPv4 Address Report.
URL http://www.potaroo.net/tools/ipv4

[5] Tunnel Information and Control protocol.
URL http://www.sixxs.net/tools/tic

[6] Tunserv - Tunel Broker systém.
URL http://ayiya.labs.nic.cz

[7] VTUN - Virtual Tunnels over TCP/IP networks.
URL http://vtun.sourceforge.net

[8] Aoun, C.; Davies, E.: Reasons to Move the Network Address Translator - Protocol Translator (NAT-PT) to Historic Status. 2007.
URL http://www.faqs.org/rfcs/rfc4966.html

[9] Bagnulo, M.; Sullivan, A.; Matthews, P.: DNS64: DNS Extensions for Network Address Translation from IPv6 Clients to IPv4 Servers. 2011.
URL http://tools.ietf.org/html/rfc6147

[10] Bao, C.; Huitema, C.; Bagnulo, M.; aj.: IPv6 Addressing of IPv4/IPv6 Translators. 2010.
URL http://www.faqs.org/rfcs/rfc6052.html

[11] Blanchet, M.; Parent, F.: IPv6 Tunnel Broker with the Tunnel Setup Protocol (TSP). 2010.
URL http://www.faqs.org/rfcs/rfc5572.html

[12] Carpenter, B.; Moore, K.: Connection of IPv6 Domains via IPv4 Clouds. 2001.
URL http://www.faqs.org/rfcs/rfc3056.html

[13] Deering, S.; Hinden, R.: The IP Network Address Translator (NAT). 1998.
URL http://www.faqs.org/rfcs/rfc2460.html

[14] Droms, R.; Bound, J.; Volz, B.; aj.: Dynamic Host Configuration Protocol for IPv6 (DHCPv6). 2003.
URL http://www.faqs.org/rfcs/rfc3315.html

[15] Durand, A.; Fasano, P.; Guardini, I.; aj.: IPv6 Tunnel Broker. 2001.
URL http://www.faqs.org/rfcs/rfc3053.html

[16] Egevang, K.; Francis, P.: The IP Network Address Translator (NAT). Květen 1994.
URL http://www.faqs.org/rfcs/rfc1631.html

[17] Gilligan, R.; Nordmark, E.: Transition Mechanisms for IPv6 Hosts and Routers. 1996.
URL http://www.faqs.org/rfcs/rfc1933.html

[18] Hagino, J.; Yamamoto, K.: An IPv6-to-IPv4 Transport Relay Translator. 2001.
URL http://www.faqs.org/rfcs/rfc3142.html

[19] Hinden, R.; Deering, S.: IP Version 6 Addressing Architecture. 2006.
URL http://www.faqs.org/rfcs/rfc4291.html

[20] Massar, J.: AYIYA: Anything In Anything. 2004.
URL http://unfix.org/~jeroen/archive/drafts/draft-massar-v6ops-ayiya-02.html

[21] Melnikov, A.; Zeilenga, K.: Simple Authentication and Security Layer (SASL). 2006.
URL http://www.faqs.org/rfcs/rfc4422.html

[22] Narten, T.; Nordmark, E.; Simpson, W.; aj.: Neighbor Discovery for IP version 6 (IPv6). 2007.
URL http://www.faqs.org/rfcs/rfc4861.html

[23] Nordmark, E.: Stateless IP/ICMP Translation Algorithm (SIIT). 2000.
URL http://www.faqs.org/rfcs/rfc2765.html

[24] Popoviciu, C.; Chown, T.; Bonness, O.; aj.: IPv6 Unicast Address Assignment Considerations. 2008.
URL http://www.faqs.org/rfcs/rfc5375.html

[25] Satrapa, P.: *IPv6*. CZ.NIC, z. s. p. o., 2008, ISBN 978-80-904248-0-7.

[26] Thomson, S.; Narten, T.; Jinmei, T.: IPv6 Stateless Address Autoconfiguration. 2007.
URL http://www.faqs.org/rfcs/rfc4862.html

[27] Topolcic, C.: Experimental Internet Stream Protocol, Version 2 (ST-II). 1990.
URL http://www.faqs.org/rfcs/rfc1190.html

[28] Tsirtsis, G.; Srisuresh, P.: Network Address Translation - Protocol Translation (NAT-PT). 2000.
URL http://www.faqs.org/rfcs/rfc2766.html

[29] Tsuchiya, K.; Higuchi, H.; Atarashi, Y.: Dual Stack Hosts using the 'Bump-In-the-Stack' Technique (BIS). 2000.
URL http://www.faqs.org/rfcs/rfc2767.html

Příloha A

Ukázky kódu

A.1 Použití funkce writev()

Funkce `tun_write()` dostává parametrem popisovač souboru (tun rozhraní) *fd*, ukazatel na tunelovaná data *buf*, která začínají IPv6 hlavičkou a délku těchto dat, *len*. Funkce vytvoří a inicializuje strukturu *tun_pi*, která obsahuje informace o tom, jaká hlavička protokolu následuje. Pole dvou struktur *iovec* naplní dle očekávání.

```
#include <sys/uio.h>
#include <if_tun.h>

int tun_write(int fd, const char * buf, unsigned int len) {

    struct tun_pi   pi = {
        .flags = 0,
        .proto = htons(ETH_P_IPV6)
    };

    struct iovec dat[2] = { {
            .iov_base = &pi,
            .iov_len = sizeof(pi)
        }, {
            .iov_base = buf,
            .iov_len = len
        }
    };

    return writev(fd, dat, 2);
}
```

Příloha B

Návod na instalaci klienta AICCU a zprovoznění tunelu se systémem provozovaným sdružením CZ.NIC

Návod na instalaci AICCU klienta a zprovoznění s testovaným systémem tunserv.

1. **Registrace na webu**
   Pokud ještě nemáte vytvořený profil, zaregistrujte se na stránkách `http://ayiya.labs.nic.cz/tunserv/`.

2. **Vytvoření tunelu na webu**
   Po přihlášení do systému v záložce *Tunel* vytvořte nový tunel pomocí tlačítka *nový tunel*. Do formuláře vyplňte plánované místo ukončení tunelu a popis účelu tunelu. S vytvořeným tunelem automaticky dostáváte jeden prefix IPv6 /64 nasměrovaný na váš konec tunelu. Tento prefix můžete použít pro vaši síť za koncem tunelu.

3. **Instalace klienta AICCU**
   Nainstalujte si klienta aiccu. Zkuste instalaci pomocí vašeho balíčkovacího systému. Debian/Ubuntu: `sudo apt-get install aiccu`. Pokud AICCU nenaleznete, stáhněte si jej ze stránek SixXS http://www.sixxs.net/tools/aiccu/ . Uživatelé systému Windows použijí verzi Console a nainstalují si příslušné ovladače ze zmíněné URL.

4. **Nastavení AICCU**
   Konfigurační soubor klienta AICCU obsahuje direktivy potřebné pro navázání tunelu: *username*, *password* a *server*. Všechny parametry zjistíte na webu pomocí odkazu *Detail* v tabulce vašich tunelů. Jsou to konkrétně *TIC Login*, *TIC Heslo* a *TIC Server*. Zjištěné údaje přepište do konfiguračního souboru.

   Direktiva *daemonize true* navíc zajistí, že se klient spustí na pozadí. Konfigurační soubor může vypadat například takto:

   ```
   username zjištěné_jméno
   password zjištěné_heslo
   server tic.labs.nic.cz
   daemonize true
   ```

   Poznámka Windows: AICCU očekává konfigurační soubor na místě C:\Windows\aiccu.conf.

5. **Zapnutí klienta**
   Podle vašeho OS, spustťe službu aiccu. např. pomocí init skriptu:

   ```
   /etc/init.d/aiccu start
   ```

   nebo spuštění binárky:

   ```
   aiccu start cesta_ke_konfiguračnímu_souboru
   ```

   Poznámky Windows: Při prvním spuštění klienta aiccu musíte mít správcovská oprávnění. (Platí pro Win Vista, Win 7).

   Windows XP v základu nemají podporu protokolu IPv6. Je ji potřeba nainstalovat. Postup naleznete na https://www.ipv6.cz/MS_Windows_XP. Pokud máte konfigurační soubor umístěn v C:\Windows\aiccu.conf, tak stačí ke spuštění příkaz `aiccu.exe start`, kde `aiccu.exe` je binární soubor verze Console stažený ze stránek sixxs.net viz bod 3.

6. **Ověření funkčnosti tunelu**
   Vypište si síťová rozhraní, pokud v nich najdete nové rozhraní, např. se jménem *aiccu* a IPv6 adresou, zkuste pomocí programu `ping` odezvu na adresu nic.cz

   Linux: `$ ping6 nic.cz`

   Windows: `C:\> ping -6 nic.cz`

   Pokud máte nativní konektivitu IPv6, lze AYIYA tunel otestovat příkazem:

   Linux: `$ ping6 -n -I <IPv6 adresa Vašeho konce tunelu> nic.cz`

7. **Když jeden prefix nestačí**
   Na webu v záložce *Subnet* si lze ke každému tunelu přidělit ještě jeden prefix o velikosti /56 bitů. Prefix se stává aktivním hned po vytvoření.

8. **Při potížích**
   Prostudujte hlášky klienta AICCU, které zapisuje do systémového logu, nebo případně přímo do terminálu, ze kterého jste program spustili.

   Užitečná je také volba konfiguračního souboru *verbose true*.

Příloha C

Obsah CD

Technická zpráva

Přiložené CD obsahuje zprávu ve formátu PDF i ve formě zdrojových kódů. Pro sazbu technické zprávy jsem využil oficiální šablonu FIT pro LaTeX. Zpráva je na CD umístěná v adresáři `zprava/` a její zdrojové kódy v adresáři `zprava/tex/`. Hlavní zdrojový soubor s obsahem zprávy je `obsah.tex`. Zprávu lze ze zdrojových kódů přeložit pomocí nachystaného *Makefile*, který pro překlad používá program `pdflatex`.

Zdrojové kódy vytvořených programů

Všechny vytvořené programy jsem napsal v jazyce C a lze je přeložit překladačem GNU GCC. Zdrojové soubory se nachází v adresáři `tunserv/src/`. Pro automatický překlad je připraven soubor *Makefile*. Výstupem překladu jsou binární soubory v adresáři `tunserv/bin/`.

Zdrojové kódy webového rozhraní jsou umístěny v adresáři `tunserv/www/`. Webové rozhraní bylo vyzkoušeno v prostředí OS Ubuntu s serverem HTTP Apache2 a interpretem PHP-5.3. Další informace jsou umístěny v souboru *README*.